Virtual Machine Monitors

Rune Johan Andresen

Contents

1 Introduction

This document describes different virtual machine monitors, their advantages and disadvantages, in order to find the one that most probably satisfies our requirements. Benchmarking results for this virtual machine monitor will tell if virtualization for a batch-oriented grid cluster architecture is useful or not.

1.1 What is virtualization?

Virtualization, in a rather loose definition, is a framework of dividing the resources of a computer into multiple execution environments. More specific it is a layer of software that provides the illusion of a real machine to multiple instances of virtual machines. There are different virtual machine abstractions. The most common examples are hardware-level virtualization, operating-level virtualization and high-level language virtual machines. The hardware-level virtualization layer sits right on top of the hardware exporting the virtual machine abstraction. Traditionally software written for it will run in the virtual machine. Later in this section we will see exceptions where a guest operating system has to be modified to run on the virtual machine monitor in order to maintain performance. The operating system-level virtualization layer is between the operating system and the application programs, which are executed by the virtual machine monitor that virtualize a particular operating system. A High-level language virtual machine layer is an application running on the top of an operating system. The layer exports an abstraction of the virtual machine that can run programs written and compiled to the particular abstract machine definition like the Java Virtual Machine.[4]

There are several hardware-level virtualization architecture variants. Some hardware-level virtual machine monitors (VMM) can run directly on real hardware, independent of any host operating system. Other hardware-level VMM run as an application on top of a host operating system and uses the host‘s API to do everything. If there are differences between the host operating system and the hardware-level VMM, instruction set emulation is often involved. Most operation systems and hardware today is not designed for virtualization, which makes it difficult to make a fully complete machine simulation (fully virtualization). Fully virtualization has the benefit of allowing unmodified operating system to be hosted, providing the illusion of a complete system of real hardware devices. Of course, operating-level virtualization and high-level language virtual machines can be run on a hardware-level VMM, like Java on a VMware guest operating system. In the rest of this report hardware-level VMM is referred as VMM.

1.2 Virtualization for the IA32 platform

This project focus on the IA32 architecture. There are several instructions on an IA32 CPU that makes it unsuitable for virtualization. The IA32 Translation Lookaside Buffer

(TLB) is hardware managed, which also makes it more difficult to virtualize in contrast to other architectures such as PPC or PA-RISC where the TLB is software managed. In order to achieve fully virtualization on the IA32 platform we would need the VMM to be hosted. This means that the VMM is an application, which run as a process on an OS. The obvious disadvantage is the overhead and the performance loss.

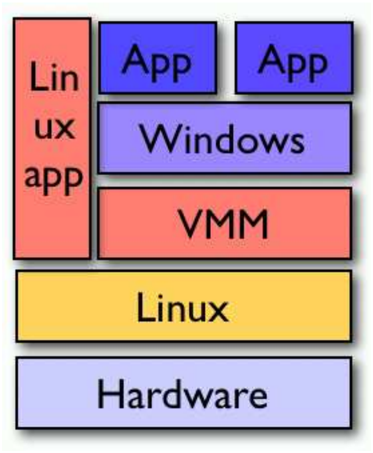


Fig1: Hosted VMM

In order to avoid the drawbacks of full virtualization for the IA32 platform, some virtual machine monitors presents a virtual machine abstraction that is similar (but different to) the underlying hardware. This is called *paravirtualization*. The advantage is improved performance, although it requires modifications to the guest operation system. For the x86 architecture it is desirable for the hosted operating system to see real as well as virtual resources to support time-sensitive tasks and correctly handle TCP timeouts and RTT estimates. This approach can be useful in a unhosted environment with the performance benefits of paravirtualization and the fact that we can skip one layer.

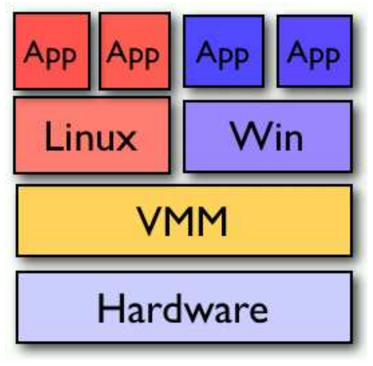


Fig2: Unhosted VMM

Because of the performance requirements for this project the unhosted VMM approach will be the natural choice.

