PERFORMANCE BENCHMARKING
PHYSICAL AND VIRTUAL LINUX ENVIRONMENTS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF SCIENCE
UNIVERSITY OF CAPE TOWN
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE (MIT)

BY
MARIO FISHER (FSHMAR003)
2012
SUPERVISED BY
PROFESSOR KEN MacGREGOR

Acknowledgments

I would like to take this opportunity to thank the following people, without whom this dissertation would not have been possible:

- My wife, Jill Combrinck, for your tireless motivation, inspiration and support.
- My parents, Derek and Devine Fisher for all your encouragement.
- Professor Ken MacGregor for your supervision and wisdom.
- The developers and contributors of Debian GNU/Linux, Xen and OpenVZ.
- The developers and contributors of lmbench3, BYTEMark* Native Mode Benchmark and UnixBench.

for Emma

Declaration

I, Mario Fisher (FSHMAR003), hereby declare that the work on which this dissertation is based is my original work (except where acknowledgements indicate otherwise) and that neither the whole work nor any part of it has been, is being or is to be submitted for another degree in this or any other University. I empower the University of Cape Town to reproduce for the purpose of research either the whole or any portion of the contents in any manner whatsoever.

M Fisher

February 2012

Abstract

Virtualisation is a method of partitioning one physical computer into multiple "virtual" computers, giving each the appearance and capabilities of running on its own dedicated hardware. Each virtual system functions as a full-fledged computer and can be independently shutdown and restarted. Xen is a form of paravirtualisation developed by the University of Cambridge Computer Laboratory and is available under both a free and commercial license. Performance results comparing Xen to native Linux as well as to other virtualisation tools such as VMWare and User Mode Linux (UML) were published in the paper "Xen and the Art of Virtualization" at the Symposium on Operating Systems Principles in October 2003 by (Barham et al, 2003). (Clark et al, 2004) performed a similar study and produced similar results.

In this thesis, a similar performance analysis of Xen is undertaken and also extended to include the performance analysis of OpenVZ, an alternative open source virtualisation technology. This study made explicit use of open-source software and commodity hardware.

Contents

List of Acronyms

ADD	Addition
AMD	Advanced Micro Devices
B	Byte
CPU	Central Processing Unit
DDR	Double-Data-Rate
DIV	Division
Dom0	Domain0 (Privileged Xen domain)
DomU	DomainU (Unprivileged Xen domain)
EXP	Exponent
ext3	Third Extended Filesystem
FIFO	First-in First-out
FP	Floating Point
FPU	Floating Point Unit
FreeBSD	Free Berkeley Software Distribution
FSB	Front-side Bus
GB	Gigabyte
GNU	GNU's Not Unix!
I/O	Input/Output
IP	Internet Protocol
IPC	Inter-process Communication
IDE	Integrated Drive Electronics
KB	Kilobyte
L2	Level Two
MBit	Megabit
MFLOPS	Million Floating Point Operations per Second
MIPS	Million Instructions per Second
MOD	Modulus
MUL	Multiplication
OS	Operating System
OSDB	Open Source Database Benchmark
RAM	Random-Access Memory

RPC	Remote Procedure Call
SPEC	Standard Performance Evaluation Corporation
SQRT	Square root
SUB	Subtraction
TCP	Transmission Control Protocol
TLB	Translation Lookaside Buffer
UDP	User Datagram Protocol
UML	User-Mode Linux
VE	Virtual Environment
VM	Virtual Machine
VMM	Virtual Machine Monitor
VT	Virtualization Technology (Intel)

List of Figures

List of Tables

Appendices

Chapter 1: Introduction

Virtualisation is the creation of a virtual version of a physical resource or device, such as an operating system, a server, a storage device or network resources, etc. Virtualisation can be described as the abstraction and simulation of physical resources or devices.

Virtualisation software allows physical hardware ("the host") to run multiple operating system images at the same time ("the guests"). Each virtual machine or guest is like a computer within the computer and functions as if it is the only operating system utilising the hardware. (Symantec, 2010) states that the technology had its start on mainframes decades ago, allowing administrators to avoid wasting expensive processing power. This decades-old technology is becoming popular again, especially in the context of energy-saving and datacentre consolidation.

Various forms of virtualisation are available, e.g. virtual machine or hardware emulation in the form of VMWare[1], paravirtualisation in the form of Citrix XenServer (Xen)[2] and operating system virtualisation in the form of OpenVZ[3].

Virtual machine or hardware emulation enables various operating systems to run on the host. The guest operating systems communicate with the physical hardware via the virtual machine monitor (VMM). The VMM emulates or virtualises the physical hardware for each virtual machine.

Paravirtualised guest operating systems are modified to recognise the VMM.
(Barham et al, 2003) describe Xen as a VMM which allows more than one operating system to share commodity hardware in a safe and resource managed approach, without sacrificing performance or functionality. Linux and various UNIX versions have been paravirtualised to run using the Xen environment. The VMM in this case is referred to as a hypervisor and marshals guest access to the physical hardware resources.

[1] http://www.vmware.com
[2] http://www.xen.org
[3] http://wiki.openvz.org

Operating system virtualisation supports only the same operating system in each guest. It replicates components of the host operating system into each guest and has no emulation through a VMM or hypervisor.

These forms of virtualisation are offered as proprietary as well as open source solutions, each with its own pros and cons.
Virtualisation offers many possibilities and benefits, but how these virtual systems perform in practice compared to a physical system still requires some more research.

In the following section the motivation behind the research is discussed.

1.1 Motivation

Common problems in datacentres are the provision of space, heat and power to house server hardware. Growing server farms require more floor space, cooling and electricity. The advent of rack-based servers has taken some steps to address the problem of floor space, but has increased the amount of cooling and power needed.

Physical resource allocation on the average server in a datacentre is hardly optimally deployed. For instance, a web server may utilise quite a bit of memory in order to serve page requests effectively, however, the CPU usage would be relatively low. This underutilised CPU resource could be used more effectively by a CPU-intensive application that does not require lots of memory, eliminating the need for two physical servers. These separate environments would need to be totally isolated and only have access to the resources assigned to them.

The interest in virtualisation is increasing, especially in the context of our power crisis and load-shedding as it provides a means of consolidating server hardware, thus reducing power and cooling needs and facilitating optimal resource usage on the physical machines.

The question which must be considered: Are virtual environments as good as the physical server?

The following section outlines the main objectives of this research.

1.2 Aims and Objectives

The aim of this research was to

Investigate the performance of virtualisation techniques, using benchmarking software and commodity hardware.

The precise objectives of the research were:

1. To select, install and setup open source paravirtualisation and operating system virtualisation on commodity hardware.

2. To quantify the performance of each environment, physical and virtual, by using open source benchmarking software.

In the following section, the methodology used in order to achieve the objectives is described.

1.3 Methodology

(Barham et al, 2003) compared the performance of Xen to VMWare, native Linux and User-mode Linux in terms of the total system throughput. (Clark et al, 2004) reproduced the tests independently and extended the study to compare the performance of Xen on commodity hardware to a high-end server.
This experiment extends the methodology used by both (Barham et al, 2003) and (Clark et al, 2004) by introducing OpenVZ as an alternative virtualisation platform to Xen. This research also makes explicit use of open source benchmarking tools as well as commodity hardware.

Paravirtualised Xen and OS-virtualised OpenVZ "guest" systems were benchmarked against a physical system, using open-source benchmarking software. The physical system also served as the host for each form of virtualised guest.

The study made use of entry-level, commodity hardware and the chosen operating system was Debian GNU/Linux[4]. The latest versions of the Xen hypervisor, OpenVZ and their respective administration tools, available in the Debian repository at the time of the study, were used.

The following open source benchmarks were used:

- lmbench3[5]
- nbench/BYTEmark* Native Mode Benchmark[6]
- UnixBench[7]

[4] http://www.debian.org
[5] http://lmbench.sourceforge.net/
[6] http://www.tux.org/~mayer/linux/bmark.html
[7] http://www.tux.org/pub/tux/niemi/unixbench/

Alternative benchmarking tests were evaluated from the Linux Benchmark Suite[8] and the Linux Benchmark Project[9], including HBench-OS[10] however, compiling this software was not successful.

The following section highlights the assumptions made.

[8] http://lbs.sourceforge.net
[9] http://www.tux.org/bench/html/frame.html
[10] http://www.eecs.harvard.edu/~vino/perf/hbench/

1.4 Assumptions

It is assumed that virtualisation has practical applications, and that, using virtualised systems to consolidate physical systems has benefits such as energy-saving and datacentre consolidation as discussed in 1.1.

The experiment assumed that the use of differing versions of the 2.6 branch of the linux kernel, 2.6.18-4 on the host, 2.6.18-6 with Xen and 2.6.18-12 with OpenVZ, would not materially affect the outcome of the benchmark tests.

It was also assumed that having xend and vzctl, the Xen and OpenVZ administration software, installed but not running on the host, would not adversely affect the host's performance during benchmarking.

The next section outlines the scope and limitations of the project.

1.5 Scope and Limitations

This experiment is focused on benchmarking virtualised systems to a physical system. The scalability of these virtual environments was not tested and the study is limited to a single Xen and OpenVZ virtual environment.

The experiment does not attempt any application benchmarking, such as measuring the performance of web or database servers in each environment. Neither does it attempt to assess which benchmarking software is best.

The next section outlines the organisation of this dissertation.

1.6 Organisation of this Dissertation

This section provides a chapter by chapter breakdown of the dissertation:

Chapter 2: Background and Literature Review

This chapter provides an overview of the technologies used in this investigation and provides a summary of existing research done in similar areas.

Chapter 3: Method and Analysis

This chapter describes the process and tools used to perform the investigation. The benchmarks and tests are discussed.

Chapter 4: Results

This chapter provides an evaluation of the results.

Chapter 5: Conclusion

This chapter summarises the findings of the investigation and discusses to what extent the original aims of the project were met. The implications of problems encountered are discussed and suggestions for future work are made.

Chapter 2: Background

This section gives an overview of datacentres and the various virtualisation technologies used in this study. It also gives a summary of similar research done in this area. It includes a description of some of the types of virtualisation and introduces Xen and OpenVZ in more detail.

Various levels of virtualisation are available and this research will focus on the performance of paravirtualisation, using Xen, and OS-virtualisation, using OpenVZ. The performance of a virtualised system can be compared to a physical system by performing identical actions on each, then quantifying and comparing the results.

This review will present some background on datacentres and the underlying principles of virtualisation through insights into various types of virtualisation. This will be followed by findings in the form of previous work in the area of performance benchmarking and how this is can be applied to virtualisation.

2.1 Data Centres

Servers are usually housed in datacentres, which warrants some background discussion about datacentres. Understanding the application and usefulness of virtual systems will be more beneficial when the environment in which the physical server is housed is better understood.

(Andrzejak et al, 2002) defines datacentres as physical premises, used to accommodate large quantities of physical computing resources and management hardware. A distinction between the computing hardware, including the applications they serve within the datacentre and the physical datacentre premises has to be made as this study focuses on the former.

In addition to physical servers and the applications they serve, datacentres also include storage and networking components. Networking and network topologies are beyond the scope of this discussion, data storage is discussed briefly.

The applications provided by datacentres are reliant on the server hardware to accept input, execute the processing and ultimately deliver the service or output. The role of the server hardware is to serve applications and the role of the application is to perform a function. Each physical server can therefore serve or perform multiple functions.

According to (Tate et al, 2003), organisations have an increasing need to store and manage data. This data is often unique to an organisation and collected over a period of time. Various storage options are available for datacentres, including physical disks, Storage Area Networks, Network Attached Storage and Server Attached Storage, to name a few.

Due to their complexity, datacentres are prone to several problems and these are often used by companies to market their virtualisation technology. Inefficient resource utilisation of physical resources is a common shortcoming of datacentres. (VMWare, 2007) markets a substantial increase in resource utilisation rates when deploying their virtualisation products, inferring that physical servers suffer from severely inefficient

resource utilisation. (Andrzejak et al, 2002) undertook a study of six datacentres housing a multitude of servers and corroborate the claims. This waste of resources translates into excess capacity as the physical hardware is being underutilised.

According to the Hewlett-Packard Systems Architecture Group[11], datacentres should be robust and able to cope with resource outages and unpredictable demand. Following on from (Andrzejak et al, 2002) and (VMWare, 2007), applications served by datacentres are typically served by dedicated physical resources. Applications are therefore served from over-provisioned physical hardware in order to meet the needs of excess demand, in the event of that need arising. The result is poor resource allocation.

According to (Carolan et al, 2004), cost reduction in the datacentre is a high priority as managing the physical systems consumes a huge portion of Information Technology budgets. These costs are incurred to keep services at an acceptable level to meet demand.

Datacentres are expected to meet varying demands and over-provision physical servers, resulting in large numbers of servers that need to be managed and maintained. According to (Sliwa and Vijayan, 2002) the management of multiple physical servers is a costly exercise. These applications served by the physical servers are often interdependent, compounding the problem.

Factors contributing to excessive physical hardware include having n-tier applications served from multiple physical servers, with each server serving one or more tiers. Other examples are those where multiple physical servers are deployed to serve separate development, testing and production environments.

[11] http://www.hpl.hp.com/research/dca/system/

Most literature concerning datacentres focuses on the requirement for cooling, uninterruptable power, fire suppression and general compliance with organisational governance and regulatory requirements. It is harder to find literature on the serving of applications and the physical hardware which enables the application serving. Most of the information was obtained from literature published by Hewlett-Packard and VMWare, showing a strong link between datacentres and vendors.

2.2 Virtualisation

Various types of virtualisation are available today. These are both proprietary and open-source and run on high-end as well as commodity hardware.

(OpenVZ, 2008) define virtualisation as “a framework or methodology of dividing the resources of a computer into multiple execution environments. Virtualisation techniques create multiple isolated partitions — Virtual Machines (VM) or Virtual Environments (VEs) — on a single physical server”.

For the purposes of narrowing this definition, three approaches to virtualisation are defined:

- System Virtual Machines
- Paravirtualisation
- Operating System Virtualisation

2.2.1 System Virtual Machines

System virtual machines can be split into three broad groups based on the method of implementation used (King et al, 2003) (Robin et al, 2000).

These groups are:

- Type I Virtual Machine Monitors
- Type II Virtual Machine Monitors
- Hybrid Virtual Machine Monitors

Type I VMM

According to (King et al, 2003), Type I Virtual Machine Monitors are also known as Native Virtual Machine Monitors. Type I VMMs are characterised by having the virtualisation layer directly between the physical hardware and guest operating system. The Type I VMM operates in a higher privilege mode that any other application on the physical server.

As stated, Type I VMMs do not have a host operating system between the physical hardware and the VMM. Type I VMMs manage the physical hardware resources directly and although this does improve performance, it comes at the cost of complexity (Robin et al, 2000). Type I VMMs suffer from the inability to make use of the I/O services of the host operating system.

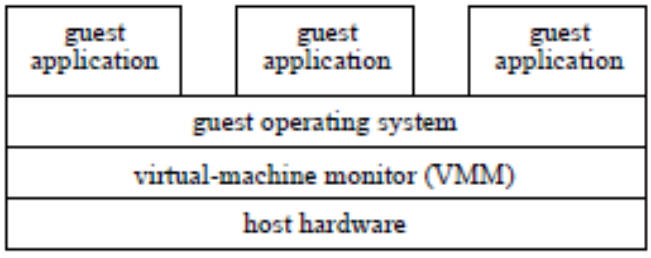

***Figure 1**: An illustration of a Type I VMM (King et al, 2003)*

Type II VMM

Type II VMMs are characterised by having the VMM hosted on an operating system installed on the physical hardware. Although this is a less efficient approach, it is a simpler approach as the VMM can utilise the host operating system services. Each virtualised guest is implemented as a process on the physical host (King et al, 2003). An example of a Type II VMM is User Mode Linux[12].

(Höxer et al, 2002) examined issues which arose when the kernel is ported to the system call interface of the host operating system as implemented under Type II VMMs. They identified problems with the system calls from applications running on a hosted kernel as a common issue. According to (Höxer et al, 2002) these rogue system calls could be handled by an intermediate process which would redirect them to the hosted kernel instead.

According to (Dike, 2000), hardware is emulated and functionality is provided by the physical host. This emulation layer introduces significant overhead according to (Barham et al, 2003). The overhead incurred by context switching between processes is particularly high. According to (Robin et al, 2000), the security of Type II VMMs depends on the security of the operating system installed on the physical host. A weakness in the host operating system could be exploited and compromise the security of the guest.

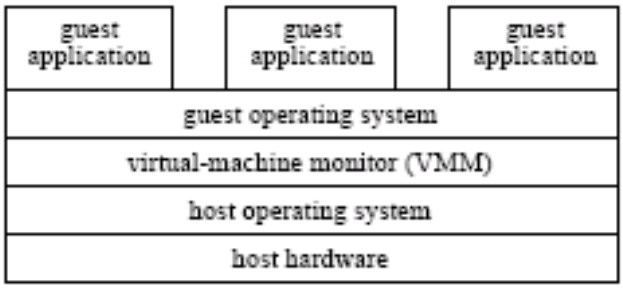

***Figure 2**: An illustration of a Type II VMM (King et al, 2003)*

[12] http://usermodelinux.org/

Hybrid VMM

Hybrid VMMs are hosted partially in privileged mode and partially in non-privileged mode and is also known as Dual-Mode VMMs (King et al, 2003). According to (King et al, 2003), hybrid VMMs access the physical host hardware directly, however, these guests rely on the physical host's operating system for I/O functionality. This does suggest that switching between the VMM and the physical host's operating system during I/O operations would introduce a certain level of overhead. The benefit of hybrid VMMs is that these guests can use the hardware drivers of the physical host's operating system.

