Quantifying the Performance Isolation Properties of Virtualization Systems

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

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

ABSTRACT

In this paper, we present the design of a performance isolation benchmark that quantifies the degree to which a virtualization system limits the impact of a misbehaving virtual machine on other well-behaving virtual machines running on the same physical machine. 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 systems –an example of full virtualization (VMware Workstation), an example of paravirtualization (Xen) and two examples of operating system level virtualization (Solaris Containers and OpenVZ). 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 1.7% degradation in the disk intensive test. The results for operating system level virtualization systems are varied – illustrating the complexity of achieving isolation of all resources in a tightly coupled system. 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.

Categories and Subject Descriptors
C.4 [Performance of Systems]
General Terms
Measurement, Performance, Experimentation
Keywords
Virtualization, Performance Isolation

ExpCS'07, 13–14 June, 2007, San Diego, CA.

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 z/OS [2]. Increasingly, virtualization environments for x86 platforms are targeting commercial server environments as well. [3, 4,15]

In recent years, there have been a number of papers comparing the performance of different virtualization environments for x86 such as Xen, VMware Workstation and UML [5] [6] [7]. 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. This might be measured 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.

Scalability 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.

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? Certainly, running in parallel with other web server VMs is quite different than running in parallel with a fork bomb or other resource hogs. Protecting well-behaved VMs from misbehaving

VMs is an important feature of virtualization systems, especially when used in a commercial hosting environment. The degree of performance isolation can also vary substantially with the type of "misbehavior". A virtualization system may isolate the impact of a CPU hog but not isolate the impact of a network hog.

There are several major types of virtualization systems including full virtualization, paravirtualization and operating system level virtualization. In full virtualization, the interface provided by the virtualization system is the same as the actual physical hardware. This allows unmodified operating system binaries to run as guests in a virtual machine. In paravirtualization, targeted changes are made in the hardware interface presented to a virtual machine to avoid some features that are difficult or expensive to virtualize. This requires that modifications be made in the operating system to deal with this modified hardware interface. In operating system level virtualization, guest virtual machines are actually processes running within a general-purpose operating system that has been modified to provide separate name spaces such that guests appear to be separate machines.

In operating system level virtualization, all guests share the same operating system as the base machine. Thus by definition, operating system level virtualization systems do not support the ability to run 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.

In this paper, we present the design of a performance isolation benchmark and use it to examine three virtualization environments – an example of full virtualization (VMware Workstation), an example of paravirtualization (Xen) and two examples of operating system level virtualization (Solaris Containers and OpenVZ). In Section 2, we describe our performance isolation benchmark suite and in Section 3, we describe the results we obtain using our benchmark suite to test the performance isolation characteristics of Xen, VMware Workstation, OpenVZ and Solaris Containers.

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 well-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 SPECweb 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 at http://www.clarkson.edu/class/cs644/isolation/. 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 without freeing it.
Fork Bomb	Loops creating new child processes.
CPU Intensive	Tight loop containing integer arithmetic 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 senders

Table 1: 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 Workstation, OpenVZ and Solaris Containers.

3.1. Baseline Data

Before beginning our stress testing, we established some baseline data. Specifically, we ran SPECweb 2005 with 4 Apache web servers each in their own 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 five different virtualization environments - Xen 3.0 stable, VMware Workstation 5.5, OpenVZ 2.6.18, an early release of OpenSolaris without additional resource controls configured in each container and a more recent build (build 62) with additional resource controls in each container. With Xen and VMware Workstation, we used the same Linux server image with Linux (2.6.12 kernel). With OpenVZ, the Linux kernel version is 2.6.18, which all the guests share. In Xen, VMware Workstation, we assigned each virtual machine 128 MB of memory. In OpenVZ, each guest was configured with the **vzsplit** tool that attempts to divide the machine's resources evenly among a given number of guests and each of the four guests received roughly a quarter of the 1 GB memory or 256 MB.