1.3 Why virtualization?

Virtualization has several potential benefits. The question is how much virtualization can exploit the system resources in a better way than traditional native operation system. The obvious advantages are:

- You can share computer resources on a PC without being forced to use the same operating system or applications.
- You can maintain higher security with more isolation between the domains.
- You can merge a virtual operating system to another VMM.
- Easier to recover after an OS crash.

2 Objectives

Modern computers are sufficiently powerful to use virtualization to present effective illusions of many virtual machines. The objective of this project is to perform tests with virtual machine monitors, which allows multiple commodity operating systems to share conventional hardware for the IA32 and preferably the IA64 platforms. The goal is to measure the performance loss of a virtual machine monitor in order to determine if this is a good solution in batch-oriented grid clusters architecture.

By virtualization on the IA32 and IA64 platform we mean paravirtualization. They are not as fully virtualizable as the PPC or IBM mainframes.

2.1 Main aspects

The main aspects of the project are:

- Evaluate different machine monitors in order to find one or more, or eventually exclude the whole idea, solutions, which satisfy a performance requirement for the virtual machines. Different benchmarking will be used to find these results.
- If the performance is sufficient enough in the first place we will evaluate if this extra layer of virtual machines is flexible enough for being used and maintaining a grid effectively.
- Evaluate the machine monitors focus on security. It is necessary to maintain a security level for the virtual machines when different operating systems execute on the same hardware. Virtualization means potential security holes and requires proper testing.

2.2 Requirements

Identified requirements at this point:

- The virtualization testing has to be performed on a Linux distribution on an IA64 or IA32 architecture. Testing on the IA32 platform with the CERN Linux distribution is the first priority.
- The CPU performance loss should most of the time not be more than 6% for a VMM.
- The VMM should be able to run popular and common operating system such as Win XP and Linux.
- Test with realistic benchmarking. It is important to measure for the proper usage for the grid. The true value of a VM is that it lets users utilize their resources more efficiently.
- Test the behavior of a VMM on a single and a dual processor.
- Focus on security. A domain[1] should not be able to interrupt another domain.

3 Considering virtual machine monitors

The virtual machines that will be taken into consideration are:

- VMware - a hosted x86 virtualization monitor which can run a guest operating system unmodified with some performance loss.
- User Mode Linux (UML), a hosted VMM, is essentially a separate port of Linux such that both the guest kernel and user layers both run as a host user process.
- Xen an unhosted x86 virtual machine monitor which requires a modified kernel for the guest OS with minimal performance and functionality loss.
- Virtual machine monitors for the IA64 platform

All the virtual machine monitors have their disadvantages and advantages. We are searching for a monitor, which might be useful for specific software, preferable on the CERN Linux distribution. We will focus on performance and security. A smart grid should exploit all the resources available, and the throughput[2] is THE important factor for this benchmarking.

[1]A domain is the execution context that contains a running virtual machine.
[2]The amount of work that a computer can do in a given time period

3.1 VMware

VMware is a commercial product, which runs as an application on the top of a guest operating system. Advantages: It is stable and is well documented. It allows running various operating systems (Windows, some Linux distributions, OpenBSD and Solaris) unmodified without disk repartition and rebooting. Disadvantages: It does not support Hyper Threading[3] and requires a host operating system, which means an extra layer and additional overhead. The CPU performance loss is between 9 and 15 %. Even though VMware has several benefits, our requirement is less than 6 % CPU performance loss in order to exploit the resources effectively. See Fig.1 under section *Introduction* for an abstract model for the VMware architecture.

3.2 User-Mode-Linux

User-Mode-Linux (UML) is a open source project that enhances the standard Linux kernel in a way that allows it to run as an executable program inside a normal Linux environment. Like VMware it needs to run on a hosted operating system. The CPU performance is not much affected by the UML, but the I/O performance is heavily decreasing. Its main purpose is to test new kernels, experimental development e.g. while performance and security seems to be less prioritized. UML is indeed interesting for Linux development, but is unlikely to satisfy the requirements for this project. UML uses *paravirtualization*. See Fig.1 in section *Introduction* for an abstract model for the UML architecture.

3.3 Xen

Xen is a GPL VMM which allows multiple commodity operating systems to share conventional hardware in a safe and resource managed fashion, without sacrificing performance or functionality. Unlike VMware and UML it can run directly on the hardware. Xen Supported guest operating systems include Linux 2.4 (2.6 in progress), FreeBSD 4.8(in progress), NetBSD 2.0 and in Windows XP(in progres). Xen uses *paravirtualization* for minimal performance loss and requires modified kernels for the guest operating systems. However, user space applications and libaries do not require modification. Accordning to the Xen team‘s own benchmarking[3, 1], it satisfies these requirements:

- The CPU performance loss should not be more than 6% for a VMM. Xen execution performance is close to native.
- The VMM should be able to run popular and common operating system such as Win XP and Linux.