According to (Sugarman et al, 2001) hybrid VMMs generally have three components:
The *native* component interacts directly with the physical hardware.
The *user* component uses the physical host's operating system to perform I/O functions and assign resources.
The *driver* component is visible to the physical host as a device driver and provides a communication network between the native and user components.

(OpenVZ, 2008) states that this approach emulates hardware and requires access to physical resources on the host. It is used by most emulators and the guest can be run without modification as it is unaware that it is not utilising real hardware resources. This approach requires the VMM to monitor instructions in real-time and ensure they executed safely. Further examples include VMware, QEMU[13] and Microsoft Virtual Server[14].

[13] www.qemu.org
[14] www.microsoft.com/windowsserversystem/virtualserver

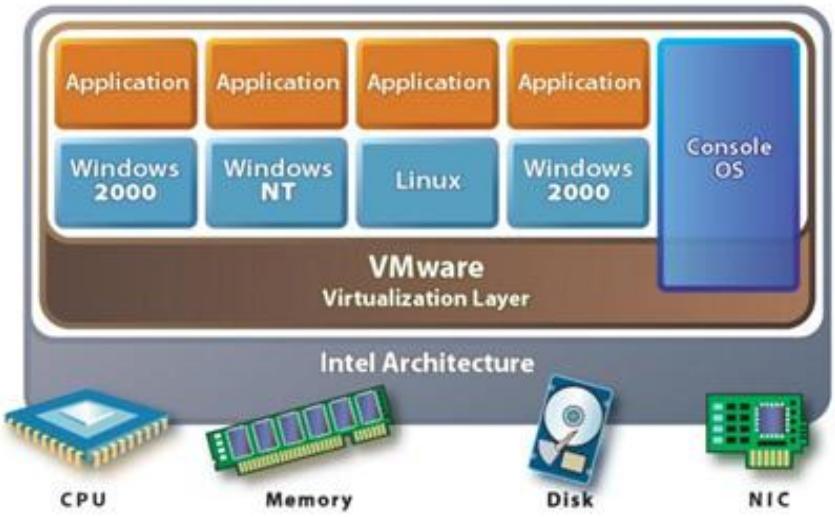

***Figure 3**: An illustration of VMWare ESX Server Architecture (http://www.vmware.com)*

Figure 3 depicts the structure of a virtual machine running VMWare ESX Server, hosting various guest operating systems, including the Console OS (VMM) and emulation/virtualisation layer.

2.2.2 Paravirtualisation

Adapted from (Barham et al, 2003), paravirtualisation involves modifying an architecture to make it more suitable for virtualisation. Under the paravirtualisation approach, guest operating systems are fully aware that they are hosted on a VMM. Handshaking is used as a communication network between the VMM and the guest operating system. This approach improves coordination between the VMM and the guest operating system. An added advantage is that duplication of functionality in the VMM and guest operating system is reduced.

Paravirtualisation involves the virtualisation of physical resources; each resource type is discussed briefly.

Processor Virtualisation

According to (Barham et al, 2003), paravirtualisation systems such as Xen alter the guest operating system to operate in a lower privilege mode. A further idiosyncrasy of the Xen architecture, highlighted by (Barham et al, 2003) is the removal of instructions which cannot be isolated safely.

Guest operating systems modified for paravirtualisation have a special handler for system calls thus bypassing the need for these calls to be sent to the hypervisor (VMM). This suggests a boost in performance as the VMM is relieved of this extra overhead. Leading CPU manufacturers, Intel and AMD have introduced hardware support for processor virtualisation through Intel VT and AMD Pacifica technology designed to simplify VMM implementation and boost performance.

Memory Virtualisation

According to (Barham et al, 2003), keeping shadow page tables up to date incurs a performance overhead and paravirtualisation can improve the performance of memory virtualisation. Xen makes use of guest operating systems modified to only access memory pages allocated to them by the Xen hypervisor. The Xen hypervisor is responsible for confirming updates to the page table. This suggests an improvement in performance as this approach removes the need for shadow page tables.

Handshaking, defined earlier, can also be used to remove the need for shadow pages by specifying non-paged mode as an option when initiating the VMM. An additional handshaking approach, pseudo-page-fault handling, allows the Xen hypervisor to handle page faults while allowing the guest operating system to schedule another process. Intel's VT and AMD's Pacifica, discussed previously, also provide extensions for memory virtualisation at the hardware layer.

Input/Output (I/O) Virtualisation

Further challenges for designers of virtualisation systems highlighted by (Barham et al, 2003) include supporting a wide range of hardware devices. Type II and Hybrid VMMs have the benefit of accessing device drivers on the physical host's operating system, at the cost of extra overhead. Type I VMMs, such as VMWare ESX Server (figure 3) generally need to include device drivers for hardware they need to support. Alternatively, a dedicated I/O virtual machine can be used to access devices on the physical host. According to (Fraser et al, 2004), this functionality was added to Xen and state that using this approach ensures the guest operating system is protected from being compromised by the failure of device drivers.

Resource Scheduling and Guarantees

VMMs have control over the resources of the physical host machine and can implement resource guarantees to the guests. Physical resources such as CPU capacity, memory, and disk quotas can be controlled and allocated by the VMM. Schedulers are used to control the physical resources types mentioned above.

(Padala et al, 2008) and (Sukaridhotoy et al, 2009) state that the Xen hypervisor contains a CPU scheduler that implements scheduling policies, along with other modules such as memory management. A detailed discussion of the extensive range of schedulers and scheduling techniques is beyond the scope of this study, however, Lottery and Stride schedulers are common examples.

(OpenVZ, 2008) states that like system virtual machines, the paravirtualisation approach also requires a VMM, but most of its work is performed on the guest operating system. Paravirtualisation also enables different operating systems to run on a single server; however they need to be modified to run under the hypervisor. According to (Sukaridhotoy et al, 2009), the Xen hypervisor provides a thin software virtualisation layer between the guest OS and the underlying hardware. Each guest is a modified version of the OS, as the hardware presented by the hypervisor is not identical to the physical hardware.

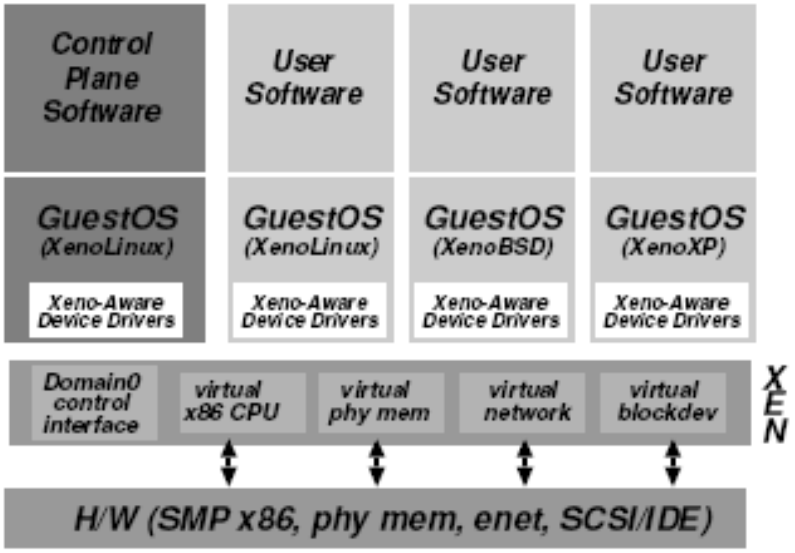

***Figure 4**: An illustration of Paravirtualisation Architecture*
(Barham et al, 2003)

Figure 4 depicts the structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including Domain0 running control software in a Xen Linux environment.

2.2.3 Operating System Virtualisation

An alternative to system virtual machines and paravirtualisation involves creating a virtual environment within a single operating system. “Container” will be used to commonly refer to the virtualised portion of the operating system. Containers provide varying degrees of isolation within a single operating system with each container appearing as a separate server.

Although resource containers do not provide virtualisation functionality, the resource management concepts described by (Banga et al, 1999) are crucial for providing isolation for OS virtualisation. These concepts will be discussed briefly.

(Banga et al, 1999) defines a resource container as an abstract operating system that contains all the system resources used by an application to accomplish a specific action. (Banga et al, 1999) identifies a number of shortcomings of operating systems and states that this approach can be used to overcome them. Resource containers treat operating system processes as independent activities, unlike many server applications which often create multiple processes.

Another common scenario identified by (Banga et al, 1999) occurs when a single process handles all requests. Additionally, kernel processing is often not counted as a resource used by a process, an example would be a process such as networking.
By grouping related activities, resource containers can be utilised to overcome these examples of resource management shortcomings.
When combining resource containers with accurate resource accounting, the allocation and management of physical resources can be improved. Resources can be allocated to a container instead of an individual process.

Container-based virtualisation systems present the user with what seems to be multiple virtual servers. These are however containers which may in fact share a single kernel with many other containers on the same physical host. Securing and isolating each container is as important as providing the impression of a separate OS instance. (Kamp et al, 2000) state that this approach is compatible with practically all applications. Each container has its own root password, IP address and a subset of the original filesystem. This isolation ensures that processes in one container cannot access information about processes in another container. Similar to the paravirtualisation approach, this functionality is implemented by modifications to the underlying OS on the physical host. (Kamp et al, 2000) discuss these modifications with reference to the security of FreeBSD jails and state that this approach has very little overhead. Each container can have resources assigned to them and be rebooted independently.

Fewer resources are consumed to enable container-based functionality compared to system virtual machines which incurs extra overhead through by using multiple operating system instances. Virtualisation is implemented at the system call level, at the cost of sharing a single kernel across containers. A disadvantage of this approach would be when one container causes the kernel to crash, affecting all other running containers. As mentioned, containers require modifications to the operating system and share a single kernel. The result is a comparatively lower level of isolation than system virtual machines and paravirtualisation.

(OpenVZ, 2008) states that most applications running on a server can easily share a physical host machine with others, provided these guests could be isolated and secured. According to (OpenVZ, 2008), different operating systems are not required on the same server, simply multiple instances of a single operating system. OS virtualisation has been designed to provide the required isolation and security to run multiple instances of the same operating system (but different distributions) on the same physical host. Examples include OpenVZ, Solaris Zones[15] and FreeBSD Jails[16].

[15] http://www.sun.com/software/solaris/ds/containers.jsp
[16] http://www.freebsd.org/doc/handbook/jails.html

The main distinction between OpenVZ and Xen is that the former approach uses a single kernel shared by all the guests whereas the latter approach does not (Sukaridhotoy et al, 2009). Therefore, according to (Sukaridhotoy et al, 2009), OpenVZ cannot provide the same fault isolation level as in Xen.

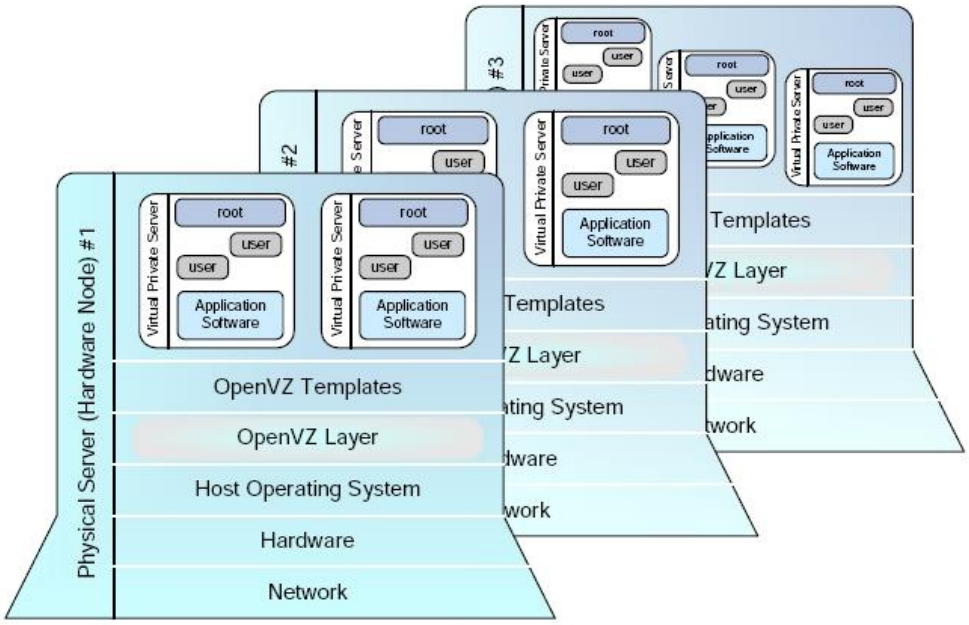

Figure 5*: An illustration of Operating System Virtualisation Architecture (http://www.linuxdevices.com/files/misc/openvz-architecture.jpg)*

Figure 5 depicts the structure of a machine running OpenVZ, hosting a number of identical guest operating systems, including the shared virtualisation layer in a Linux environment.

These three approaches present a subset of various virtualisation techniques and the rest of this review will focus purely on paravirtualisation as provided by Xen and OS virtualisation as provided by OpenVZ.

2.3 Findings

In order for virtualisation to be adopted in industry it is necessary to measure its performance. Comparing systems by benchmarking one against another is frequently used to gauge performance.

(Padala, et al, 2008) state there is rich literature on the Xen virtualization system. (Barham et al, 2003) undertook a performance evaluation of Xen, using SPEC CPU2000[17], OSDB[18], dbench[19] and SPECWeb[20] benchmarks and the results were publicised in the first Symposium on Operating Systems Principles paper on Xen.

According to the online documentation, SPEC CPU2000 is a software benchmark product developed by the Standard Performance Evaluation Corporation (SPEC). It consists of a benchmark for measuring and comparing CPU integer performance and a benchmark for measuring and comparing CPU floating point performance.

As described on the website, OSDB (Open Source Database Benchmark) evolved out of a small project at Compaq Computer Corporation, originally designed to test the I/O throughput and processing power of GNU Linux/Alpha. The result was a database benchmarking suite.

According to the description online, dbench is a tool to simulate I/O workloads to a local filesystem or to a networked file server.

According to the online documentation, SPECWeb is a benchmark for evaluating the performance of web servers.

(Barham et al, 2003) compared the performance of Xen to VMWare, native Linux and User-mode Linux in terms of the total system throughput. (Clark et al, 2004) reproduced the tests independently.

[17] www.spec.org/cpu2000/
[18] http://osdb.sourceforge.net/
[19] http://dbench.samba.org/
[20] http://www.spec.org/web99/

(Barham et al, 2003) list various studies comparing Xen to other virtualisation approaches as well as physical hardware. Their study was targeted at hosting up to 100 virtual machine instances simultaneously on a modern server. They allow operating systems such as Linux and Windows XP to be hosted simultaneously for a negligible performance overhead, at most a few percent compared with the physical host. Their results show Xen considerably outperforms competing commercial and freely available virtualisation solutions, VMWare and User-mode Linux, in a range of micro-benchmarks and system-wide tests (Barham et al, 2003). As mentioned above, these benchmarks included SPEC CPU2000, OSDB, dbench and SPECWeb, benchmarking CPU, database, filesystem and web server performance. The developers of Xen also have some performance benchmarking statistics on their website[21] where they show the results of comparisons to native Linux, VMware Workstation 3.2, and User-mode Linux. These indicate a performance gain using the paravirtualisation approach over VMWare and User-mode Linux.

(Clark et al, 2004) extends the paper of (Barham et al, 2003) and provides good insights into benchmarking tools and techniques. In their study, they repeat (Barham et al, 2003)'s performance analysis of Xen and also extend the analysis in several ways, including comparing XenoLinux on x86 to an IBM zServer. (Clark et al, 2004) state that they use their study as an example of repeated research and argue that this model of research, which is enabled by open source software, is an important step in transferring the results of computer science research into production environments.

(Clark et al, 2004) state that they had great difficulty in replicating the Xen team's results. Specifically, (Clark et al, 2004) had trouble replicating the physical host environment and settled for a different SCSI controller than used by (Barham et al, 2003). Secondly, (Clark et al, 2004) did not replicate the performance tests exactly; substituting some of the closed source tests with their own equivalent. They also comment that reproducing or extending existing research validates the original published work. This would also result in additional insights beyond the scope of the original research. Reproducing an existing body of research also serves as a third-

[21] www.xen.org

party verification of the original results and serves as a platform for transferring research in to a production environment (Clark et al, 2004).

(Clark et al, 2004) also highlighted the fact that commercial software can sometimes be extremely expensive and free alternatives exist which could be used to accomplish similar research objectives.

(Clark et al, 2004) performed a set of scalability tests on low-end commodity hardware and produced good results, indicating that performance testing is possible as long as the number of guest environments is kept low. A number of tests could be applied during the benchmarking process and (Clark et al, 2004) tried to replicate the tests of (Barham et al, 2003) as closely as possible. They even went as far as replicating the specific hardware used by (Barham et al, 2003). (Clark et al, 2004) specifically discuss their approach to replacing closed or proprietary testing suites with open source equivalents.

