Performance Isolation of a Misbehaving Virtual Machine with Xen, VMware and Solaris Containers

Todd Deshane, Demetrios Dimatos, Gary Hamilton, Madhujith Hapuarachchi, Wenjin Hu, Michael McCabe, Jeanna Neefe Matthews
Clarkson University
{deshantm, dimatosd, hamiltgr, hapuarmg , huwj, mccabemt, jnm}@clarkson.edu

Abstract

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. 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 – the target environment for some 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 customers to want a certain guaranteed level of performance regardless of the actions taken by other VMs on the same physical host. In that light, another key aspect of the comparison between virtualization environments has received less attention - how well do different virtualization systems protect VMs from misbehavior or resource hogging on other VMs? In this paper, we present the results of running a variety of different misbehaving applications under three different virtualization environments VMware, Xen, and Solaris containers. These are each examples of a larger class of virtualization techniques namely full virtualization, paravirtualization and generic operating systems with additional isolation layers. To test the isolation properties of these systems, we run six different stress tests - a fork bomb, a test that consumes a large amount of memory, a CPU intensive test, a test that runs 10 threads of IOzone and two tests that send and receive a large amount of network I/O. Overall, we find that VMware protects the well-behaved virtual machines under all stress tests, but sometimes shows a greater performance degradation for the misbehaving VM. Xen protects the well-behaved virtual machines for all stress tests except the disk I/O intensive one. For Solaris containers, the well-behaved VMs suffer the same fate as the misbehaving one for all tests.

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. However, in a commercial hosting environment, there is another important aspect to the comparison – 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 examine three virtualization environments – Xen, VMware and Solaris containers. On each, we host 4 web servers – each in their own virtual machine. After establishing some baseline data, we run a variety of different stress tests in one of the four virtual machines and report the impact on SPECweb perform-

ance in all four virtual machines. In each case, we ask – what was the impact on the misbehaving VM and what was the impact on the other three well-behaved VMs. From this we determine how well the system isolated the virtual machines from each other.

In particular, we run a memory intensive test, a fork bomb, a CPU intensive test, a disk intensive test and two network intensive tests. We find that the answer to these questions varies significantly with the type of stress test. We also find that the results seem to highlight the difference between full virtualization systems like VMware, paravirtualization systems like Xen, and general purpose operating systems that are retrofitted to support namespace and resource isolation like Solaris containers. Mark Fiuczynski from Princeton has called the latter paenevirtualization systems (paene is Latin for nearly) [7].

Each of these environments could implement a high degree of resource isolation. There is nothing preventing strong resource isolation in any of these environments. However, one might naturally expect the full virtualization system to have the highest degree of resource isolation followed by paravirtualization systems followed by paenevirtualization systems. Our results do generally validate this expectation. VMware completely protects the well-behaved VMs for all of our stress tests. With Xen, the well-behaved VMs suffer a significant impact only for the disk intensive test. However, there is a slight but repeatable degradation on other tests. With Solaris containers, the well-behaved containers shared the fate of the misbehaving containers in all cases.

2. 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 our SPECweb tests, the clients were also IBM Thinkcentres. We used a different physical client to connect to each server virtual machine. Unless otherwise noted, all 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.

3. 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.

3.1. Memory Consumption

We began with a stress test which simply loops constantly allocating and touching memory. After this test, none of the Solaris containers presented any results. It effectively shuts down all servers in the machine. 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. Fork Bomb

We also ran a program that loops, creating new child processes. As in the memory consumption test, under both Xen and VMware, the misbehaving virtual machine presented no results. For both Xen and VMware, 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

[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

not reported for any of the four containers. In the second case, results were reported, but the good response rate averages 10% across all four containers with the misbehaving container actually showing the best results with 12% good response.

3.3. CPU Intensive Test

Our third test stressed 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%.

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.

3.4. 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]

3.5. 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.

3.5.1. Server Transmits Data

For the server transmitting 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%.

3.5.2. Server Receives Data

Finally, for the server transmitting test, we started 4 threads which each constantly read 60K packets over UDP from external receivers.

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.

---

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.

	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 1: 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 1. 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.

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.

4. Conclusions

Strong resource isolation could be implemented in any one of the systems we have studied, but in many ways it is a hard problem. It means extending resource scheduling and the idea of fairness to all resources: disk, network (incoming and outgoing), memory, CPU, etc. While any of these three models could provide the necessary resource isolation, it may be more difficult to retrofit it into a general purpose OS that in a clearly defined virtualization layers. 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].

For now, many people are looking at virtualization environments and deciding which ones to use for different applications. The issue of isolation from misbehaving VMs is an important one to consider, especially for a commercial hosting environment, and we hope this data will provide some guidance in this area.

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. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), ACM Operating Systems Review, Winter 2002 Special Issue, 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.

[7] S. Soltesz, M. Fiuczynski, L. Peterson, M. McCabe, J. Matthews. Virtual Doppelg(anger: On the Performance, Isolation, and Scalability of Para- and Paene- Virtualized Systems. Submitted to Eurosys 2006.

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

[9] IOzone 3.257, http://www.iozone.org, Accessed January 2006.

[10] K.Fraser, S.Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe Hardware Access with the Xen Virtual Machine Monitor, 2004.

[11] Solaris Forums. Zoneadm- Why not action=deny in rctl?.URL http://forum.sun.com/thread.jspa?threadID=21712&tstart=0, Accessed January 2006.

[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 2006.