[3]A technique to optimize CPU resource utilization and is included in the latest Intel processors. Its a technology, also known as simultaneous multithreading (SMT), enabling the simultaneous processing of multiple threads of different processes.

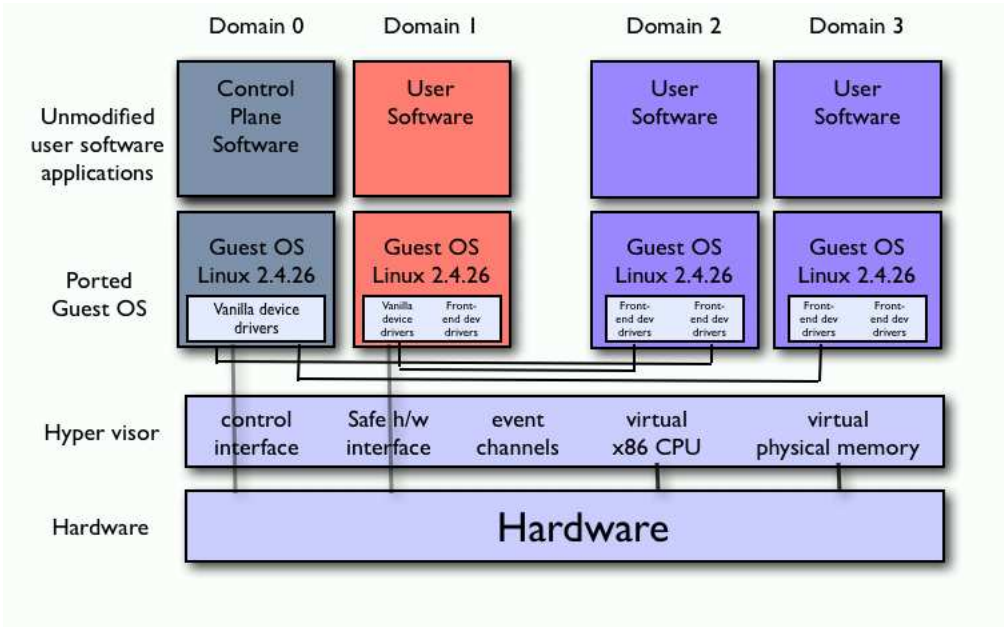
- A domain should not be able to interrupt another domain.

Concerning that domains are not able to interrupt each other, Xen has a "back-end" block driver in domain0[4], which checks to see if a domain is allowed to access a given part of a block device. There are no known way for a domain in Xen to circumvent this unless the domain is privileged, which is NOT the usual case.

The Xen system has multiple layers. The lowest layer is Xen itself, the most privileged piece of code in the system. On the top of Xen run guest operating system kernels, which are scheduled pre-emptively by Xen. On the top level run the applications of the guest operating systems. Guest operating systems are responsible for scheduling their own applications within the time allocated to them by Xen, which also performs suspend, resume and migration[5] of other virtual machines.

3.3.1 Xen I/O devices

The Xen team has developed a next-generation architecture for Xen 2.0 that addresses problems of dependability, maintainability, and manageability of I/O devices and their software drivers on the PC platform. They have reduced the risk of device misbehavior by enforcing isolation between device-granularity protection domains, introduced a set of simple interfaces between OS and driver software and unified the control and configuration of devices in a single OS-agnostic system interface.



[4]The first domain to be started on a Xen machine and is responsible for managing the system
[5]Moving a virtual machine between hosts

Fig.3 Xen 2.0 Architecture

In Xen 2.0 the "real" device drivers run in one or more privileged guest operating systems. Xen only deals with the timer (APIC) hardware, the low-level parts of interrupt dispatch, and some parts of the device probing functionality. Essentially linux device drivers, or BSD drivers, can run unmodified in a guest OS which gives more device support in Xen. Many devices are shared between guest operating systems, but there is only one "real" device driver. To make this sharing work, a privileged guest also includes a "back-end" driver for every real hardware device. All unprivileged guests wishing to share the device include a "front end" driver. Both of these driver are virtual, they don't talk directly to the hardware but are connected together using a device channel. The vanilla Linux device driver is a "native" Linux device driver, while the front-end and back-end drivers are new (fig.3). The lines in the figure connecting domains together represent the device channels that allow domains to talk to one another. The lines in the figure going from a domain trough xen represent the way in which a privileged domain can be given direct access to a subset of the real hardware.

