University of Glasgow | School of Computing Science

Virtualisation

Advanced Operating Systems
Lecture 17

Lecture Outline

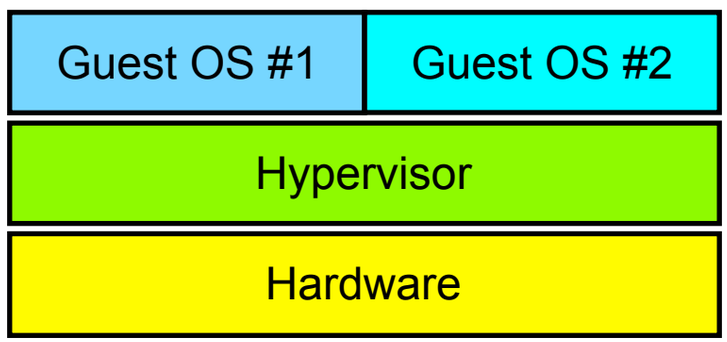
- Virtualisation concepts
- Full system virtualisation and virtual machines
- Jails, containers, and the benefits of imperfect virtualisation

Virtualisation Concepts

- Enable resource sharing by decoupling execution environments from physical hardware
  - Full system virtualisation – run multiple operating systems on a single physical host
  - Desirable for data centres and cloud hosting environments
  - Benefits for flexible management and security
- Use isolation to protect services
  - Virtualise resources of interest – lightweight, but imperfect, virtualisation
  - Constrain ability of a group of processes to access system resources
  - Enhances security within an operating system

Full System Virtualisation

- Partition system resources to allow more than one operating system to run on the same physical hardware
  - A *hypervisor* supports the partition, and is the operating system for the underlying hardware
  - G*uest operating systems* are processes that run on the hypervisor
  - The hypervisor API presents a virtual machine abstraction – each operating system thinks it's running on real hardware
  - i.e., the hypervisor emulates the hardware – requires CPU support for good performance



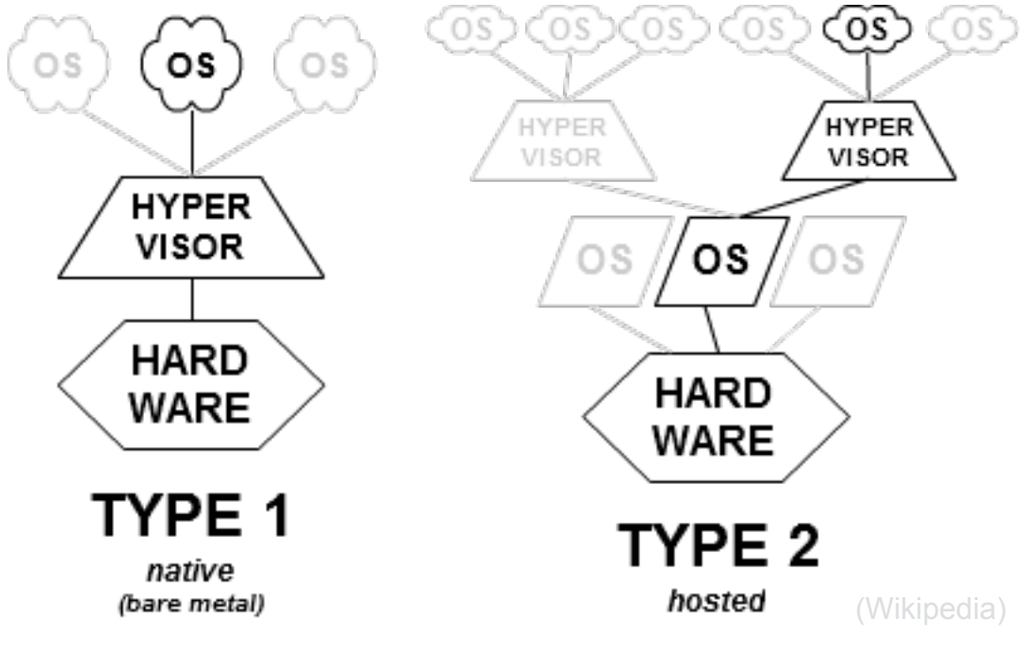
- First implemented IBM System/360 mainframe – mid-1960s
- Popular current implementations: Xen, VMWare, QEMU, VirtualBox