(Clark et al, 2004) were able to repeat the performance measurements of (Barham et al, 2003) and found that Xen lives up to its claim of high performance virtualisation of the x86 platform. They also found that Xen can easily support 16 moderately loaded servers on a relatively inexpensive server class machine, but did not manage the 100 guest target they set. (Clark et al, 2004) found that Xen performs well in tests on an older PC and that Xen on x86 compares surprisingly well to an entry model IBM zServer machine designed specifically for virtualisation.

As previously stated in section 2.2.2, a paravirtualised machine gives the guest operating system access to the host operating system's hardware resources through a hypervisor. It could therefore be inferred that benchmarking the performance of a paravirtualised machine is possible in much the same way as benchmarking the physical machine. As previously stated in section 2.2.3, operating system virtualisation is simply identical guest operating systems with access to the host operating system's kernel. It could therefore be inferred that benchmarking the performance of this virtual environment is possible in much the same way as benchmarking the physical machine.

(Makhija et al, 2006) state that these benchmarks however, were specifically designed for physical machines and not for virtualised machines. (Makhija et al, 2006) continue by stating that because these benchmarks test hardware under load to prove that they are capable of a certain performance, this may not be applicable to test a guest operating system on a host machine running multiple virtual guest machines at the same time as the results could be misleading. (Barham et al, 2003) and (Clark et al, 2004) were successful at benchmarking virtual systems and published their findings. The benchmarks used in this research are discussed in section 3.3.

2.4 Summary

Many forms of virtualisation are available, and these are offered as proprietary as well as open source solutions each with its own pros and cons.

The methodology and results of performance testing of paravirtualised systems were examined and form the basis of this research, as well as extending the study to include performance testing of operating system virtualisation, using OpenVZ.

Although (Clark et al, 2004)'s research was successful and corroborated the findings of (Barham et al, 2003) they had trouble replicating the environment and tests exactly, substituting some of the closed source tests with their own equivalent.
Some areas of concerns highlighted in the literature include the difficulty of applying a benchmark test to a virtual system running on a host with several other guests.

Some benchmarking software is not freely available and could be a stumbling block if no open source alternative is available. It is important to identify a suitable suite of benchmarking tests that could be applied fairly to a physical system as well as a paravirtualised system and an operating system virtualised environment in order to produce comparative results.

Chapter 3: Method, Analysis and Evaluation

The previous chapter contained an overview about how both Xen and OpenVZ are structured as well as highlighting their respective approach to virtualisation.

The scope of this experiment was to test each of these forms of virtualisation and compare them to each other as well as a physical system. This chapter is dedicated to the discussion of these tests, starting with a description of each environment. This is following by a description of benchmarking as well as a discussion on running and interpreting the various open source benchmark results.

As stated, the objective is to provide a performance comparison by applying a standard set of benchmarks across all environments. The outcome of these benchmarking tests will then be presented by summary results in chapter 4. The detailed results are presented in the appendices.

3.1 The Physical and Virtual Environments

The physical system will serve as both a test environment as well as host to the two virtual environments. In order for the objectives of the experiment to be met, certain conditions have to exist to ensure that each benchmark test within each environment is applied as consistently as possible.

Debian GNU/Linux (Etch) was chosen as the operating system for both the physical and virtual environments. There are many reasons for this choice, the main ones being:

- Debian GNU/Linux uses mainly free software in their repository.
- OpenVZ and Xen are available in binary (.deb) form.
- It is used as the basis for derivative Linux systems such as Ubuntu and Knoppix.

The following hardware system was used for the installation of a host environment:

- AMD Athlon XP 3000+ 2165MHz 333MHz FSB CPU
- 512KB L2 Cache
- Gigabyte GA-7VM400M-RZ Socket A Motherboard
- 1GB DDR333 PC2700 Memory
- 40GB Seagate IDE Hard Drive
- Onboard 10/100MBit Ethernet

The host was set up as follows:

OS	:	Debian GNU/Linux (Etch)
Kernel	:	2.6.18-4-486
C compiler	:	gcc version 4.1.2 20061115 (prerelease) (Debian 4.1.1-21)
libc	:	libc-2.3.6.so

The test environment was set up to allow the optimal allocation of physical resources as follows:

A physical host with a single Xen and a single OpenVZ guest was created.
Identical resources were configured for the virtualised Xen and OpenVZ environments by modifying the configurations files of each to match the physical hardware described above as close as possible (see Appendix D and E).

The following open source benchmarks were chosen to measure the performance of each environment:

- lmbench3-alpha1

- BYTEmark* Native Mode Benchmark ver. 2 (10/95)
  Index-split by Andrew D. Balsa (11/97)
  Linux/Unix* port by Uwe F. Mayer (12/96,11/97)

- UnixBench Version 4.0.1 -- 1997.06.20

Apart from the base installation of the Debian GNU/Linux operating system, the following packages were installed from the local Debian repository:

Xen packages

- xen-hypervisor-3.0.3-1-i386-pae_3.0.3-0-4
- xen-linux-system-2.6.18-6-xen-686_2.6.18.dfsg.1-22etch2
- 2.6.18-6-xen-686 (Xen domain0 kernel)

OpenVZ packages

- vzctl 3.0.22-1dso1
- vzquota 3.0.11-1dso1
- vzctl-ostmpl-debian-4.0-i386-minimal_20080518
- 2.6.18-12-fza-686 (OpenVZ kernel)

Both the Xen and OpenVZ guests were set up with a base installation of the Debian GNU/Linux operating system. Additionally, the guest environments have also been provided with the GNU C Compiler gcc version 4.1.2 20061115 (prerelease) (Debian 4.1.1-21) in order to compile the benchmarks from source.

Benchmarking each environment, physical and virtual, will be discussed in detail in section 3.3. The host and the guests have been installed on one hard disk. OpenVZ was installed in physical folder whereas Xen was installed in an image.

Each environment, physical and virtual makes use of the third extended filesystem, (ext3)[22]. The ext3-filesystem was chosen as it is widely used by a variety of Linux variants, including Debian GNU/Linux.

The `/etc/apt/souces.list` file contains the list of repositories used

```
# DEBIAN SECURITY REPOSITORY
deb http://security.debian.org/ etch/updates main contrib
deb-src http://security.debian.org/ etch/updates main contrib

# DEBIAN MAIN AND CONTRIB REPOSITORY
deb http://debian.mirror.ac.za/debian etch main contrib
deb-src http://debian.mirror.ac.za/debian etch main contrib

# OpenVZ REPOSITORY
deb http://download.openvz.org/debian-systs etch openvz
```

[22] http://olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.html

These repositories were used on the host as well as the OpenVZ and Xen virtual environments.

The OpenVZ installation requires a custom kernel to be installed on the physical host as well as the OpenVZ administration tools. These customised kernels and administration tools are available directly from OpenVZ as well as third party providers. OpenVZ guests are created using pre-built templates, and therefore the OpenVZ template cache needs to be created as well. Once these requirements are met, OpenVZ guest environments can be created. Detailed installation requirements and instructions are available in the OpenVZ manual[23]. OpenVZ was installed and the resources were allocated to the guest.

The physical host configuration, as described in section 3.1, does not contain CPU extensions for hardware virtualisation, therefore, an unmodified installation of Debian GNU/Linux is not possible. As was the case with the OpenVZ kernel and administration tools, the equivalent Xen kernel and administration tools are also available in binary form from the Debian repository. Once the necessary software is installed, the host will become the privileged domain, referred to as Dom0, or Domain0 in Xen terms. Dom0 contains the hypervisor, facilitating paravirtualisation of virtual environments. The unprivileged virtual environments are known as DomU or DomainU in Xen terms. Xen was installed and the resources were allocated to the guest.

[23] http://ftp.openvz.org/doc/OpenVZ-Users-Guide.pdf

3.2 Benchmarking

Having completed the installation, it was possible to test the relative performance and distribution of resources of both virtual environments by using benchmark tests.

The aim of this thesis is to quantify the performance overhead of paravirtualisation and OS virtualisation. The best way to achieve these results is through simulating workloads targeting various hardware and system components and interpreting the output.

The benchmarks selected need to be freely available and easy to install and execute. Preference would be given to standard, widely used benchmarks. (Barham et al, 2003) and (Clark et al, 2004) used lmbench in their analysis as it includes both latency and throughput tests. A general overview on several benchmark tests available can be found at http://lbs.sourceforge.net/. Each benchmark test was performed five times on the host, Xen guest and OpenVZ guest to determine consistency. The comparative "score" for each benchmark is the arithmetic average of each set of individual tests. It is important to note that these benchmarks are software applications running on an operating system and may be affected by other running processes, resulting in a margin of error. The standard deviation of each of the five tests was calculated to measure this variation (see appendices for the detailed results tables).

The host score was based to 1 and each guest's over- or under-performance is expressed relative to this index.

3.2.1 Synthetic Benchmarks

Synthetic benchmarks are designed to measure the performance of individual components of computer hardware. A good example of a synthetic benchmark is the Whetstone suite, originally programmed by Harold Curnow in 1972 (Curnow et al, 1976). This benchmark measures the floating-point performance of a CPU.

According to (Balsa, 1997), the main criticism of synthetic benchmarks is that they do not represent performance of real-world situations. For instance, the Whetstone suite's main loop is very short and easily fits in the primary cache of a CPU, keeping the FPU queue filled and testing it to its full capability (Balsa, 1997). The interpretation of synthetic benchmarking results must therefore be done very carefully when testing modern CPUs.

According to (Balsa, 1997), synthetic benchmarks should test the performance of hardware components in isolation. (Balsa, 1997) gives the example of benchmarking Ethernet card I/O on various hardware configurations and states that the results should be relatively similar.

3.2.2 High-level vs. Low-level Benchmarks

According to (Balsa, 1997), low-level benchmarks directly measures the performance of hardware components like the CPU clock, memory cycle times, average hard disk access times, latency, etc.

(Balsa, 1997) states that high-level benchmarks are used for evaluating the performance of the hardware driver and operating system, for a specific aspect of a physical computer system. For example, file I/O performance or application benchmarking.

Low-level benchmarks are classified as synthetic whereas high-level benchmarks may be synthetic or application benchmarks (Balsa, 1997).

3.3 Benchmarking Software Selected

The benchmarks selected for this project were lmbench, nbench/BYTEMark and UnixBench. These were chosen because they met the criteria of being freely available, widely used, ease of installation and well documented test output. These benchmarks also met the criteria of being synthetic benchmarks as discussed in section 3.2.1. The aim of this thesis is not to quantify the actual workloads experienced by the physical and virtual environments, rather it aim is to quantify the extent of performance lost by using paravirtualisation and OS virtualisation. Synthetic benchmarks are well suited to deliver these tests.

(Barham et al, 2003) and (Clark et al, 2004) made use of lmbench in their evaluations of Xen and using lmbench in this study would allow for comparative analysis with their findings.

Nbench/BYTEMark and UnixBench, like lmbench also offer throughput tests and would provide a useful way of confirming the results produced by the lmbench tests.

3.4 lmbench

lmbench is described as a GPL'd suite of atomic benchmarks developed by Larry McVoy and has no publishing restrictions.

According to (McVoy and Staelin, 1996), lmbench provides a suite of benchmarks that attempt to measure the most commonly found performance bottlenecks in a wide range of system applications. (McVoy and Staelin, 1996) identified these bottlenecks, then isolated, and reproduced them in a set of small 'microbenchmarks'. These microbenchmarks measure system latency and bandwidth of data movement among the processor and memory, network, file system, and disk. The intent was to produce numbers that real applications would reproduce, instead of the less easily reproducible marketing material claimed by hardware manufacturers.

(McVoy and Staelin, 1996) state that performance issues are usually caused by latency problems, bandwidth problems, or some combination of the two. Each benchmark exists because it captures some unique performance problem present in one or more applications.

(McVoy and Staelin, 1996) state that lmbench measures only a system's ability to transfer data between processor, cache, memory, network, and disk. It does not measure other parts of the system, such as the graphics subsystem, nor is it a MIPS, MFLOPS, throughput, saturation, stress, graphics, or multiprocessor test suite.
The benchmarks are designed to be portable and similar over a wide set of UNIX systems.

lmbench is freely distributed under the Free Software Foundation's General Public License, with the additional restriction that results may be reported only if the benchmarks are unmodified (McVoy and Staelin, 1996).
According to the lmbench man pages[24], the microbenchmarks fall into three general classes: bandwidth, latency, and 'other'.

[24] http://lmbench.sourceforge.net/man/lmbench.8.html

The **bandwidth** benchmarks have two main components: operating system overhead and memory speeds. According to the lmbench man pages, the bandwidth benchmarks report their results as megabytes moved per second but cautions that the data moved is not necessarily the same as the memory bandwidth used to move the data.

bw_file_rd	reading and summing of a file via the read(2) interface.
bw_mem_cp	memory copy.
bw_mem_rd	memory reading and summing.
bw_mem_wr	memory writing.
bw_mmap_rd	reading and summing of a file via the memory mapping mmap(2) interface.
bw_pipe	reading of data via a pipe.
bw_tcp	reading of data via a TCP/IP socket.
bw_unix	reading data from a UNIX socket.

***Table 1**: lmbench Bandwidth Test descriptions*

The **latency** measurements are intended to show how fast a system can perform some operation. The pipe, rpc, tcp, and udp transactions are all identical benchmarks. Latency numbers are mostly in microseconds per operation.

lat_connect	the time it takes to establish a TCP/IP connection.
lat_ctx	context switching; the number and size of processes is varied.
lat_fcntl	fcntl file locking.
lat_fifo	'hot potato' transaction through a UNIX FIFO.
lat_fs	creating and deleting small files.
lat_pagefault	the time it takes to fault in a page from a file.
lat_mem_rd	memory read latency (accurate to the ~2-5 nanosecond range, reported in nanoseconds).
lat_mmap	time to set up a memory mapping.
lat_ops	basic processor operations, such as integer XOR, ADD, SUB, MUL, DIV, and MOD, and float ADD, MUL, DIV, and double ADD, MUL, DIV.
lat_pipe	'hot potato' transaction through a Unix pipe.
lat_proc	process creation times (various sorts).
lat_rpc	'hot potato' transaction through Sun RPC over UDP or TCP.
lat_select	select latency
lat_sig	signal installation and catch latencies. Also protection fault signal latency.
lat_syscall	non trivial entry into the system.
lat_tcp	'hot potato' transaction through TCP.
lat_udp	'hot potato' transaction through UDP.
lat_unix	'hot potato' transaction through UNIX sockets.
lat_unix_connect	the time it takes to establish a UNIX socket connection.

***Table 2**: lmbench Latency Test descriptions*

Other measurements:

mhz	processor cycle time
tlb	TLB size and TLB miss latency
line	cache line size (in bytes)
cache	cache statistics, such as line size, cache sizes, memory parallelism.
stream	John McCalpin's stream benchmark
par_mem	memory subsystem parallelism. How many requests can the memory subsystem service in parallel, which may depend on the location of the data in the memory hierarchy.
par_ops	basic processor operation parallelism.

***Table 3**: lmbench 'Other' Test descriptions*

3.5 BYTEmark* Native Mode Benchmark (nbench)

(Helvick, 2008) states that nbench is based on BYTE Magazine's BYTEMark benchmark program. NBench is a synthetic benchmark used to test the CPU, GPU and memory. The tests include reporting CPU, cache and memory, integer and floating-point results.

NBench runs ten single-threaded tests, including integer and string sorting, Fourier coefficients and Huffman compression. Scores in nbench represent measurements baselined to those of a Pentium90 and AMD K6/233. According to (Williams, 2000), BYTEmark reports both raw and indexed scores for each test. The example given by (Williams, 2000) says that the numeric sort test reports the number of arrays it was able to sort per second as its raw score. The indexed score is the raw score divided by the raw score of the baseline machine, a 90Mhz Pentium with 16MB of memory. The index score is an effort to normalise the raw scores, therefore an index score of 2 can be interpreted as performing twice as fast as the baseline (Williams, 2000). The unit of measure is number of iterations per second, compared to the baseline as discussed above (Helvick, 2008).

BYTEmark reports an Integer and Floating-point index. The integer index is the geometric mean of the tests that involve integer processing. The floating-point index is the geometric mean of the remaining tests (Williams, 2000).

A unique feature of nbench is that it analyses its own results for confidence levels in real-time and increases the number of runs if necessary. Theoretically, this means that nbench could be run on a system under load and still produces accurate results, although it may produce a larger variance and take longer to run.

nbench consists of 10 tests:

Numeric sort	Integer-sorting benchmark
String sort	String-sorting benchmark
Bitfield	Bit manipulation package
Emulated floating-point	Small software floating-point package
Fourier coefficients	Numerical analysis benchmark for calculating series approximations of waveforms
Assignment algorithm	Task allocation algorithm
Huffman compression	Well-known text and graphics compression algorithm
IDEA encryption	Block cipher encryption algorithm
Neural net	Black-propagation network simulator
LU Decomposition	Robust algorithm for solving linear equations

***Table 4**: nbench Test descriptions*

3.6 UnixBench

According to the byte-unixbench[25] website, UnixBench development started in 1983 at Monash University, as a simple synthetic benchmarking application. Linux compatibility modifications were made by Jon Tombs, and original authors Ben Smith, Rick Grehan, and Tom Yager. Similar to nbench, the tests evaluate UNIX-like systems by comparing their results to a set of scores set by running the code on a baseline system, which in this case is a SPARCstation 20-61, indexed at a value of 10.0. The entire set of index values is then combined at the end of the test to make an overall index for the system.