3.3.2 Control interactions

There are two mechanisms for control interactions between Xen and a domain. Synchronous calls from a domain to Xen is made by a hypercall, while notifications are delivered to domains from Xen using an asynchronous event mechanism. The hypercall interface allows domains to perform a synchronous software trap into the hypervisor to perform a privileged operation, analogous to the use of system calls in conventional operating systems. An example use of a hypercall is to request a set of pagetable updates, in which Xen validates and applies a list of updates, returning control to the calling domain when this is completed.

3.3.3 Paging and interrupts

Guest operating systems in Xen can perform its own paging using its own guaranteed memory reservation and disk allocation (direct access to hardware page tables), but updates are batched and validated by the hypervisor. A domain may be allocated discontiguous machine pages. Xen requires every guest operating system to deal with all its own memory faults using its own concrete resources. All paging operations are removed from the xen kernel, exception of updates, while the hypervisor simply is responsible for dispatching fault notifications. This approach is also called *Self Paging*[2]

The asynchronous communication event mechanisms (3.3.2) replaces the usual delivery mechanisms for hardware interrupts and allows lightweight notification of important events such as domain-termination requests. These notifications are made by updating a bitmap of pending event types and by calling an event handler specified by the guest operating system.

3.4 VMMs for the IA64 platform

There are not yet many IA64 VMM products on the market exception of SWsoft release of the IA64 OS Virtualization. Xen 2.0 is currently ported by HP Labs Fort Collins but is unlikely to be ready for testing before next year (2005).

4 Benchmarking

Because of Xen2.0‘s better architecture for safity and performance, this is the VMM we want to benchmark in order to decide if virtualization for a batch-oriented grid clusters architecture is preferable or not. Xen was tested on two different computers, one single CPU and one dual CPU computer for testing how Xen exploits the system resources.

4.1 Benchmarking on a single Pentium4 computer

We wanted to measure the performance loss of the extra layer Xen represents compare to native Linux. A CERN benchmark application was used for testing.

Hardware:

- Pentium 4 2.4 GHz
- 256MB memory

Benchmark Software:

- test40forSPEC (GEANT4)

Operating system:

- Debian Sarge [6]

4.1.1 Benchmarking with one guest operating system

For testing the overhead performance loss, test40forSPEC was first tested on native Linux and then compared to *one* Xen 2.0 guest running the same program.

Results running test40forSPEC on native Linux:

Real	4m16.865s
User	4m16.530s

[6]Please read the *Problems* section

Where *Real* is the wall clock time from the beginning to the end of the program execution and *User* is the time the program did run on the CPU. The system time is close to zero and not important for this benchmarking.

Results running test40forSPEC on ONE *Xen .2.0 guest operating system:*

Real	4m18.900s
User	4m18.860s

Performance loss:

Real	0,787%
User	0,773%

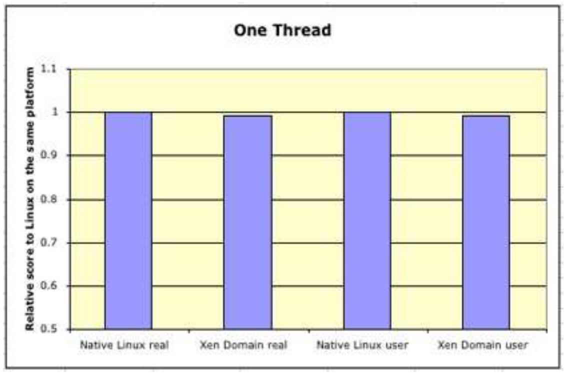


Fig.4 The performance is close to native Linux.

4.1.2 Benchmarking with two guest operating system

Results running test40forSPEC on two guest operating systems simultaneously:

	Guest OS 1	Guest OS2
Real	8m51.721s	8m51.954s
User	8m51.681s	8m51.890s

PS The user and real time is almost identical because the guest operating systems don't know they are sharing the CPU. They can only "see" what is given by the hypervisor.

The time running test40forSPEC on two domains simultaneously is about 2.6% slower than the double of one domain (4.1.1). The overhead performance loss giving resources to both domains is minimal.

4.2 Benchmarking on a dual Xeon computer

Benchmarking of Xen2.0‘s ability to exploit a dual CPU computer.