IBM System/360 (Wikipedia)

Types of Hypervisor

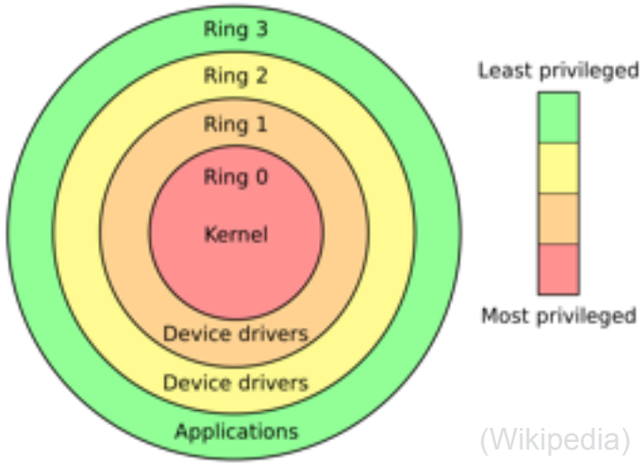
- Type 1 (“bare metal”) hypervisor:
  - The hypervisor is the operating system for the underlying hardware
  - It requires its own device drivers, and has to be ported to new hardware platforms
  - Xen
- Type 2 (“hosted”) hypervisor:
  - The hypervisor is an application running on an existing operating system
  - VMWare, VirtualBox



- Type 1 hypervisors have potentially lower overhead, but are more complex to implement and less portable
- Type 1 hypervisors can also make use of hardware virtualisation support, to offer better performance

Hardware Virtualisation

- Two approaches to virtualising the hardware:
- Complete virtualisation
  - Allow guest OS to run unchanged, performing any operations it desires – entirely untrusted
  - Processors have *protection rings* that allow various instructions to be performed – attempts to execute a privileged instruction from a higher ring cause a kernel trap and permission check
  - Full virtualisation requires either:
    - Ability for hypervisor to run in more privileged protection ring than the guest OS, allowing it to trap/emulate instructions that would expose virtualisation: not possible on older x86 processors, newer processors allow this
    - Ability for hypervisor to trap access to virtual memory, and dynamically re-write machine code to trap to the hypervisor, rather than execute an unsafe instruction – possible, but expensive on all x86 processors
- Paravirtualisation – provide a virtual machine that is similar, but not identical, to underlying hardware
  - Prohibit the guest OS from performing operations that are difficult to virtualise
  - Provide more convenient API for such operations
  - Requires cooperation from the guest OS – not for security, but for performance



(Wikipedia)

Xen



Xen and the Art of Virtualization

Paul Barham*, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris,
Alex Ho, Rolf Neugebauer†, Ian Pratt, Andrew Warfield

University of Cambridge Computer Laboratory
15 JJ Thomson Avenue, Cambridge, UK, CB3 0FD
{firstname.lastname}@cl.cam.ac.uk

ABSTRACT

Numerous systems have been designed which use virtualization to subdivide the ample resources of a modern computer. Some require specialized hardware, or cannot support commodity operating systems. Some target 100% binary compatibility at the expense of performance. Others sacrifice security or functionality for speed. Few offer resource isolation or performance guarantees; most provide only best-effort provisioning, risking denial of service.

This paper presents Xen, an x86 virtual machine monitor which allows multiple commodity operating systems to share conventional hardware in a safe and resource managed fashion, but without sacrificing either performance or functionality. This is achieved by providing an idealized virtual machine abstraction to which operating systems such as Linux, BSD and Windows XP, can be *ported* with minimal effort.