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 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 attributed 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 machines 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. In both Xen and OpenVZ cases, the misbehaving VM did not report results, but all others continued to report nearly 100% good results as before. In the VMware Workstation case, the misbehaving VM survived to report significantly degraded performance (8.7% average good responses) and the other three servers continued to report 100% good response time, as in the baseline.

We ran two configurations of Solaris Containers. First, an installation in which no resource control options were added to the container configurations. Second, an installation in which the following limits were added to the container configurations:

```
add capped-memory
    set physical=128M
    set swap=512M
    set locked=64M
end
```

This sets the container's physical memory to 128 MB, its maximum swap space to 512MB and the total amount of locked memory to 64 MB.

Without the resource limits in place, none of the Solaris Containers survived to report results. The test effectively shut down all virtual machines – misbehaving and well behaved.

With the resource limits in place, we experienced two different situations that together illustrate the nature of the resource limits nicely. In both situations, the well-behaved VMs reports a trivial degradation of 0.06%, but the results for the misbehaving VM depends on a slight difference in timing whether we start the memory bomb first or the SPECweb test first. (Note: In general, we tried to start them at approximately the same time.) If we start the memory bomb first, no results will be produced because the web server's memory requirements will be denied. If we start the SPECweb test first, it will report little degradation (0.03%) similar to the well-behaved VMs. In this case, the web server's memory requests are satisfied, but the memory bomb will report insufficient memory errors. We examined the memory bomb program in this second case and found that its virtual memory consumption was capped at 382 MB.

3.2.2. Fork Bomb

We used a classic fork bomb test that loops creating new child processes. Under both Xen and VMware Workstation, the misbehaving virtual machine presented no results, but the other three well-behaved containers continued to report 100% (or near 100%) good response time. Under OpenVZ, the well-behaved guests were also protected and even the misbehaving guest survived to report results, but only 12.2% good response times.

For Solaris, we once again tested in two configurations. Without resource controls, results were not reported for any of the four containers. In the second case, we added the following limits to the container[1] configurations:

```
set max-lwps=175
set scheduling-class=FSS
add capped-cpu
   set ncpus=0.25
end
set cpu-shares=10
```

This sets the scheduler to the fair share scheduler (FSS) and the number of shares for this container to 10. It sets the maximum number of lightweight processes (LWPs) in simultaneous use to 175 and gives this container 25% of the CPU.

As is suggested by this list, one disadvantage of resource controls is the complexity of configuring them properly. For example, we found that if the number of threads (max-lwps) is not set high enough the container would fail to boot. Sun is actively working on making these resource controls easier to use and more integrated with containers [17] [18] [19][20]. More information on available resource control options is available online [21].

With the resource limits in place, the misbehaving VM was still completely unresponsive, but the well-behaved VMs experienced only 0.04% degradation on average.

3.2.3. CPU Intensive Test

Our third test stresses CPU usage with a tight loop containing integer arithmetic operations. All four of our

[1] The term zone is sometimes used instead of container.

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.

For Solaris, we ran without resource controls and with the same resource controls described in previous sections. With resource controls, there is no degradation for any of the VMs. Interestingly, there was slight degradation in the case with resource controls of 0.06% for the well-behaved VMs and 0.07% for the misbehaving VM.

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.

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

For Open VZ, the well-behaved and misbehaving VMs saw similar degradation of 2.52% and 2.63% respectively. Although the degradation is relatively minor, there is no evidence of isolation on this test. The situation is similar for Solaris. With and without resource controls, all VMs experience a slight degradation of 1.13 – 1.59%. We are unaware of any configuration options for disk resources in Solaris.

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.

Under VMware Workstation, 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 also show no degradation and the misbehaving VMs shows a slight but repeatable degradation of less than 1%.

For OpenVZ, all VMs experience significant degradation. The misbehaving VM experiences almost 29% degradation, while the well-behaved VMs fare almost as poorly with an average degradation of 21.3%. Once again, this is evidence of weak isolation.

On Solaris with no resource controls, a degradation of 3.53% to 4.2% is reported for all VMs. We then used the resource controls described in other sections, but no network specific controls. Here the overall degradation is quite low (1.0% for the well-behaved VMs and 0.93% for the misbehaving VM), but no evidence of isolation.