Hardware:

- 2 Xeon Pentium 4 2.4 GHz
- 1024MB memory

Benchmark Software:

- test40forSPEC

Operating system:

- Scientific Linux CERN 3

4.2.1 Benchmarking with one guest operating system

Results testing Xen2.0 on native Linux, one thread:

Real	4m49.870s
User	4m49.550s

Results running test40forSPEC on ONE *Xen .2.0 guest operating system:*

Real	4m48.001s
User	4m47.930s

Performance loss:

Real	0.645%
User	0.560%

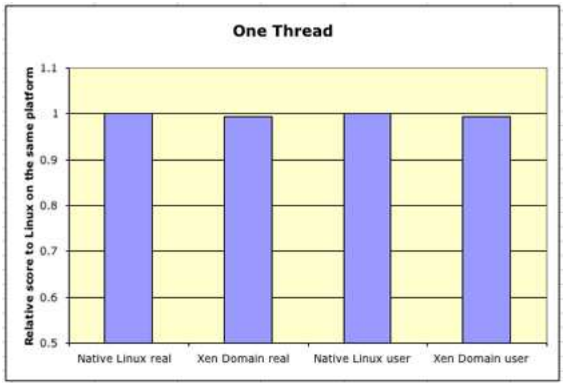


Fig.5 Overhead performance loss

The overhead performance loss on the Xeon dual CPU computer with Scientific CERN 3 is pretty close to the results in 4.1.1 We can assume that the performance loss is a “fixed” factor and changes minimal because of the underlying hardware and “host” operating system.

4.2.2 Testing Xen2.0 with four threads on two domains

We wanted to figure out if executing four threads simultaneously, two on each domain, increase the performance loss. (Fig 4)

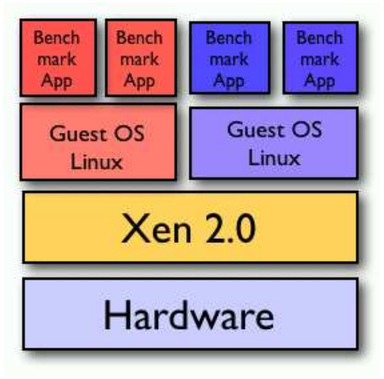


Fig 4. Four threads, two running on each domain

Results running four thread, two on each domain(fig.4):

Domain1	Thread1	Thread2	Domain2	Thread1	Thread2
Real	9m34.915s	9m35.373s	Real	9m42.334	9m43.317s
User	4m48.990s	4m50.060s	User	4m53.550s	4m54.54s

Result four threads on native Linux:

	Thread1	Thread2	Thread3	Thread4
Real	9m40.700s	9m41.536s	9m40.627s	9m41.100s
User	4m50.660s	4m50.480s	4m50.360s	9m51.510s

These numbers show almost identical performance results for Xen2.0 and native Linux.

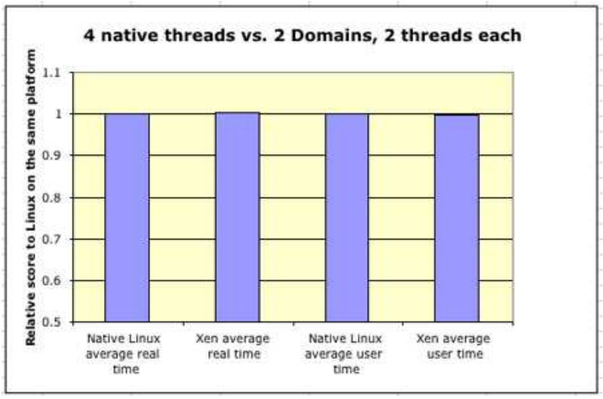


Fig.5 Four threads on native Linux vs. 2 threads x 2 domains on Xen

4.2.3 Testing running five threads, one on each domain

Results running five threads on native Linux:

	Thread1	Thread2	Thread3	Thread4	Thread5
Real	10m29.481s	11m27.273s	12m2.370s	12m7.205s	11m11.145s
User	4m50.690s	4m51.400s	4m49.510s	4m49.140s	4m50.110s

Results running five thread, one on each domain:

	Domain1	Domain2	Domain3	Domain4	Domain5
Real	9m53.510s	9m52.395s	15m16.424s	15m17.536s	15m18.627s
User	9m53.390s	9m52.270s	15m16.320s	15m17.420s	15m18.540s

PS The user and real time is almost identical because the guest operating systems don't know they are sharing the CPU. They can only "see" what is given by the hypervisor.