Our design is targeted at hosting up to 100 virtual machine instances simultaneously on a modern server. The virtualization approach taken by Xen is extremely efficient: we 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 unvirtualized case. We considerably outperform competing commercial and freely available solutions in a range of microbenchmarks and system-wide tests.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management; D.4.2 [**Operating Systems**]: Storage Management; D.4.8 [**Operating Systems**]: Performance

General Terms

Design, Measurement, Performance

Keywords

Virtual Machine Monitors, Hypervisors, Paravirtualization

*Microsoft Research Cambridge, UK
†Intel Research Cambridge, UK

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SOSP'03, October 19–22, 2003, Bolton Landing, New York, USA.
Copyright 2003 ACM 1-58113-757-5/03/0010 ...$5.00.

1. INTRODUCTION

Modern computers are sufficiently powerful to use virtualization to present the illusion of many smaller *virtual machines* (VMs), each running a separate operating system instance. This has led to a resurgence of interest in VM technology. In this paper we present Xen, a high performance resource-managed virtual machine monitor (VMM) which enables applications such as server consolidation [42, 8], co-located hosting facilities [14], distributed web services [43], secure computing platforms [12, 16] and application mobility [26, 37].

Successful partitioning of a machine to support the concurrent execution of multiple operating systems poses several challenges. Firstly, virtual machines must be isolated from one another: it is not acceptable for the execution of one to adversely affect the performance of another. This is particularly true when virtual machines are owned by mutually untrusting users. Secondly, it is necessary to support a variety of different operating systems to accommodate the heterogeneity of popular applications. Thirdly, the performance overhead introduced by virtualization should be small.

Xen hosts commodity operating systems, albeit with some source modifications. The prototype described and evaluated in this paper can support multiple concurrent instances of our XenoLinux guest operating system; each instance exports an application binary interface identical to a non-virtualized Linux 2.4. Our port of Windows XP to Xen is not yet complete but is capable of running simple user-space processes. Work is also progressing in porting NetBSD.

Xen enables users to dynamically instantiate an operating system to execute whatever they desire. In the XenoServer project [15, 35] we are deploying Xen on standard server hardware at economically strategic locations within ISPs or at Internet exchanges. We perform admission control when starting new virtual machines and expect each VM to *pay* in some fashion for the resources it requires. We discuss our ideas and approach in this direction elsewhere [21]; this paper focuses on the VMM.

There are a number of ways to build a system to host multiple applications and servers on a shared machine. Perhaps the simplest is to deploy one or more hosts running a standard operating system such as Linux or Windows, and then to allow users to install files and start processes — protection between applications being provided by conventional OS techniques. Experience shows that system administration can quickly become a time-consuming task due to complex configuration interactions between supposedly disjoint applications.

More importantly, such systems do not adequately support performance isolation; the scheduling priority, memory demand, network traffic and disk accesses of one process impact the performance of others. This may be acceptable when there is adequate provisioning and a closed user group (such as in the case of com-

- A modern type 1 hypervisor for x86
- Supports two types of virtualisation:
  - Full virtualisation – needs hardware support
    - Older x86 CPUs had no way of trapping supervisor mode instruction calls, to allow efficient emulation by a hypervisor; modern x86 CPUs do provide this
  - Paravirtualisation
    - Guest operating system is ported to a new system architecture, which looks like x86 with problematic features adapted to ease virtualisation
    - The operating system knows it is running in a VM
    - Processes running within the OS cannot tell that virtualisation is in use
- Control operations delegated to a privileged guest operating system
  - Known as *Domain0*
  - Provides policy and controls the hypervisor

Paravirtualisation

- Guest operating systems must be ported to Xen
- The Xen hypervisor provides an API to mediate hardware access by guest operating systems:
  - Privileged CPU instructions
  - Memory management
  - Interrupt handling
  - Access to network, disk, etc., hardware