3.2.5.2. Server Receives Data

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

The results for this test are the most varied. For OpenVZ, none of the 4 VMs survived to report any results. While on VMware Workstation, the opposite occurred - all four VMs retained 100% good response. We did not even see degradation on the misbehaving VM as we did in the sender transmit case. One hypothesis is that for VMware workstation, in the face of network contention, the incoming packets are simply dropped before they impact any of the four web servers. We did not collect additional data to prove or disprove this hypothesis. In the OpenVZ case, however, it is clear that the incoming packets are indeed creating interference.

For Xen, the misbehaving VM and the well-behaving VMs are similarly affected with a very slight degradation of 0.03-0.04%.

Solaris presents the most surprising results. With no resource controls, a degradation of 1.24% to 1.67%

	VMware Workstation		Xen		OpenVZ	
	Good	***Bad***	***Good***	***Bad***	***Good***	***Bad***
Memory	0	91.30	0.03	DNR	0	DNR
Fork	0	DNR	0.04	DNR	0.01	87.80
CPU	0	0	0.01	0	0	0
Disk Intensive	0	39.80	1.67	11.53	2.52	2.63
Network Server Transmits	0	52.9	0	0.33	21.3	28.97
Network Server Receives	0	0	0.04	0.03	DNR	DNR

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.

degradation is reported for all VMs. We then used a later build of Solaris with the resource controls described in other sections, but no network specific controls. In this case, the results for the misbehaving and well-behaved VMs are once again similar to each other. However, this time a very high degradation of about 92% is reported for all VMs. In discussions with engineers at Sun, we were unable to find the root of cause this difference in time for the publication of this paper.

In addition, we tried using some network controls using a tool called **ipqosconf.** However, after adding the desired rules, the system became unstable (a continual loop of crashing and automatically rebooting). One hypothesis is that the recent work on IP instances may have caused a bug in the ipqos module. We were using a pre-release version of Open Solaris (build 62) to get the most up-to-date resource controls and this problem may be fixed in later releases.

3.3 Summary of Results

We collect our results in Tables 2 and 3. Table 2 reports the results for VMware Workstation, Xen and OpenVZ. Table 3 reports results for the two Solaris configurations. 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.

Both tables illustrate 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 of Table 2, it is clear that VMware Workstation completely protects the well-behaved VMs under all stress tests. Its performance is sometimes substantially lower for the misbehaving VM. This is likely due to the architectural differences between Xen and VMware Workstation. The virtual machine monitor in the VMware Workstation runs hosted on top of a general-purpose operating system, while the virtual machine monitor in Xen runs directly on the hardware with no intervening software layer. We would expect degradation for the misbehaving VM to be lower using a non-hosted version of VMware such as VMware ESX Server.

	Solaris (Without resource controls)		Solaris (With resource controls)	
	Good	***Bad***	***Good***	***Bad***
Memory	DNR	DNR	0.06	0.03 / DNR
Fork	DNR	DNR	0.04	DNR
CPU	0	0	0.06	0.07
Disk Intensive	1.48	1.23	1.59	1.13
Network Server Transmits	4.20	3.53	1.00	0.93
Network Server Receives	1.24	1.67	92.73	92.43

Table 3: 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.

Xen also protects the well-behaved VMs very well. The average degradation for the disk intensive case is the worst at 1.67%. One thing that this table highlights, however, is a slight but consistent degradation on most tests.

OpenVZ has a mixed record. It clearly demonstrates isolation on the memory intensive test and the fork bomb test. In the CPU test, no VM showed any degradation. In the other tests, however, the well-behaving VMs suffered the same degradation in performance as the misbehaving VM.