(Hatt et al, 2007) state that the purpose of UnixBench is to provide a straightforward indicator of the performance of a UNIX-like system; hence, multiple small tests are used to evaluate various aspects of the system's performance. According to (Hatt et al, 2007), UnixBench is a system benchmark, not a CPU, memory or disk benchmark and focuses on different aspects of OS functionality like process spawning, inter-process communication and filesystem throughput. The results will depend not only on the physical hardware, but on the operating system, libraries, and even compiler.

[25] http://code.google.com/p/byte-unixbench/

UnixBench consists of nine tests, described in the flowing table:

Dhrystone	Focuses on string handling, as there are no floating point operations.
Whetstone	Measures the speed and efficiency of floating-point operations using a wide variety of C functions including sin, cos, sqrt, exp, and log as well as integer and floating-point math operations, array accesses, conditional branches, and procedure calls. This test measure both integer and floating-point arithmetic.
Execl Throughput	Measures the number of execl calls that can be performed per second. Execl is part of the exec family of functions that replaces the current process image with a new process image.
File Copy	Measures the rate at which data can be transferred from one file to another, using various buffer sizes. File read, write and copy tests capture the number of characters that can be written, read and copied in a specified time (default is 10 seconds).
Pipe Throughput	Pipe throughput is the number of times (per second) a process can write 512 bytes to a pipe and read them back.
Pipe-based Context Switching	Measures the number of times two processes can exchange an increasing integer through a pipe. The test program spawns a child process with which it carries on a bi-directional pipe conversation.
Process Creation	Measures the number of times a process can fork and reap a child that immediately exits. Process creation refers to actually creating process control blocks and memory allocations for new processes, so this applies directly to memory bandwidth.
Shell Scripts	Measures the number of times per minute a process can start and reap a set of one, two, four and eight concurrent copies of a shell script where the shell script applies a series of transformations to a data file.
System Call Overhead	Estimates the cost of entering and leaving the operating system kernel, i.e. the overhead for performing a system call. It consists of a simple program repeatedly calling the *getpid*[26] system call. The time to execute such calls is used to estimate the cost of entering and exiting the kernel.

***Table 5**: UnixBench Test descriptions*

[26] returns the process id of the calling process

3.7 Evaluation

As discussed in section 3.3, lmbench3, nbench/BYTEMark and UnixBench were chosen to test and evaluate the physical host, OpenVZ virtual environment and Xen virtual environment on metrics covering CPU utilization, inter-process communication, hard disk access and network performance.

Once Xen and OpenVZ guests had been successfully created and tested, all benchmark tests were first applied to the host without any guests in place. Thereafter each guest was created separately and all the benchmark tests applied to it.

As highlighted in section 3.2, each benchmark test was performed five times on the host, Xen guest and OpenVZ guest to determine consistency. Each test produced a set of results and these were captured for analysis.

The comparative "score" for each benchmark is the arithmetic average of each set of individual tests. Since the benchmarking software itself is run on an operating system and may be affected by other running processes, introducing a margin of error, the standard deviation of each of the five tests was calculated.

Chapter 4: Results

As discussed in section 3.7, the results of each set of five tests for each benchmark, within each environment, were captured and summarised for numerical analysis. Each benchmark will be discussed individually with summary results describing the outcome of each test or group of tests. Detailed results tables are available in Appendix A, B and C.

4.1 lmbench Results

As described in section 3.2, the individual and averaged lmbench3 test results are included in Appendix A. A brief description of each test is included in section 3.4 (Tables 1, 2 and 3).

As discussed in section 3.4, lmbench only measures a system's ability to transfer data between processor, cache, memory, network, and disk.
The tests can be grouped in to five categories:
CPU and process tests, Context switching tests, Communication tests and File and VM tests are all latency tests, and therefore smaller index numbers are better.
On the other hand, for Communication Bandwidth tests, bigger index numbers are better as they are measuring throughput.

lmbench CPU & Processes Results:

Both guests produced marginally higher latency results than the host in the following tests:

- *null I/O* test, a simple read and write benchmark, (host 0.21, OpenVZ 0.27, Xen 0.24 microseconds)
- *stat* test, a simple stat and fstat benchmark which retrieves information about a file, (host 1.41, OpenVZ 1.56, Xen 1.38 microseconds)
- *open clos* test, a simple file operation benchmark, (host 2.24, OpenVZ 2.68, Xen 2.36 microseconds)
- *slct TCP* test, a select() system call using TCP, (host 12.42, OpenVZ 10.84, Xen 6.76 microseconds)
- *sig inst* test, signal handler install benchmark, (host 0.35, OpenVZ 0.45, Xen 0.43 microseconds)
- *sig hndl* test, a signal handler overhead benchmark, (host 1.43, OpenVZ 1.53, Xen 1.32 microseconds)

These scores indicate there is negligible performance loss incurred by either virtualisation layer when executing the above operations. Better than native results were produced by both guests in the *slct TCP* test and by Xen in the *sig hndl* test.

OpenVZ produced marginally poor results and Xen produced materially poor results in the following tests:

- *null call* test, a simple system call, (host average 0.09, OpenVZ average 0.13, and Xen average 0.15 microseconds)
- *fork proc* test, a process forking benchmark, (host 101.44, OpenVZ 145.20, and Xen 420.60 microseconds)
- *exec proc* test, a process execution benchmark, (host 349.40, OpenVZ 472.80, and Xen 1100.80 microseconds)
- *sh proc* test, a process creation benchmark, (host 2188.20, OpenVZ 2677.60, and Xen 3787.80 microseconds)

With the exception of the process-based tests, both guests produce relatively close to native test results. However, the magnitude of underperformance of the process-based tests does raise some questions on the ability of Xen to cope with process-based latency demands. This is an expected result as the hypervisor incurs an extra level of overhead between the kernel and guest OS. As discussed in section 2.2.3, OpenVZ does not require a VMM and performs relatively consistently. Compared to the CPU benchmarking done by (Barham et al, 2003) and (Clark et al, 2004) the Xen results are expected. The process-based results are consistent with the Linux build time tests performed by (Barham et al, 2003) and (Clark et al, 2004), where an underperformance was reported compared to the physical host, however, not to the same degree.

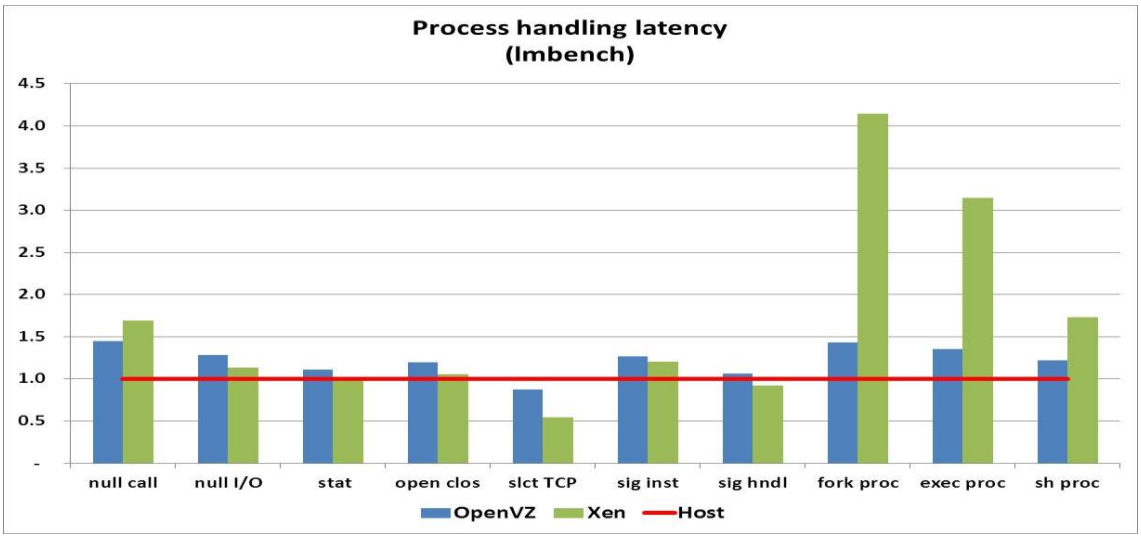

***Figure 6**: Process handling latency*

lmbench Context Switching Results

A context switch is the switching of the CPU from one process or thread to another. In terms of these tests, the context switching benchmark expects two parameters; the size of the process, in KB and the number of processes to simulate. A 2p/0K test would therefore simulate context switching of two processes of 0K in size.

Both guests produced marginally higher latency results than the host in the following tests:

- *2p/64K* test, 2 processes of 64K in size, (host average 5.64, OpenVZ average 5.81, Xen average 8.71 microseconds)
- *8p/64K* test, 8 processes of 64K in size, (host 38.04, OpenVZ 40.46, Xen 53.74 microseconds)
- *16p/64K* test, 16 processes of 64K in size, (host 63.58, OpenVZ 65.02, Xen 72.06 microseconds)

Xen produced materially poor results in the following tests (OpenVZ performed consistently as above):

- *2p/0K* test, 2 processes of 0K in size, (host average 1.14, OpenVZ average 0.86, Xen average 4.17 microseconds)
- *2p/16K* test, 2 processes of 16K in size, (host 0.93, OpenVZ 1.32, Xen 4.39 microseconds)
- *8p/16K* test, 8 processes of 16K in size, (host 2.41, OpenVZ 2.65, Xen 6.42 microseconds)
- *16p/16K* test, 16 processes of 16K in size, (host 2.91, OpenVZ 3.41, Xen 8.14 microseconds)

OpenVZ produces relatively close to native performance on all context switching tests and manages to produce better than native results on one of the tests. Xen underperforms the host on all the tests and produces exceptionally poor results for four of the seven tests in this group. This is consistent with the results produced by (Barham et al, 2003) and indicates the extra overhead of the hypervisor layer.

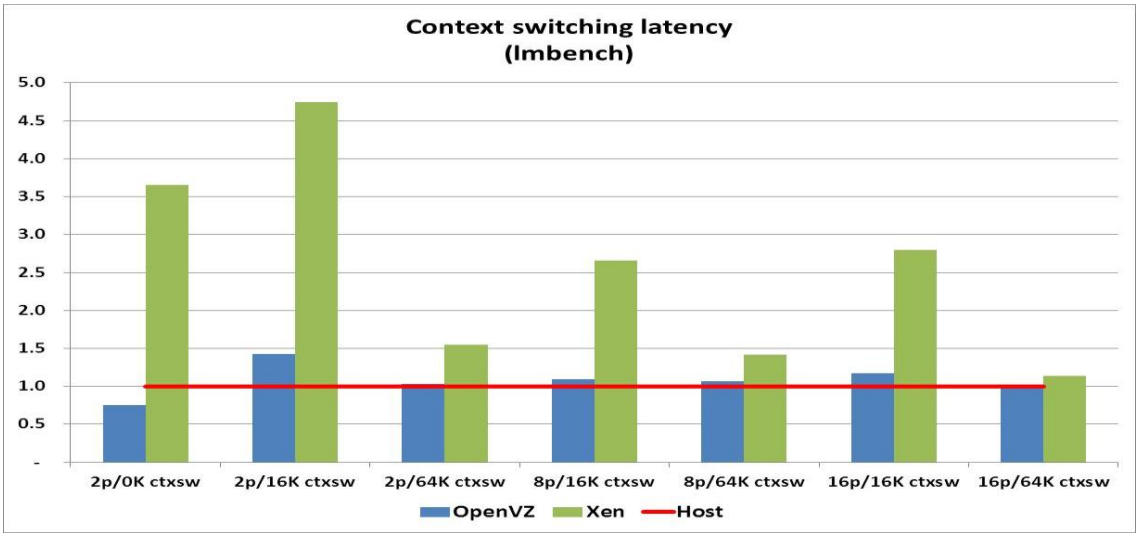

***Figure 7**: Context switching latency*

lmbench Communication Latencies Results

Xen produced materially poor results in the following tests and OpenVZ unperformed materially on one of the following tests:

- *UNIX Pipe* test, which simulates one process writing to the standard output and another process reading from the standard input, (host average 4.54, OpenVZ average 5.10, Xen average 21.90 microseconds)
- *AF UNIX* test, 2 a local socket for communication between applications on the same OS, (host 6.46, OpenVZ 7.59, Xen 15.58 microseconds)
- *UDP* test, a network communications benchmark, (host 7.71, OpenVZ 13.10, Xen 15.36 microseconds)

Both guests produced marginally higher latency results than the host in the following tests:

- *TCP* test, a network communications benchmark, (host average 13.30, OpenVZ average 19.28, Xen average 20.26 microseconds)
- *TCP conn* test, a network communications connection benchmark, (host 42.20, OpenVZ 58.26, Xen 63.20 microseconds)

Overall, OpenVZ produced near native results. Xen performed worse than both the host and the OpenVZ guest, producing exceptionally poor results for four of the six tests in this group. (Barham et al, 2003) did not publish comparative test results for local communication latencies.

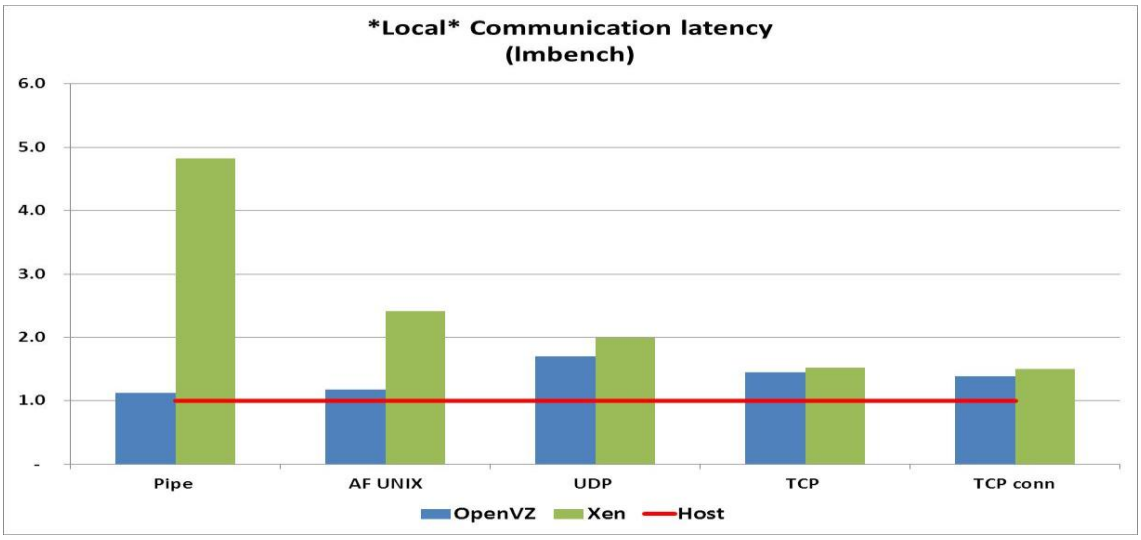

***Figure 8**: Local Communication latency*

lmbench File & VM Latencies Results

OpenVZ produced marginally higher latency results than the host in the following tests; Xen outperformed the host on the following tests:

- *0K File Create* test, a file creation benchmark, (host average 14.14, OpenVZ average 16.66, Xen average 9.96 microseconds)
- *0K File Delete* test, a file deletion benchmark, (host 11.92, OpenVZ 15.00, Xen 8.19 microseconds)
- *10K File Create* test, a file creation benchmark, (host 64.56, OpenVZ 74.10, Xen 56.76 microseconds)
- *10K File Delete* test, a file deletion benchmark, (host 25.62, OpenVZ 31.72, Xen 17.16 microseconds)
- *Prot Fault* test, a general protection fault test, (host 0.77, OpenVZ 1.01, Xen 0.44 microseconds)
- *100fd selct* test, benchmarks the time to do a select on 100 file descriptors, (host 2.63, OpenVZ 2.73, Xen 2.80 microseconds)

Both guests produced materially higher latency results than the host in the following tests:

- *Mmap Latency* test, benchmarks a system call that maps files or devices into memory, (host average 4157.80, OpenVZ average 10865.80, Xen average 15640.00 microseconds)
- *Page Fault* test, benchmarks the cost of pagefaulting pages from a file, (host 1.99, OpenVZ 3.69, Xen 4.92 microseconds)

It would appear that Xen does not suffer from the same latency issues in this group of tests compared to the other latency tests performed above, beating the host on five of the eight tests in these tests. The underperformance by Xen in the two tests above is consistent with the results of (Barham et al, 2003).