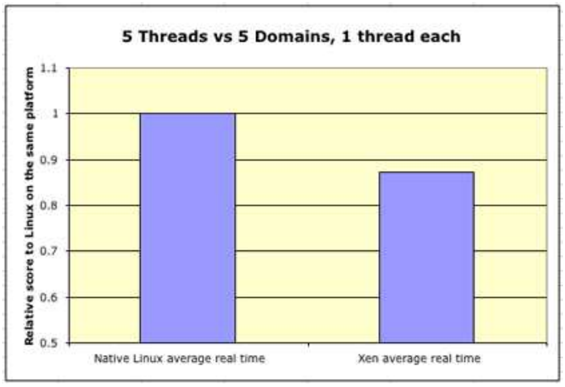


Fig.6 5 Linux native threads vs. 5 domains. one thread each

Native Linux is able to share the system resources more “fair”, while Xen demonstrates that one domain only can have one CPU dedicated to it. Here CPU1 is dedicated to two domains while CPU2 is dedicated to three domains. The average real time used by native Linux is 11m27.495s while it is 13m7.699 for Xen2.0. In this special case Xen 2.0 has a performance loss of 12.72%. After that said, the threads in this test are not isolated from each other. They run on the same user.

5 Problems

Some problems encountered while I installed and benchmarked Xen 2.0.

5.1 Xen-unstable(2.0) install

Because there are several differences between Xen 2.0/Unstable and Xen 1.2, it was necessary to benchmark and install Xen-Unstable, which in these days will go Xen 2.0. In the beginning of this project the unstable was pretty buggy. However, Xen-unstable never booted on Scientific Linux CERN 3 on the first computer (P4 2.4 GHz 256Mb ram). It hanged on “Freeing unused kernel memory”. This often happens when the kernel is compiled for the wrong CPU, even though this was not the case. It seemed that Xen had problems to mount the root file system, but I never figured out why. For some reasons it did boot on Debian Sarge on the same computer. If there was an upgrade in the Xen-unsable source tree that made it work on Debian and the dual CPU computer, or it was because of some local conflicts on the first computer is unknown. I also tried to compile the xen kernel with sysreq enabled in order to debug the boot up process, without positive results.

5.2 Benchmarking Xen 2.0

I got some strange results during the benchmarking because I dedicated random vmids[7] to the domains. The mod of the vmid decides which CPU the domain gets. In order avoid this it is a good idea to let Xen do the CPU management. This is not a default option in Xen-Unstable(2.0), it has to be changed manually in the configuration file for the respective domains.

6 Conclusion

Xen was without doubt the right virtual machine monitor for benchmarking, with its performance results and security advantages. The first results show that the overhead

[7]The domain ID

performance loss for pure usage of the CPU is less than 1 %. Testing the IO overhead is yet to be done.

Installing Xen-unstable(2.) was a challenge, and a lot of minor problems continuously occurred. There are still many features to add in order to make Xen a user friendly product and easy to use for system administrators e.g. However, after installed, it provides an excellent platform for deploying a wide variety of network-centric services. In fact, it allows up to 100 operating systems to run on a single server.

Xen2.0 exploited the dual CPU computer very well, specially when the numbers of domains are even. Somewhat more performance loss occurred when we tried to run a uneven number of domains on the dual CPU computer. With the knowledge that Xen performs at almost native Linux speed, the virtualization for a batch-oriented grid cluster architecture should be very useful where the advantages of virtualization are needed. *Paravirtualization* is indeed a good solution for the IA32 platform where close to native Linux performance is important.

After performing these tests we can say that Xen2.0's performance is very good. Xen lives up to its claim of high performance virtualization of the IA32 platform.[3, 1]

References

[1] Eli Dow Stephen Evanchik Matthew Finlayson Jason Herne Jeanna Neefe Matthews Bryan Clark, Todd Deshane. *Xen and the Art of Repeated Research*. Clarkson University, 2004.

[2] Steven M.Hand. *Self-Paging in the Nemesis Operating System*. University of Cambridge Computer Laboratory, 2004.

[3] Keir Fraser Steven Hand Tim Harris Alex Ho Rolf Neugebauer Ian Pratt Andrew Warfield Paul Barham, Boris Dragovic. *Xen and the Art of Virtualization*. University of Cambridge, 2004.

[4] Mendel Rosenblum. *The Reincarnation of Virtual Machines*. qFOCUS, 2004.