Solaris also has somewhat of a mixed record. Without resource controls, there is no evidence of isolation on any test, although overall degradation is quite low in many cases (<= 5% degradation on 4 of the 6 tests). With resource controls, there is clear isolation for the fork bomb test. Similarly, for the memory intensive tests, isolation of the well-behaved VMs is achieved. However for the disk intensive and network intensive tests, the results continue to demonstrate poor isolation and in the case of the network server receive test, the degradation is substantial. Resource controls also continue to be somewhat confusing and difficult to configure properly. Work is on-going to develop new resource controls and to make them easier to use. . However, our experiences suggest that it takes longer to retrofit isolation of all resources -- disk, network (incoming and outgoing), memory, CPU, etc. -- into a general purpose OS than to add it in clearly defined virtualization layers.

Operating system level virtualization has advantages that appear in other situations, like ease of creating a new VM and the ability to create more VMs than would be possible with Xen or VMware Workstation (e.g. 8192 VMs 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 VMs are under the same administrative control, this may be a reasonable trade-off.

4. Future Work

We would like to evaluate other virtualization systems with our benchmark suite especially VMware ESX Server. Similarly, we would like to continue to track resource controls that are added to Solaris and OpenVZ. For example, evaluating network and disk resource controls as they become more widely available. We would like to evaluate some other isolation methods that are more appropriate to larger systems

such as assigning dedicated processors or network interfaces to each guest VM. We will continue to post new results to http://www.clarkson.edu/class/cs644/isolation as we have them and welcome suggestions for additional experiments using this infrastructure.

5. 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 Workstation, Xen, OpenVZ and Solaris Containers.

Our results highlight differences between major classes of virtualization systems – full virtualization like VMware Workstation, paravirtualization like Xen and operating system level virtualization like Solaris Containers and OpenVZ. 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 1.7% degradation for the disk intensive test with many other tests showing only slight, but repeatable degradation.

With operating system level virtualization the need for resource controls, either as a default or through proper configuration, was clear. Without them, well-behaved and misbehaving workloads both suffered. Strong resource isolation clearly can be added to operating system level virtualization. As in the case of Solaris and OpenVZ, the operating system can be modified to implement new resource scheduling algorithms that enforce resource isolation across VMs. When resource controls were available and used properly, only slight degradation, not significant enough to cause noticeable changes in response time or usability, was observed.

Our benchmark suite is available at http://www.clarkson.edu/class/cs644/isolation/.

6. Acknowledgements

We would like to thank Jeff Victor at Sun who went out of his way to help us identify appropriate resource control settings to use when configuring Solaris Containers. Thanks also to Sean Hogan, Jerry Jelinek, Tariq Magdon-Ismail, and Todd Clayton from Sun for their suggestions and comments. We would also like to thank Hermant Gaidhani from VMware for helping to obtain permission to publish the VMware results.

7. 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] IBM's zOS Operating System, http://www-03.ibm.com/servers/eserver/zseries/zos/, Accessed April 2007.

[3] 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.

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

[5] 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.

[6] 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

[7] 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. Resource Control Concepts, URL http://docs.sun.com/app/docs/doc/817-1592/6mhahuoiq?l=en&a=view, Accessed May 2007.

[15] VMware Workstation, http://www.vmware.com/products/ws/, Accessed May 2007.

[16] OpenVZ – Server Virtualization Open Source Project, http://openvz.org, Accessed May 2007.

[17] G. Jelinek, “Containers in SX build 56”, http://blogs.sun.com/jerrysblog/entry/containers_in_sx_build_56, Accessed May 2007.

[18] G. Jelinek, “Improved Zones/RM Integration”, http://www.opensolaris.org/jive/thread.jspa?threadID=10451&tstart=0, Accessed May 2007.

[19] G. Jelinek, “Improved Zones/RM Integration”, http://www.opensolaris.org/os/community/arc/caselog/2006/496/, August 21 1996.

[20] G. Jelinek, “Swap Resource Control; Locked Memory RM”, http://www.opensolaris.org/os/community/arc/caselog/2006/598/, Accessed May 2007.

[21] Sun Microsystems. Configuring Resource Controls and Attributes, URL http://docs.sun.com/app/docs/doc/819-2450/6n4o5md6p?a=view#rmctrls-tbl-5, Accessed May 2007.