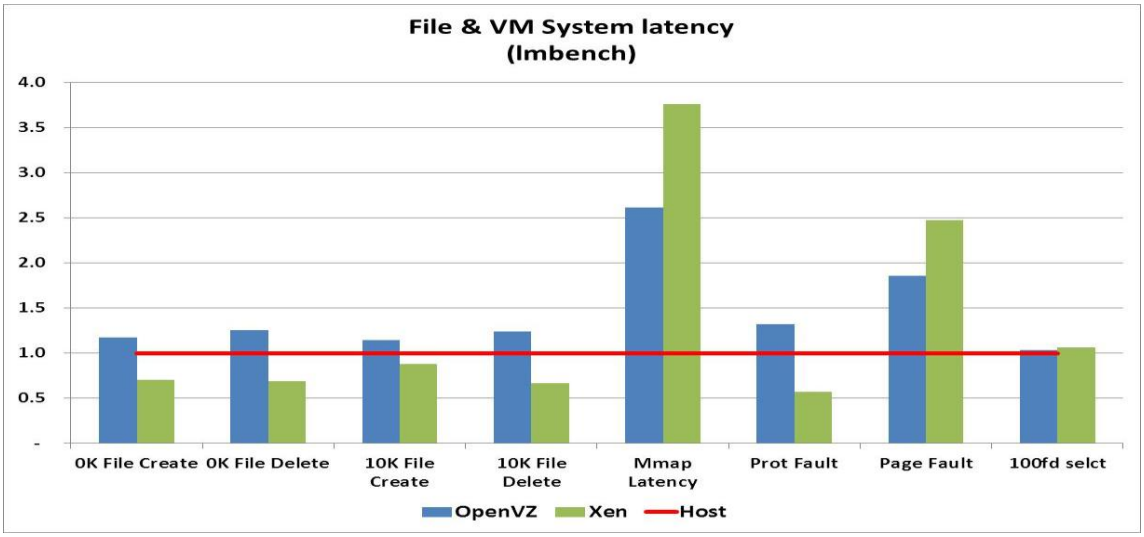

***Figure 9**: File & VM System latency*

lmbench Communication Bandwidths Results

Both guests produced marginally lower throughput results than the host in the following tests:

- *UNIX Pipe* test, which simulates one process writing to the standard output and another process reading from the standard input, (host average 1124.00, OpenVZ average 1044.00, Xen average 662.60 MB/second)
- *AF UNIX* test, which benchmarks local communication on a socket between applications on the same OS, (host 1239.20, OpenVZ 1179.80, Xen 1153.80 MB/second)
- *TCP* test, a network communications bandwidth benchmark, (host 214.00, OpenVZ 193.60, Xen 199.00 MB/second)
- *File reread* test, a simple file operation, (host 551.16, OpenVZ 541.20, Xen 511.16 MB/second)
- *Mmap reread* test, which benchmarks a system call that maps files or devices into memory, (host 1040.70, OpenVZ 1025.20, Xen 1028.00 MB/second)
- *Bcopy (libc)* test, benchmarks the user-level library bcopy interface, (host 376.68, OpenVZ 363.06, Xen 366.72 MB/second)
- *Bcopy (hand)* test, a loop that loads and stores associated 8-byte words, (host 377.70, OpenVZ 357.24, Xen 368.12 MB/second)
- *Mem read* test, a loop that sums up a series of integers, (host 1014.80, OpenVZ 1000.00, Xen 1001.60 MB/second)
- *Mem write* test, a loop that stores a value as an integer and then increments the pointer, (host 637.94, OpenVZ 610.20, Xen 618.86 MB/second)

These results show good local communication throughput with low overhead incurred by the virtualisation layer. (Barham et al, 2003) only reported the TCP results and these result concur with their findings.

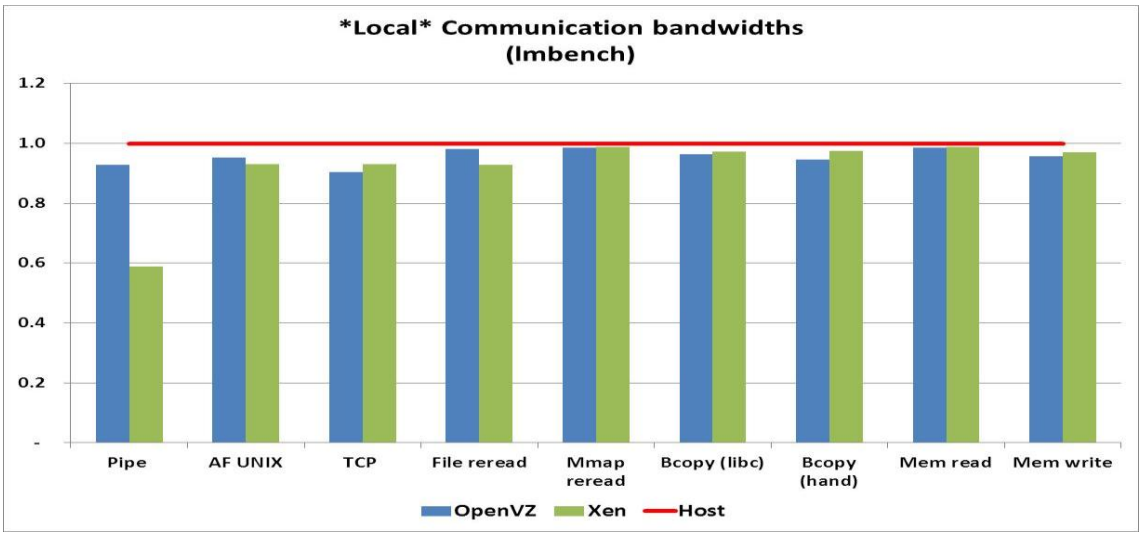

***Figure 10**: Local Communication bandwidth throughput*

4.2 nbench/BYTEmark* Native Mode Benchmark Results

As described in section 3.2, the individual and averaged nbench/BYTEMark test results are included in Appendix B. A brief description of each test is included in section 3.5 (Table 4). As discussed in section 3.5, nbench scores are compared to a baseline Pentium90, higher scores indicate better performance and the units are iterations per second.

The nbench benchmark involves only user-level CPU workload and therefore does not cause kernel level memory accesses nor trigger kernel level services. For the purposes of this evaluation, the 10 individual tests as well as the composite indices as described in section 3.2 will be discussed.

Both guests produced marginally lower throughput results than the host in the following tests:

- *Numeric Sort* test, an integer sorting benchmark, (host average 24.76, OpenVZ average 24.94, Xen average 24.81 iterations /second)
- *String Sort* test, a string sorting benchmark, (host 73.75, OpenVZ 73.47, Xen 73.60 iterations /second)
- *Bitfield* test, a bit manipulation package, (host 82.19, OpenVZ 80.98, Xen 82.90 iterations /second)
- *Emulated Floating-point* test, a small software floating point benchmark, (host 75.91, OpenVZ 75.75, Xen 75.90 iterations /second)
- *Fourier Coefficients* test, a numerical analysis benchmark for calculating series approximations of waveforms, (host 24.25, OpenVZ 24.15, Xen 24.24 iterations /second)
- *Assignment Algorithm* test, a task allocation benchmark, (host 89.95, OpenVZ 89.52, Xen 89.88 iterations /second)
- *IDEA Encryption* test, a block cipher encryption algorithm, (host 55.59, OpenVZ 55.44, Xen 55.64 iterations /second)
- *Huffman Compression* test, a text and graphics compression algorithm, (host 44.33, OpenVZ 44.18, Xen 44.33 iterations /second)

- *Neural Net* test, a back-propagation network simulator, (host 50.91, OpenVZ 48.78, Xen 50.37 iterations /second)
- *LU Decomposition* test, an algorithm for solving linear equations, (host 57.74, OpenVZ 56.10, Xen 56.75 iterations /second)

Both guests perform exceptionally well against the host. As mentioned, nbench only tests user-level CPU workload. The VMM layer would incur extra overhead if these tests were targeted at system-level CPU workloads. Neither (Barham et al, 2003) or (Clark et al, 2004) used nbench in their benchmarking. The nbench results do confirm the CPU latency and bandwidth throughput test results produced by lmbench in section 4.1.

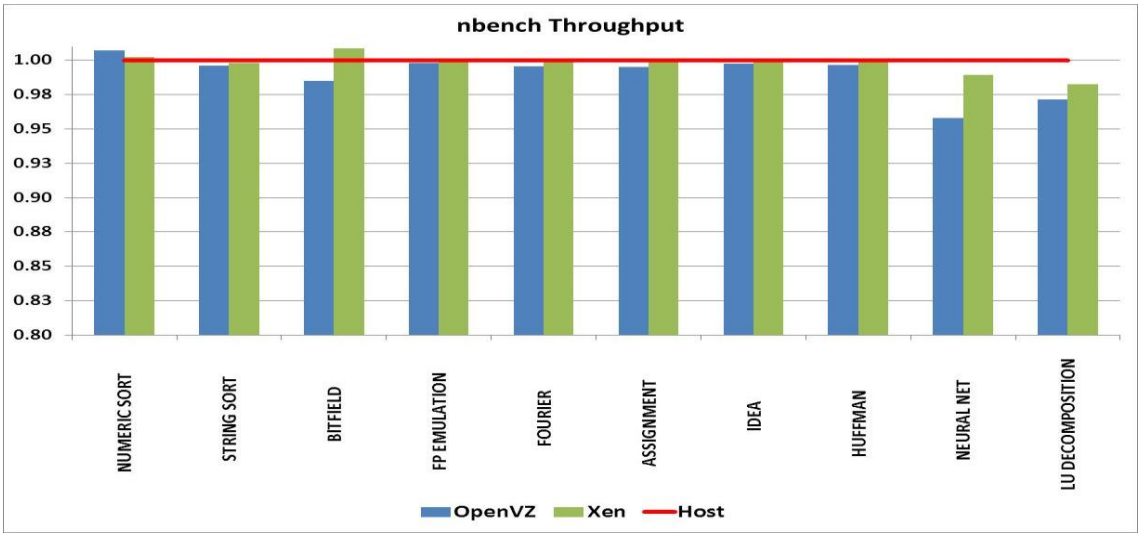

***Figure 11**: nbench/BYTEMark throughput.*

Nbench/BYTEMark Composite Index Results

After running the 10 tests, BYTEMark produces three composite indices, Memory Index, Integer Index and Floating Point Index. The two important indices relate to Integer and Floating Point tests.
The Integer Index is the geometric mean of the integer processing tests; Numeric Sort, String Sort, Bitfield, Emulated Floating Point, Assignment Algorithm, Huffman Compression and IDEA Encryption. The Floating Point Index is the geometric mean of the remaining tests.

Both guests produce marginally lower throughput scores than the host in the following tests:

- *Memory Index* test, the geometric mean of memory test scores, (host average 16.60, OpenVZ average 16.47, Xen average 16.63 iterations /second)
- *Integer Index* test, the geometric mean of integer test scores, (host 13.51, OpenVZ 13.50, Xen 13.52 iterations /second)
- *Floating-point Index* test, the geometric mean of floating-point test scores, (host 23.00, OpenVZ 22.42, Xen 22.78 iterations /second)

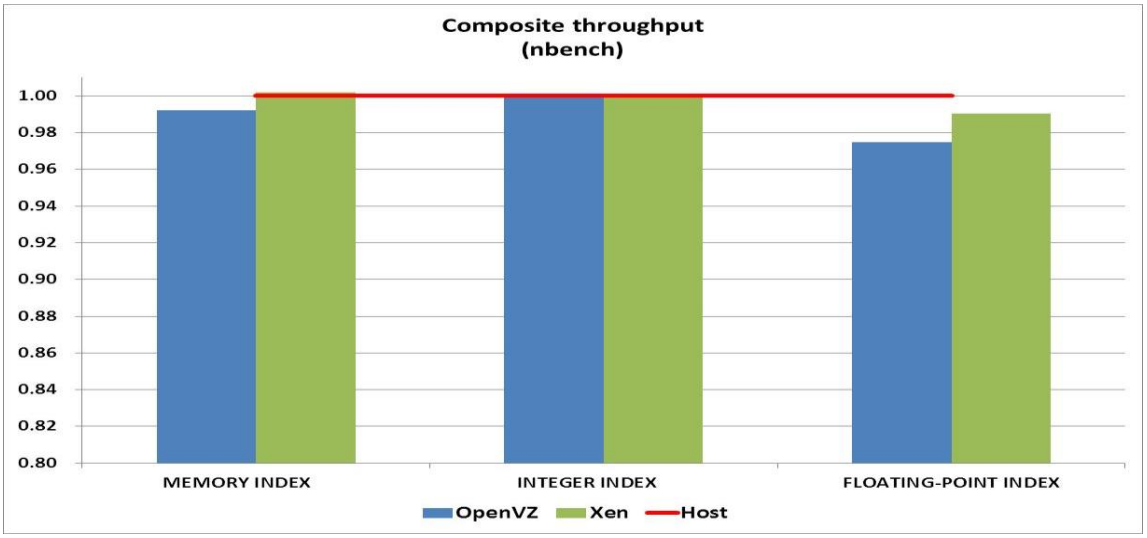

***Figure 12**: nbench/BYTEMark Composite throughput*

4.3 UnixBench Benchmark Results

As described in section 3.2, the individual and averaged UnixBench test results are included in Appendix C. A brief description of each test is included in section 3.6 (Table 5). As discussed in section 3.6, UnixBench, similar to nbench, is baselined to a SPARCstation 20-61 (based to 10), higher index numbers indicate better throughput performance.

UnixBench CPU Benchmark Results

Both guests produce marginally lower throughput scores for the following tests:

- *Arithmetic* test, evaluates assignment, addition, subtraction and multiplication calculations that substitute datatypes for numbers, (host average 290, OpenVZ average 289, Xen average 290 iterations /second)
- *Dhrystone 2* test, evaluates the manipulation of arrays, character strings, indirect addressing, and other common non-floating point instructions, (host 470, OpenVZ 413, Xen 471 iterations /second)

Both guests produce materially lower throughput scores for the following tests:

- *Execl Throughput* test, evaluate the replacement of a currently running process with a new process, (host average 1298, OpenVZ average 841, Xen average 389 iterations /second)

The results are a mixed bag and indicate where each of the virtualisation layers, as used by OpenVZ and Xen, impacts the CPU performance. The Execl Throughput test confirms the results of the process latency tests performed using lmbench in section 4.1. The VMM does indeed impact both latency and throughput for process based tasks.

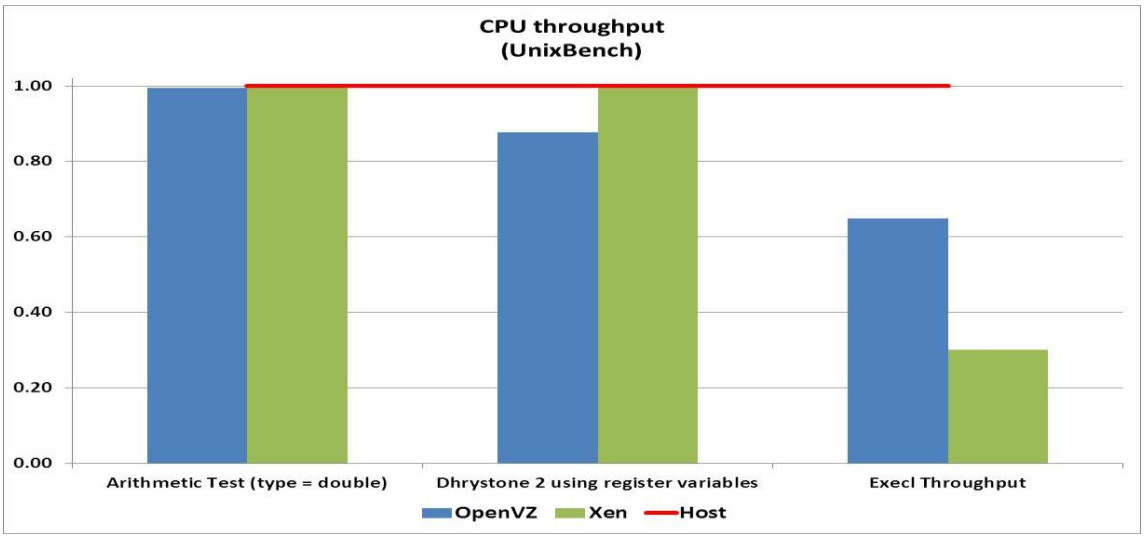

***Figure 13**: CPU throughput*

UnixBench Inter-Process Communication Benchmark Results

Both guests produced results with materially lower throughput than the host in the following tests:

- *Pipe Throughput* test, evaluates a single process which opens a pipe to itself and communicates data in a loop, (host average 824, OpenVZ average 592, Xen average 119 iterations /second)
- *Process Creation* test, evaluates the repeated creation of a child process which immediately dies after its own fork(), (host 1260, OpenVZ 870, Xen 228 iterations /second)
- *Shell Scripts* test, evaluates a shell script that is run by 1, 2, 4, and 8 concurrent processes, (host 786, OpenVZ 718, Xen 485 iterations /second)
- *System Call Overhead* test, evaluates the time required to do iterations of dup(), close(), getpid(), getuid(), and umask() calls, (host 1476, OpenVZ 951, Xen 875 iterations /second)

OpenVZ outperformed Xen on all the IPC tests; however, both guests struggled to produce near native performance results in this group of tests. Using these simulations, the effect of the virtualisation layers during inter-process communication becomes very apparent. This area leaves room for improvement for both virtualisation approaches. The IPC bottleneck highlighted by the simulation could likely escalate as more guests are added to the host and/or the host or guest system(s) is placed under increasing load.

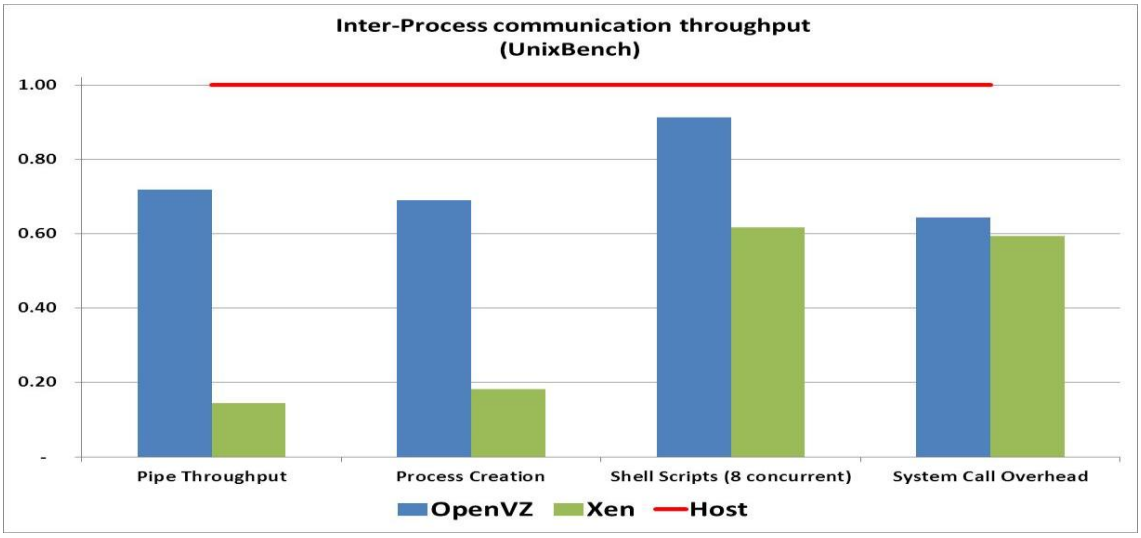

***Figure 14**: Inter-Process communication throughput*

UnixBench Filesystem Benchmark Results

The filesystem tests capture the number of characters that can be copied within 10 seconds based on buffer sizes of 256 bytes, 1 kilobyte and 4 kilobytes.

Both guests performed with materially lower throughput than the host in the following tests:

- *File Copy 256B* test, (host average 566, OpenVZ average 454, Xen average 171 iterations /second)
- *File Copy 1K* test, (host 450, OpenVZ 414, Xen 212 iterations /second)
- *File Copy 4K* test, (host 398, OpenVZ 378, Xen 288 iterations /second)

OpenVZ outperformed Xen on all the Filesystem tests, producing near native results on two of the three tests in this group. These results do not corroborate the Filesystem and VM microbenchmarks in lmbench even though they test similar characteristics. This could be attributed to the way in which UnixBench and lmbench implement their tests. Another explanation for the major difference in the filesystem benchmark results and the severe relative underperformance experience by the Xen guest is OpenVZ was installed as a directory on the host filesystem, whereas Xen was installed as an image, which would incur additional overhead under these simulations.

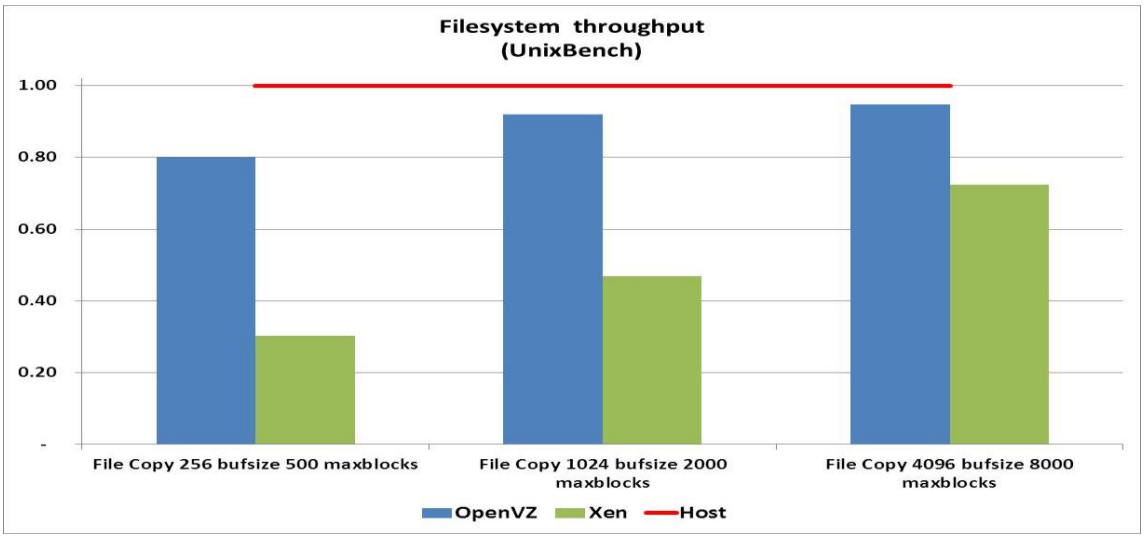

***Figure 15**: Filesystem throughput*

UnixBench Composite Score

The composite score shows a lower throughput by the guests, compared to the host: (host average 679, OpenVZ average 555, Xen average 277 iterations /second).

The UnixBench tests focus on system resources such as CPU, file systems, pipes, and processes. These processes interact with kernel services and activate kernel-level memory events. UnixBench uses a shell command `time`, in order to aggregate timing performances for each of its benchmarks. Along with counting execution loops of each microbenchmark, the timings are used by UnixBench in order to derive performance index scores for each microbenchmark. The virtualisation layer will greatly influence how these resources are accessed and allocated from host to guest.
The UnixBench results show that OpenVZ appears twice as efficient as Xen at the UnixBench tests according to the composite results (Figure 16).

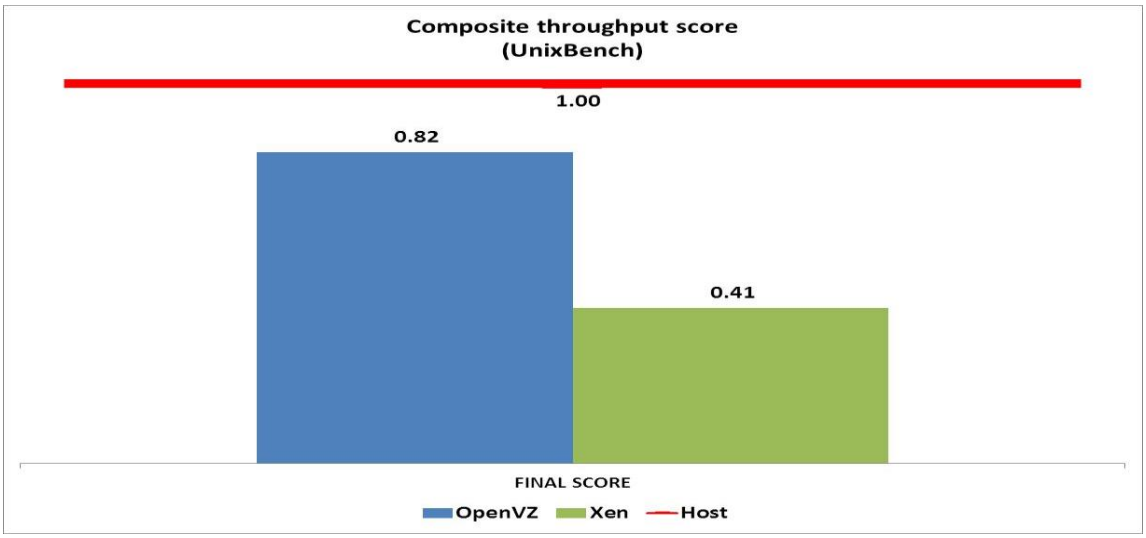

***Figure 16**: Composite throughput score*

4.4 Performance Summary

The guests perform relatively well compared to the physical host with a few exceptions as highlighted above. They also perform better or worse when compared to each other on differing benchmarks and no overall winner can be chosen based on all three sets of benchmarks.

This could be attributed to the approach to virtualisation adopted by each of these guests. Xen is a hypervisor-based virtualisation technology and OpenVZ is an OS-level virtualisation technology. Xen is designed to run multiple guest OSes, with a shared or isolated kernel whereas OpenVZ shares a single kernel among the guests.
Xen achieves guest isolation by separating guests into separate memory spaces and the guest can only access hardware resources through a set of hypervisor instructions.
OpenVZ guests are monitored using beancounters and isolation is achieved by accounting for the physical resources being used.
Xen therefore has the extra overhead of the hypervisor which manages the guests and resources.

It can therefore be inferred that Xen trades-off performance in favour of isolation, whereas OpenVZ trades-off isolation in favour of performance.

This research extended the work of (Barham et al 2003) and (Clark et al, 2004), by adding two new benchmarking suites as well as OpenVZ, a new form of virtualisation, not tested by (Barham et al 2003) and (Clark et al, 2004). The overlapping tests in the lmbench benchmark as used by (Barham et al 2003) and (Clark et al, 2004) produce relatively similar results, confirming their claims. The new benchmarks used in this research affirm the results of (Barham et al 2003) and (Clark et al, 2004) by showing that that the VMM does not impact throughput and the virtual environments perform at acceptable levels relative to the host.

Overall, based on these performance results, the performance of virtual environments are more than acceptable when compared to the performance of the physical host.

Chapter 5: Conclusion and Future Work

5.1 Conclusion

The objective of this research was to investigate the performance of virtualisation techniques, using benchmarking software and commodity hardware.

The precise objectives of the research were:

1. To install and setup open source paravirtualisation, using Xen and operating system virtualisation, using OpenVZ on commodity hardware.
2. To quantify the performance of each environment, physical and virtual, by using open source benchmarking software.

Objective 1 was achieved by installing Debian GNU/Linux on commodity hardware along with Xen and OpenVZ and their respective administration tools and utilities.

Objective 2 was achieved by installing benchmarking software on the host as well as Xen and OpenVZ guests. Xen and OpenVZ were benchmarked against the host and each other. A quantitative analysis was done to analyse the performance overheads incurred by using virtualisation.

The results show that near-native performance is possible using virtualisation, however, the type of application being hosted and the number of virtual environments running concurrently could impact performance and was beyond the scope of this research. There is no conclusive "clear winner" among the paravirtualised environment provided by Xen or the OS virtualised environment provided by OpenVZ and that could be attributed to the fact that OpenVZ and Xen are essentially two differing approaches to virtualisation and would be best suited to different use-cases. OpenVZ could be used to provision virtual environments where performance is more critical than isolation whereas Xen could be more suited to environments where isolation and resource guarantees are required.

This research and its findings serve as a starting point for systems researchers to develop and optimise virtualisation technologies to make them more suited for server consolidation.

5.2 Future Work

This research can be extended to include performance benchmarking of guests as more guest instances are provisioned on the same physical server. It can also be extended to quantify application benchmarking with the view to a better understanding of the real-world performance implications of hosting applications in virtual environments. Enterprise applications such as web and database servers can be tested under various resource configurations.

Cloud computing extends the field of virtualisation and opens the way to on-demand, high-performance computing. Possible areas of research in cloud computing include quantifying the cloud under load and measuring the efficacy of scheduling systems to efficiently allocate resources as virtual instances demand them.

References

Agarwal A., Desmarais R., Gable I., Norton A., Sobie R., Vanderster D.: 'Evaluation of Virtual Machines for HEP Grids', *Proceedings of Computing in High Energy Physics*, 2006

Andrzejak A., Arlitt M., Rolia J.: 'Bounding the Resource Savings of Utility Computing Models', Internet Systems and Storage Laboratory, *HP Laboratories Palo Alto, Technical Report HPL-2002-339*, 2002

Balsa AD. [online]: 'Linux Benchmarking HOWTO', v0.12, 1997, http://tldp.org/HOWTO/Benchmarking-HOWTO.html

Banga G., Druschel P.: 'Resource Containers: A New Facility for Resource Management in Server Systems', *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation USENIX*, 1999

Barham P., Dragovic B., Fraser K., Hand S., Harris T., Ho A., Neugebauer R., Pratt I., Warfield A.: 'Xen and the Art of Virtualization', *University of Cambridge Computer Laboratory*, 2003
http://www.cl.cam.ac.uk/research/srg/netos/papers/2003-xensosp.pdf

Bhatia S., Motiwala M., Muhlbauer W., Mundada Y., Valancius V., Bavier A., Feamster N., Peterson L., Rexford J.: 'Trellis: A Platform for Building Flexible, Fast Virtual Networks on Commodity Hardware', *Proceedings of ROADS'08*, 2008

byte-unixbench [online], http://code.google.com/p/byte-unixbench/

Carolan J., Radeztsky S., Strong P., Turner E.: 'Building N1 Grid Solutions - Preparing, Architecting, and Implementing Service-Centric Data Centers', *Prentice Hall and Sun Microsystems Press*, 2004
www.filibeto.org/sun/lib/blueprints/books/BP_Building_N1_Grid_Solutions.pdf

Chak H: ‘Virtualizing Asterisk’, ;*LOGIN:*, vol. 32, No. 1, pp64-66, 2007
http://www.usenix.org/publications/login/2007-02/index.html

Clark B., Deshane T., Dow E., Evanchik S., Finlayson M., Herne J., Matthews JN.: ‘Xen and the Art of Repeated Research’, *Proceedings of the Annual Technical Conference, FREENIX Track, USENIX*, pp135-144, 2004
http://www.clarkson.edu/class/cs644/xen/files/repeatedxen-usenix04.pdf

Dike J: ‘A user-mode port of the Linux kernel’, *Proceedings of the 4th Annual Linux Showcase & Conference USENIX*, 2000

Curnow HJ., Wichmann BA.: ‘A Synthetic Benchmark’, *Computer Journal*, Vol 19, No 1, pp43-49, 1976

Egi N., Greenhalgh A., Handley M., Hoerdt M., Huici F., Mathy L.: ‘Towards High Performance Virtual Routers on Commodity Hardware’, *Proceedings of ACM CoNEXT*, 2008

Fraser K., Hand S., Neugebauer R., Pratt I., Warfield A., Wiiliamson M.: ‘Safe Hardware Access with the Xen Virtual Machine Monitor’, *University of Cambridge Computer Laboratory* 2004
http://www.cl.cam.ac.uk/research/srg/netos/papers/2004-oasis-ngio.pdf

Hatt N., Sivitz A., and Kuperman BA.: ‘Benchmarking Operating Systems’ *Midstates Conference for Undergraduate Research in Computer Science and Mathematics*, pp63-68, November 2007

Helvick S.: ‘A Survey of Hardware Performance Analysis Tools’, *Project Report*, http://www.cse.wustl.edu/~jain/cse567-08/index.html, 2008

Höxer H., Buchacker K., Sieh V.: ‘Implementing a User Mode Linux with Minimal Changes from Original Kernel’, *9th International Linux System Technology Conference*, pp72-82, 2002

Kamp P., Watson R.: 'Jails: Confining the omnipotent root', The FreeBSD Project, http://ivanlef0u.fr/repo/madchat/sysadm/bsd/kamp.pdf, 2000

King ST., Dunlap GW., Chen PM.: 'Operating System Support for Virtual Machines' *Annual Technical Conference on USENIX*, pp71-84 of the Proceedings, 2003

Makhija V., Herndon B., Smith P., Roderick L., Zamost E., Anderson J.: 'Vmmark: A Scalable Benchmark for Virtualized Systems.' *Technical Report, VMware, Inc*., 2006 (http://www.vmware.com/pdf/vmmark_intro.pdf)

McVoy L., Staelin C.: 'lmbench: Portable tools for Performance Analysis' Silicon Graphics, Inc. Hewlett-Packard Laboratories, *Proceedings of the Annual Technical Conference on USENIX*, 1996

OpenVZ [online], http://wiki.openvz.org/Introduction_to_virtualization

Padala P., Zhu X., Wang Z., Singhal S., Shin KG.: 'Performance Evaluation of Virtualization Technologies for Server Consolidation', *HP Laboratories, HPL-2007-59R1*, 2008 (http://www.hpl.hp.com/techreports/2007/HPL-2007-59R1.pdf)

Robin JS., Irvine CE.: 'Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor', *Proceedings of the 9th USENIX Security Symposium*, 2000

Sliwa, C., Vijayan, J.: 'IT Managers Tackle Windows Server Sprawl', Computerworld, 2002 (http://www.computerworld.com/softwaretopics/os/story/0,10801,75272,00.html)

Staelin C.: 'lmbench3: Measuring Scalability', *HP Laboratories Israel, HPL-2002-313*, 2002 (http://www.hpl.hp.com/techreports/2002/HPL-2002-313.pdf)

Staelin C.: 'lmbench – an extensible micro-benchmark suite', *Software – Practice and Experience*, vol.0 no.0, pp1-7, 2004

Sugerman J., Venkitachalam G., Lim B.: 'Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor', *Proceedings of the 2001 USENIX Annual Technical Conference*, 2001

Sukaridhotoy S, Funabikiy N, Nakanishiy T, and Pramadihanto D: 'A Comparative Study of Open Source Softwares for Virtualization with Streaming Server Applications' *The 13th IEEE International Symposium on Consumer Electronics*, pp577-581, 2009

Symantec Enterprise Security Solutions: 'Securing Virtual Environments with Symantec Endpoint Protection', *White Paper*, 2010

Tate J., Lucchese F., Moore R.: 'Introduction to Storage Area Networks', *IBM RedBooks*, 2003, http://www.redbooks.ibm.com/redbooks/pdfs/sg245470.pdf

VMware, 'A Performance Comparison of Hypervisors', *VMware Technical Papers*, 2007, (http://www.vmware.com/resources/techresources/711)

Wang Y, Biskeborn B, van der Merwe J, Rexford J: 'Virtual Routers on the Move: Live Router Migration as a Network-Management Primitive', *Proceedings of SIGCOMM'08*, 2008

Williams TL.: 'A General-Purpose Model for Heterogeneous Computation' *Doctoral Dissertation, School of Electrical Engineering and Computer Science, College of Engineering and Computer Science, University of Central Florida*, 2000 (http://portal.acm.org/citation.cfm?id=932536&coll=DL&dl=GUIDE&CFID=9894978&CFTOKEN=75295008)

Appendix

Outliers are highlighted in *italics*

μ = Arithmetic Average of Test 1 to 5

$$= \frac{1}{n}\sum_{i=1}^{n} x_i$$

σ = Standard Deviation of Test 1 to 5

$$= \sqrt{\frac{\sum (x-\mu)^2}{(n-1)}}$$

A.1: lmbench Summary Results: Host

Processor, Processes in microseconds (smaller is better)							
Physical Host	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
null call	0.08	0.08	0.13	0.08	0.08	**0.09**	0.02
null I/O	0.20	0.18	0.27	0.15	0.26	**0.21**	0.05
Stat	1.41	1.35	1.39	1.39	1.49	**1.41**	0.05
open clos	2.30	2.21	2.13	2.31	2.26	**2.24**	0.07
slct TCP	7.46	19.10	13.10	15.00	7.45	**12.42**	5.03
sig inst	0.33	0.33	0.43	0.34	0.34	**0.35**	0.04
sig hndl	1.52	1.28	1.53	1.61	1.23	**1.43**	0.17
fork proc	105.00	97.50	95.70	106.00	103.00	**101.44**	4.59
exec proc	347.00	350.00	338.00	367.00	345.00	**349.40**	10.78
sh proc	2190.0	2169.0	2166.0	2233.0	2183.0	**2188.2**	26.92

Table 6: *Physical Host CPU & Processes latency*

Context Switching in microseconds (smaller is better)							
Physical Host	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
2p/0K ctxsw	1.40	1.17	1.11	*0.80*	1.22	**1.14**	0.219
2p/16K ctxsw	1.06	0.94	1.07	0.87	*0.69*	**0.93**	0.156
2p/64K ctxsw	5.71	5.52	5.58	5.69	5.68	**5.64**	0.082
8p/16K ctxsw	2.47	2.37	2.68	2.19	2.36	**2.41**	0.180
8p/64K ctxsw	39.00	34.10	*44.60*	36.40	36.10	**38.04**	4.080
16p/16K ctxsw	2.64	2.85	2.98	2.60	*3.48*	**2.91**	0.354
16p/64K ctxsw	63.70	63.60	63.70	63.20	63.70	**63.58**	0.217

Table 7: *Physical Host Context Switching latency*

Local Communication Latencies in microseconds (smaller is better)							
Physical Host	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
2p/0K ctxsw	1.40	1.17	1.11	*0.80*	1.22	**1.14**	0.219
Pipe	4.48	4.39	4.76	4.55	4.52	**4.54**	0.138
AF UNIX	8.26	5.44	6.01	5.99	6.59	**6.46**	1.086
UDP	6.86	7.70	8.78	7.83	7.39	**7.71**	0.705
TCP	12.70	13.90	13.30	12.10	14.50	**13.30**	0.949
TCP conn	42.00	44.00	42.00	40.00	43.00	**42.20**	1.483

Table 8: *Physical Host Summary Local Communication latency*

File & VM System Latencies in microseconds (smaller is better)							
Physical Host	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
0K File Create	14.10	14.20	14.40	14.80	13.20	**14.14**	0.590
0K File Delete	11.50	12.30	11.40	13.40	11.00	**11.92**	0.952
10K File Create	61.90	63.50	63.50	68.10	65.80	**64.56**	2.418
10K File Delete	24.50	25.70	25.50	27.30	25.10	**25.62**	1.045
Mmap Latency	4248	4363	4148	*3882*	4148	**4158**	177.84
Prot Fault	*1.59*	*1.66*	*0.06*	*0.11*	*0.41*	**0.77**	0.798
Page Fault	2.70	2.63	1.45	1.57	1.58	**1.99**	0.625
100fd selct	N/A	N/A	2.57	2.60	2.72	**2.63**	0.079

Table 9: *Physical Host File & VM System latency*

Local Communication Bandwidths in MB/s (bigger is better)							
Physical Host	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
Pipe	1122.0	1123.0	1122.0	1129.0	1124.0	**1124.0**	2.915
AF Unix	1225.0	1256.0	1217.0	1249.0	1249.0	**1239.2**	17.094
TCP	212.0	214.0	214.0	218.0	209.0	**214.0**	3.674
File reread	553.1	551.3	550.5	550.2	550.7	**551.2**	1.157
Mmap reread	1040.6	1040.6	1040.9	1040.7	1040.7	**1040.7**	0.122
Bcopy (libc)	369.3	376.6	379.0	380.9	377.6	**376.7**	4.430
Bcopy (hand)	371.7	379.4	379.8	379.3	378.3	**377.7**	3.399
Mem read	1015.0	1015.0	1015.0	1014.0	1015.0	**1014.8**	0.447
Mem write	625.6	635.8	638.8	642.3	647.2	**637.9**	8.099

Table 10: *Physical Host Local Communication Bandwidth throughput*

A.2: lmbench Summary Results: OpenVZ

Processor, Processes in microseconds (smaller is better)							
OpenVZ	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
null call	0.13	0.13	0.13	0.13	0.13	**0.13**	0.00
null I/O	0.28	0.28	0.31	0.23	0.26	**0.27**	0.03
Stat	1.51	1.56	1.67	1.53	1.53	**1.56**	0.06
open clos	2.61	2.68	2.74	2.70	2.68	**2.68**	0.05
slct TCP	8.17	8.15	8.16	12.90	16.80	**10.84**	3.92
sig inst	0.45	0.45	0.45	0.45	0.45	**0.45**	0.00
sig hndl	1.51	1.65	1.48	1.46	1.53	**1.53**	0.07
fork proc	139.00	142.00	143.00	146.00	156.00	**145.20**	6.53
exec proc	445.00	450.00	468.00	491.00	510.00	**472.80**	27.53
sh proc	2651.0	2664.0	2644.0	2712.0	2717.0	**2677.6**	34.49

Table 11: *OpenVZ CPU & Processes latency*

Context Switching in microseconds (smaller is better)							
OpenVZ	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
2p/0K ctxsw	0.59	1.14	0.74	0.88	0.93	**0.86**	0.207
2p/16K ctxsw	1.37	1.51	1.02	1.44	1.26	**1.32**	0.191
2p/64K ctxsw	5.92	5.66	6.19	5.53	5.77	**5.81**	0.254
8p/16K ctxsw	2.75	2.74	2.65	2.56	2.54	**2.65**	0.100
8p/64K ctxsw	42.3	37.0	46.2	37.2	39.6	**40.46**	3.862
16p/16K ctxsw	2.93	2.81	2.79	*4.80*	3.70	**3.41**	0.865
16p/64K ctxsw	65.00	64.80	65.20	65.00	65.10	**65.02**	0.148

Table 12: *OpenVZ Context Switching latency*

Local Communication Latencies in microseconds (smaller is better)							
OpenVZ	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
2p/0K ctxsw	0.59	1.14	0.74	0.88	0.93	**0.86**	0.207
Pipe	4.34	4.99	5.26	4.80	6.10	**5.10**	0.652
AF UNIX	7.84	6.97	7.8	7.94	7.42	**7.59**	0.401
UDP	14.50	12.50	13.10	12.50	12.90	**13.10**	0.825
TCP	19.70	19.50	16.90	20.90	19.40	**19.28**	1.460
TCP conn	61.00	67.00	69.00	*30.30*	64.00	**58.26**	15.921

Table 13: *OpenVZ Local Communication latency*

File & VM System Latencies in microseconds (smaller is better)							
OpenVZ	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
0K File Create	17.70	16.60	15.50	17.10	16.40	**16.60**	0.82
0K File Delete	13.50	15.70	14.50	15.60	15.70	**15.00**	0.98
10K File Create	73.90	76.00	74.00	72.40	74.20	**74.10**	1.28
10K File Delete	29.00	33.00	30.70	32.80	33.10	**31.70**	1.81
Mmap Latency	9829	10900	11000	10900	11700	**10866**	669.18
Prot Fault	0.23	*3.97*	0.21	0.21	*0.43*	**1.01**	1.66
Page Fault	3.84	2.70	3.98	3.94	4.00	**3.69**	0.56
100fd selct	2.73	N/A	2.72	2.68	2.80	**2.73**	0.05

Table 14: *OpenVZ File & VM System latency*

Local Communication Bandwidths in MB/s (bigger is better)							
OpenVZ	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
Pipe	1049	1044	1065	1020	1042	**1044**	16.171
AF Unix	1170	1200	1182	1169	1178	**1179.8**	12.538
TCP	193	191	202	191	191	**193.60**	4.775
File reread	543.80	540.30	541.50	540.00	540.40	**541.20**	1.560
Mmap reread	1025.5	1025.4	1024.9	1025.3	1024.9	**1025.2**	0.283
Bcopy (libc)	347.40	368.60	362.90	367.00	369.40	**363.06**	9.106
Bcopy (hand)	330.30	357.20	364.20	366.10	368.40	**357.24**	15.631
Mem read	1000	1000	1000	1000	1000	**1000**	0.000
Mem write	598.70	605.70	612.70	615.80	618.10	**610.20**	7.945

Table 15: *OpenVZ Local Communication Bandwidth throughput*

A.3: lmbench Summary Results: Xen

Processor, Processes in microseconds (smaller is better)							
Xen	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
null call	0.15	0.15	0.15	0.15	0.16	**0.15**	0.00
null I/O	0.24	0.24	0.24	0.24	0.24	**0.24**	0.00
Stat	1.40	1.38	1.36	1.38	1.38	**1.38**	0.01
open clos	2.33	2.36	2.35	2.38	2.38	**2.36**	0.02
slct TCP	6.92	6.72	6.72	6.73	6.73	**6.76**	0.09
sig inst	0.43	0.43	0.43	0.42	0.42	**0.43**	0.01
sig hndl	1.34	1.34	1.32	1.31	1.31	**1.32**	0.02
fork proc	414.00	415.00	427.00	429.00	418.00	**420.60**	6.95
exec proc	1097.0	1120.0	1077.0	1098.0	1112.0	**1100.8**	16.45
sh proc	3755.0	3783.0	3776.0	3805.0	3820.0	**3787.8**	25.35

Table 16: *Xen CPU & Processes latency*

Context Switching in microseconds (smaller is better)							
Xen	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
2p/0K ctxsw	4.24	4.17	4.05	4.18	4.19	**4.166**	0.070
2p/16K ctxsw	4.32	4.50	4.14	4.32	4.68	**4.39**	0.205
2p/64K ctxsw	8.65	8.85	8.46	8.82	8.78	**8.71**	0.160
8p/16K ctxsw	6.27	6.09	6.49	6.25	6.98	**6.42**	0.346
8p/64K ctxsw	54.50	47.60	56.00	56.50	54.10	**53.74**	3.575
16p/16K ctxsw	7.68	8.06	8.55	8.54	7.89	**8.14**	0.390
16p/64K ctxsw	71.30	71.50	72.60	72.20	72.70	**72.06**	0.635

Table 17: *Xen Context Switching latency*

Local Communication Latencies in microseconds (smaller is better)							
Xen	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
2p/0K ctxsw	4.24	4.17	4.05	4.18	4.19	**4.16**	0.070
Pipe	21.80	22.00	21.60	22.40	21.70	**21.90**	0.316
AF UNIX	15.50	15.80	15.30	15.70	15.60	**15.58**	0.192
UDP	15.20	15.20	15.40	15.50	15.50	**15.36**	0.152
TCP	20.30	20.30	20.20	20.30	20.20	**20.26**	0.055
TCP conn	63	63	63	64	63	**63.20**	0.447

Table 18: *Xen Local Communication latency*

File & VM System Latencies in microseconds (smaller is better)							
Xen	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
0K File Create	9.90	9.89	9.85	10.2	9.94	**9.96**	0.139
0K File Delete	8.53	7.98	7.97	8.22	8.25	**8.19**	0.231
10K File Create	54.20	56.00	65.40	54.90	53.30	**56.76**	4.930
10K File Delete	17.60	16.90	17.10	16.90	17.30	**17.16**	0.297
Mmap Latency	15200	15900	15500	15800	15800	**15640**	288.09
Prot Fault	0.44	0.42	0.45	0.47	0.40	**0.44**	0.027
Page Fault	4.81	4.92	4.90	4.96	4.99	**4.92**	0.069
100fd selct	2.97	2.74	2.77	2.77	2.76	**2.80**	0.093

Table 19: *Xen File & VM System latency*

Local Communication Bandwidths in MB/s (bigger is better)							
Xen	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
Pipe	*1016*	*1007*	426	426	438	**662.6**	318.55
AF Unix	1155	1153	1153	1154	1154	**1153.8**	0.837
TCP	198	199	199	198	201	**199**	1.225
File reread	516.50	512.50	512.20	508.90	505.70	**511.16**	4.072
Mmap reread	1028.8	1028.4	1027.8	1027.7	1027.3	**1028.0**	0.596
Bcopy (libc)	359.10	367.50	367.20	368.90	370.90	**366.72**	4.504
Bcopy (hand)	361.60	369.60	369.20	370.20	370.00	**368.12**	3.665
Mem read	1003	1002	999	1002	1002	**1001.6**	1.517
Mem write	607.10	617.80	621.00	623.30	625.10	**618.86**	7.118

Table 20: *Xen Local Communication Bandwidth throughput*

B.1: nbench/BYTEMark* Summary Results: Host

Physical Host	Test1	Test2	Test3	Test4	Test5	μ	σ
Numeric Sort	24.75	24.74	24.79	24.76	24.74	**24.76**	0.02
String Sort	73.72	73.57	73.85	73.94	73.69	**73.75**	0.14
Bitfield	80.25	83.91	83.53	79.74	82.19	**82.19**	2.02
Floating Point Emulation	75.97	75.86	75.94	75.90	75.89	**75.91**	0.04
Fourier Coefficients	24.22	24.30	24.31	24.25	24.19	**24.25**	0.05
Assignment	90.09	90.24	89.66	89.69	90.06	**89.95**	0.26
IDEA Encryption	55.63	55.54	55.63	55.61	55.54	**55.59**	0.05
Huffman Compression	44.37	44.33	44.37	44.31	44.28	**44.33**	0.04
Neural Net	51.22	51.11	50.47	50.58	51.18	**50.91**	0.36
LU Decomposition	58.11	57.68	57.89	56.88	58.14	**57.74**	0.52
Memory Index	16.47	16.72	16.68	16.69	16.43	**16.60**	0.13
Integer Index	13.51	13.50	13.52	13.51	13.50	**13.51**	0.01
Floating-Point Index	23.08	23.03	22.97	22.83	23.07	**23.00**	0.10

Table 21: *Physical Host BYTEMark throughput*

B.2: nbench/BYTEMark* Summary Results: OpenVZ

OpenVZ	Test1	Test2	Test3	Test4	Test5	μ	σ
Numeric Sort	24.91	25.04	24.76	24.64	25.33	**24.94**	0.27
String Sort	74.13	72.29	74.58	73.51	72.85	**73.47**	0.93
Bitfield	80.16	82.14	81.41	82.20	78.98	**80.98**	1.39
Floating Point Emulation	75.76	75.76	75.68	75.78	75.76	**75.75**	0.04
Fourier Coefficients	24.13	24.13	24.22	24.13	24.13	**24.15**	0.04
Assignment	89.29	89.41	89.34	89.90	89.67	**89.52**	0.26
IDEA Encryption	55.41	55.46	55.47	55.48	55.40	**55.44**	0.04
Huffman Compression	44.23	44.23	44.21	44.12	44.12	**44.18**	0.06
Neural Net	48.78	48.76	48.78	48.80	48.80	**48.78**	0.02
LU Decomposition	53.77	57.34	54.36	57.37	57.64	**56.10**	1.87
Memory Index	16.45	16.45	16.57	16.58	16.30	**16.47**	0.11
Integer Index	13.50	13.52	13.48	13.46	13.55	**13.50**	0.03
Floating-Point Index	22.10	22.58	22.21	22.59	22.63	**22.42**	0.24

Table 22: *OpenVZ BYTEMark throughput*

B.3: nbench/BYTEMark* Summary Results: Xen

Xen	Test1	Test2	Test3	Test4	Test5	μ	σ
Numeric Sort	24.79	24.92	24.79	24.86	24.69	**24.81**	0.09
String Sort	73.41	73.57	73.85	73.61	73.57	**73.60**	0.16
Bitfield	82.74	82.90	82.80	83.12	82.96	**82.90**	0.15
Floating Point Emulation	75.83	76.01	75.89	75.89	75.90	**75.90**	0.07
Fourier Coefficients	24.29	24.22	24.25	24.22	24.23	**24.24**	0.03
Assignment	89.72	89.66	90.05	89.85	90.10	**89.88**	0.20
IDEA Encryption	55.74	55.61	55.61	55.63	55.61	**55.64**	0.06
Huffman Compression	44.37	44.30	44.41	44.28	44.28	**44.33**	0.06
Neural Net	50.45	50.36	50.25	50.47	50.32	**50.37**	0.09
LU Decomposition	57.67	57.69	58.16	52.13	58.08	**56.75**	2.59
Memory Index	16.59	16.61	16.65	16.64	16.64	**16.63**	0.02
Integer Index	13.52	13.53	13.52	13.52	13.49	**13.52**	0.01
Floating-Point Index	22.93	22.90	22.95	22.15	22.94	**22.78**	0.35

Table 23: *Xen BYTEMark throughput*

C.1: UnixBench Summary Results: Host

CPU Benchmarks							
Physical Host	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
Arithmetic Test (type = double)	290	290	290	290	290	**290**	0.05
Dhrystone 2 Using register variables	469	471	470	471	471	**470**	0.80
Execl Throughput	*2,201*	1,082	1,085	1,059	1,064	**1,298**	504.75

Table 24: *Physical Host CPU throughput*

Inter-Process Communication Benchmarks							
Physical Host	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
Pipe Throughput	940	812	734	855	780	**824**	78.73
Process Creation	1,316	1,269	1,218	1,230	1,267	**1,260**	38.44
Shell Scripts (8 concurrent)	787	791	781	787	785	**786**	3.88
System Call Overhead	1,520	1,523	1,295	1,518	1,522	**1,476**	101.09

Table 25: *Physical Host IPC throughput*

Filesystem Benchmarks							
Physical Host	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
File Copy 256 bufsize 500 maxblocks	614	583	535	529	570	**566**	35.29
File Copy 1024 bufsize 2000 maxblocks	460	448	445	444	454	**450**	6.88
File Copy 4096 bufsize 8000 maxblocks	400	392	399	400	401	**398**	3.76

Table 26: *Physical Host Filesystem throughput*

C.2: UnixBench Summary Results: OpenVZ

CPU Benchmarks							
OpenVZ	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
Arithmetic Test (type = double)	289	289	288	289	289	**289**	0.17
Dhrystone 2 Using register variables	413	412	413	413	413	**413**	0.47
Execl Throughput	858	838	838	837	836	**841**	9.50

Table 27: *OpenVZ CPU throughput*

Inter-Process Communication Benchmarks							
OpenVZ	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
Pipe Throughput	559	575	616	604	604	**592**	23.90
Process Creation	914	857	874	842	861	**870**	27.18
Shell Scripts (8 concurrent)	719	718	722	715	715	**718**	2.94
System Call Overhead	952	951	951	949	952	**951**	1.10

Table 28: *OpenVZ IPC throughput*

Filesystem Benchmarks							
OpenVZ	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
File Copy 256 bufsize 500 maxblocks	463	467	435	452	452	**454**	12.62
File Copy 1024 bufsize 2000 maxblocks	423	415	413	408	411	**414**	5.74
File Copy 4096 bufsize 8000 maxblocks	383	385	376	369	375	**378**	6.26

Table 29: *OpenVZ Filesystem throughput*

C.3: UnixBench Summary Results: Xen

CPU Benchmarks							
Xen	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
Arithmetic Test (type = double)	290	290	290	290	289	**290**	0.22
Dhrystone 2 Using register variables	472	472	471	471	470	**471**	0.69
Execl Throughput	384	387	390	392	394	**389**	3.91

Table 30: *Xen CPU throughput*

Inter-Process Communication Benchmarks							
Xen	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
Pipe Throughput	118	118	119	119	120	**119**	0.76
Process Creation	226	227	230	230	229	**228**	1.90
Shell Scripts (8 concurrent)	487	486	485	483	483	**485**	1.47
System Call Overhead	876	876	876	875	871	**875**	2.40

Table 31: *Xen IPC throughput*

Filesystem Benchmarks							
Xen	**Test1**	**Test2**	**Test3**	**Test4**	**Test5**	**μ**	**σ**
File Copy 256 bufsize 500 maxblocks	169	172	170	169	171	**170**	1.35
File Copy 1024 bufsize 2000 maxblocks	213	211	212	211	211	**212**	0.63
File Copy 4096 bufsize 8000 maxblocks	288	289	288	289	287	**288**	0.83

Table 32: *Xen Filesystem throughput*

D.1: OpenVZ global configuration file

```
## Global parameters
VIRTUOZZO=yes
LOCKDIR=/var/lib/vz/lock
DUMPDIR=/var/lib/vz/dump
VE0CPUUNITS=1000

## Logging parameters
LOGGING=yes
LOGFILE=/var/log/vzctl.log
LOG_LEVEL=0
VERBOSE=0

## Disk quota parameters
DISK_QUOTA=yes
VZFASTBOOT=no

# The name of the device whose ip address will be used as
source ip for VE.
# By default automatically assigned.
#VE_ROUTE_SRC_DEV="eth0"

# Controls which interfaces to send ARP requests and
modify APR tables on.
NEIGHBOUR_DEVS=detect

## Template parameters
TEMPLATE=/var/lib/vz/template

## Defaults for VEs
VE_ROOT=/var/lib/vz/root/$VEID
VE_PRIVATE=/var/lib/vz/private/$VEID
CONFIGFILE="vps.solo"
```

```
DEF_OSTEMPLATE="debian-4.0-i386-minimal"

## Load vzwdog module
VZWDOG="no"

## IPv4 iptables kernel modules
IPTABLES="ipt_REJECT ipt_tos ipt_limit ipt_multiport
iptable_filter iptable_mangle ipt_TCPMSS ipt_tcpmss
ipt_ttl ipt_length"

## Enable IPv6
IPV6="no"

## IPv6 ip6tables kernel modules
IP6TABLES="ip6_tables ip6table_filter ip6table_mangle
ip6t_REJECT"
```

D.2: OpenVZ guest configuration file

```
# Configuration file generated by vzsplit for 1 VEs
# on HN with total amount of physical mem 979 Mb
# low memory 883 Mb, swap size 839 Mb, Max treads 8000
# Resourse commit level 0:
# Free resource distribution. Any parameters may be
increased
# Primary parameters
NUMPROC="8000:8000"
AVNUMPROC="2262:2262"
NUMTCPSOCK="8000:8000"
NUMOTHERSOCK="8000:8000"
#VMGUARPAGES="150470:2147483647"
VMGUARPAGES="454656:2147483647"

# Secondary parameters
KMEMSIZE="185323520:203855872"
TCPSNDBUF="29006506:61774506"
TCPRCVBUF="29006506:61774506"
OTHERSOCKBUF="14503253:47271253"
DGRAMRCVBUF="14503253:14503253"
OOMGUARPAGES="150470:2147483647"
#PRIVVMPAGES="150470:165517"
PRIVVMPAGES="229376:252317"

# Auxiliary parameters
LOCKEDPAGES="9049:9049"
SHMPAGES="15047:15047"
PHYSPAGES="0:2147483647"
NUMFILE="72384:72384"
NUMFLOCK="1000:1100"
NUMPTY="512:512"
NUMSIGINFO="1024:1024"
```

```
DCACHESIZE="40478819:41693184"
NUMIPTENT="200:200"
DISKSPACE="4181364:4599504"
DISKINODES="1017425:1119168"
CPUUNITS="54150"
VE_ROOT="/var/lib/vz/root/$VEID"
VE_PRIVATE="/var/lib/vz/private/$VEID"
OSTEMPLATE="debian-4.0-i386-minimal"
ORIGIN_SAMPLE="vps.solo"
IP_ADDRESS="192.168.2.4"
HOSTNAME="ovz.domain.name"
NAMESERVER="192.168.2.2"
ONBOOT="yes"
```

E.1: Xen Xend configuration file

```
# -*- sh -*-

#
# Xend configuration file.
#

# This example configuration is appropriate for an
installation that
# utilizes a bridged network configuration. Access to
xend via http
# is disabled.

# Commented out entries show the default for that entry,
unless otherwise
# specified.

#(logfile /var/log/xen/xend.log)
#(loglevel DEBUG)

#(xend-http-server no)
#(xend-unix-server no)
#(xend-tcp-xmlrpc-server no)
#(xend-unix-xmlrpc-server yes)
#(xend-relocation-server no)

#(xend-unix-path /var/lib/xend/xend-socket)

# Port xend should use for the HTTP interface, if xend-
http-server is set.
#(xend-port            8000)
```

```
# Port xend should use for the relocation interface, if
xend-relocation-server
# is set.
#(xend-relocation-port 8002)

# Address xend should listen on for HTTP connections, if
xend-http-server is
# set.
# Specifying 'localhost' prevents remote connections.
# Specifying the empty string '' (the default) allows all
connections.
#(xend-address '')
#(xend-address localhost)

# Address xend should listen on for relocation-socket
connections, if
# xend-relocation-server is set.
# Meaning and default as for xend-address above.
#(xend-relocation-address '')

# The hosts allowed to talk to the relocation port.  If
this is empty (the
# default), then all connections are allowed (assuming
that the connection
# arrives on a port and interface on which we are
listening; see
# xend-relocation-port and xend-relocation-address
above).  Otherwise, this
# should be a space-separated sequence of regular
expressions.  Any host with
# a fully-qualified domain name or an IP address that
matches one of these
# regular expressions will be accepted.
#
```

```
# For example:
#  (xend-relocation-hosts-allow '^localhost$
^.*\.example\.org$')
#
#(xend-relocation-hosts-allow '')

# The limit (in kilobytes) on the size of the console
buffer
#(console-limit 1024)

##
# To bridge network traffic, like this:
#
# dom0: fake eth0 -> vif0.0 -+
#                            |
#                          bridge -> real eth0 -> the
network
#                            |
# domU: fake eth0 -> vifN.0 -+
#
# use
#
# (network-script network-bridge)
#
# Your default ethernet device is used as the outgoing
interface, by default.
# To use a different one (e.g. eth1) use
#
# (network-script 'network-bridge netdev=eth1')
#
# The bridge is named xenbr0, by default.  To rename the
bridge, use
#
# (network-script 'network-bridge bridge=<name>')
```

```
#
# It is possible to use the network-bridge script in more
complicated
# scenarios, such as having two outgoing interfaces, with
two bridges, and
# two fake interfaces per guest domain.  To do things
like this, write
# yourself a wrapper script, and call network-bridge from
it, as appropriate.
#
(network-script network-bridge)

# The script used to control virtual interfaces.  This
can be overridden on a
# per-vif basis when creating a domain or a configuring a
new vif.  The
# vif-bridge script is designed for use with the network-
bridge script, or
# similar configurations.
#
# If you have overridden the bridge name using
# (network-script 'network-bridge bridge=<name>') then
you may wish to do the
# same here.  The bridge name can also be set when
creating a domain or
# configuring a new vif, but a value specified here would
act as a default.
#
# If you are using only one bridge, the vif-bridge script
will discover that,
# so there is no need to specify it explicitly.
#
(vif-script vif-bridge)
```

```
## Use the following if network traffic is routed, as an
alternative to the
# settings for bridged networking given above.
#(network-script network-route)
#(vif-script     vif-route)


## Use the following if network traffic is routed with
NAT, as an alternative
# to the settings for bridged networking given above.
#(network-script network-nat)
#(vif-script     vif-nat)


# Dom0 will balloon out when needed to free memory for
domU.
# dom0-min-mem is the lowest memory level (in MB) dom0
will get down to.
# If dom0-min-mem=0, dom0 will never balloon out.
(dom0-min-mem 196)

# In SMP system, dom0 will use dom0-cpus # of CPUS
# If dom0-cpus = 0, dom0 will take all cpus available
(dom0-cpus 0)

# Whether to enable core-dumps when domains crash.
#(enable-dump no)

# The tool used for initiating virtual TPM migration
#(external-migration-tool '')

# The interface for VNC servers to listen on. Defaults
```

```
# to 127.0.0.1  To restore old 'listen everywhere'
behaviour
# set this to 0.0.0.0
#(vnc-listen '127.0.0.1')
```

E.2: Xen guest configuration file

```
#  Configuration file for the Xen instance xen, created
#  on Fri Oct 10 01:01:27 2008.

#  Kernel + memory size
#
kernel  = '/boot/vmlinuz-xen'
ramdisk = '/boot/initrd.img-xen'

memory  = '896'

#  Disk device(s).
#
root    = '/dev/sda1 ro'

disk    = [ 'file:/xen/domains/xen/disk.img,sda1,w',
'file:/xen/domains/xen/swap.img,sda2,w' ]

#  Hostname
#
name    = 'xen'

#  Networking
#
vif  = [ 'ip=192.168.2.6' ]

#  Behaviour
#
on_poweroff = 'destroy'
on_reboot   = 'restart'
on_crash    = 'restart'
```

F: Xen-tools configuration file

```
#
# /etc/xen-tools/xen-tools.conf
#
#  Global configuration file for the scripts included
with Xen-tools.
#
#  Values may be set here so they don't need to be
specified upon the
# command line.
#
# Steve
# --
#




#
##
#  Output directory for storing loopback images.
#
#  If you choose to use loopback images, which are simple
to manage but
# slower than LVM partitions, then specify a directory
here and uncomment
# the line.
#
#  New instances will be stored in subdirectories named
after their
# hostnames.
#
##
dir = /xen
```

```
#


#
##
#
# If you don't wish to use loopback images then you may
specify an
# LVM volume group here instead
#
##
# lvm = skx-vg



#
##
#
#  Installation method.
#
#  There are four different methods you can use to
install a new copy
# of Linux to use in your Xen guest domain:
#
#   - Installation via the debootstrap command.
#   - Installation via the rpmstrap command.
#   - Installation by copying a directory containing a
previous installation.
#   - Installation by untarring a previously archived
image.
#
#  NOTE That if you use the "untar", or "copy" options
you should ensure
# that the image you're left with matches the 'dist'
setting later in
# this file.
```

```
#
#  Note that you can only uncomment one method - they are
mutually exclusive.
# However the command line installation method will allow
you to override
# the choice you make here.
#
##
#
# copy = /path/to/pristine/image
debootstrap = 1
# rpmstrap = 1
# tar = /path/to/img.tar
#

#
##
#  Command definitions.
##
#
# The "debootstrap" and "rpmstrap" commands are
hardwired, but if you
# wish to alter the commands invoked when using the "--
copy" + "--tar"
# options you can adjust these two settings:
#
# --copy:
# copy-cmd = /bin/cp -a $src/* $dest
#
# --tar:
# tar-cmd  = /bin/tar --numeric-owner -xvf $src
#
#
```

```
#
##
#  Disk and Sizing options.
##
#
size   = 4Gb      # Disk image size.
memory = 896Mb    # Memory size
swap   = 880Mb    # Swap size
# noswap = 1      # Don't use swap at all for the new
system.
fs     = ext3     # use the EXT3 filesystem for the disk
image.
dist   = etch     # Default distribution to install.
image  = sparse   # Specify sparse vs. full disk images.

#
#  Currently supported and tested distributions include:
#
#   sid           - Debian
#   sarge         - Debian
#   etch          - Debian
#   dapper        - Ubuntu
#   centos4       - CentOS 4
#   fedora-core4 - Fedora Core 4 (codname stentz)
#



##
# Networking setup values.
##
```

```
#
# Uncomment and adjust these network settings if you wish
to give your
# new instances static IP addresses.
#
gateway   = 192.168.2.1
netmask   = 255.255.255.0
#
# Uncomment this if you wish the images to use DHCP
#
# dhcp = 1


##
# Misc options
##

#
# Uncomment the following line if you wish to disable the
caching
# of downloaded .deb files when using debootstrap to
install images.
#
# cache = no
#

#
# Uncomment the following line if you wish to
interactively setup
# a new root password for images.
#
passwd = 1

#
```

```
# If you'd like all accounts on your host system which
are not present
# on the guest system to be copied over then uncomment
the following line.
#
# accounts = 1
#

#
# Default kernel and ramdisk to use for the virtual
servers
#
kernel = /boot/vmlinuz-xen
initrd = /boot/initrd.img-xen

#
#  The architecture to use when using debootstrap or
rpmstrap.
#
#  This is most useful on 64 bit host machines, for other
systems it
# doesn't need to be used.
#
# arch=i386
#

#
# The default mirror for debootstrap which can be used to
install
# Debian Sid, Sarge, and Etch.
#
mirror = http://debian.mirror.ac.za/debian/

#
```

```
# A mirror suitable for use when installing the Dapper
release of Ubuntu.
#
# mirror = http://gb.archive.ubuntu.com/ubuntu/

#
#  Uncomment if you wish newly created images to boot
once they've been
# created.
#
boot = 1
```