Memory Management	
Segmentation	Cannot install fully-privileged segment descriptors and cannot overlap with the top end of the linear address space.
Paging	Guest OS has direct read access to hardware page tables, but updates are batched and validated by the hypervisor. A domain may be allocated discontiguous machine pages.
CPU	
Protection	Guest OS must run at a lower privilege level than Xen.
Exceptions	Guest OS must register a descriptor table for exception handlers with Xen. Aside from page faults, the handlers remain the same.
System Calls	Guest OS may install a 'fast' handler for system calls, allowing direct calls from an application into its guest OS and avoiding indirecting through Xen on every call.
Interrupts	Hardware interrupts are replaced with a lightweight event system.
Time	Each guest OS has a timer interface and is aware of both 'real' and 'virtual' time.
Device I/O	
Network, Disk, etc.	Virtual devices are elegant and simple to access. Data is transferred using asynchronous I/O rings. An event mechanism replaces hardware interrupts for notifications.

Table 1: The paravirtualized x86 interface.

- Required changes to guest systems are small
  - Porting guests to an API that allows efficient virtualisation allows significant performance benefits
  - Simplifies hypervisor design for x86, where the processor architecture makes full virtualisation expensive
- The user-space API of guest operating systems is unchanged

OS subsection	# lines	
	Linux	**XP**
Architecture-independent	78	1299
Virtual network driver	484	–
Virtual block-device driver	1070	–
Xen-specific (non-driver)	1363	3321
Total	**2995**	**4620**
(Portion of total x86 code base	**1.36%**	**0.04%)**

Table 2: The simplicity of porting commodity OSes to Xen. The cost metric is the number of lines of reasonably commented and formatted code which are modified or added compared with the original x86 code base (excluding device drivers).

Control and Management

- Xen provides mechanisms, but does not define policy
  - Virtualisation primitives are well understood and stable
  - Control mechanisms and policies evolve to match business models
  - Separation of mechanism and policy allows these to develop at separate rates

- Interactions via a mix of hypercalls and message passing
  - Hypercalls allow a guest operating system to call into the hypervisor
    - Implement the Domain0 control interface
    - Implement the paravirtualisation interface
  - Message passing allows the hypervisor to queue events to be processed by a guest
    - Callbacks to a guest, to indicate completion of a request or receipt of asynchronous data

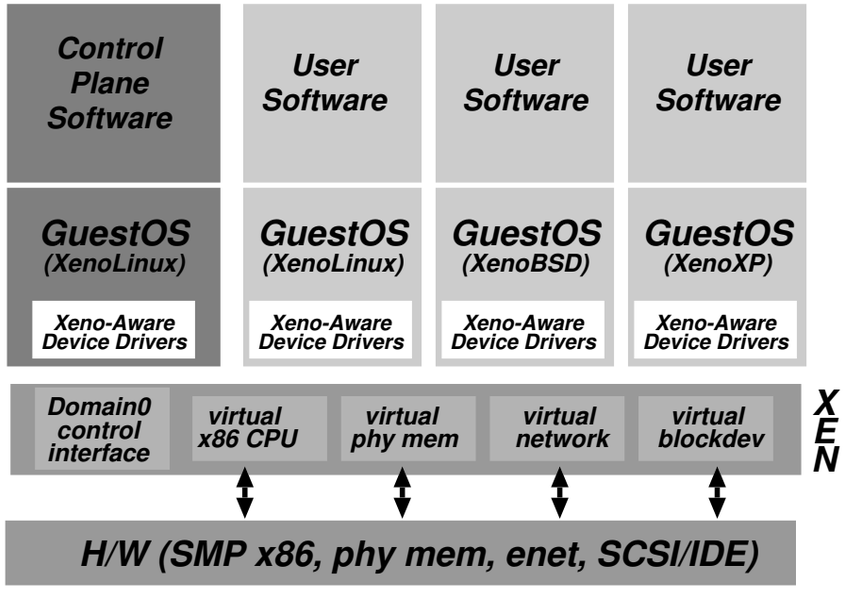
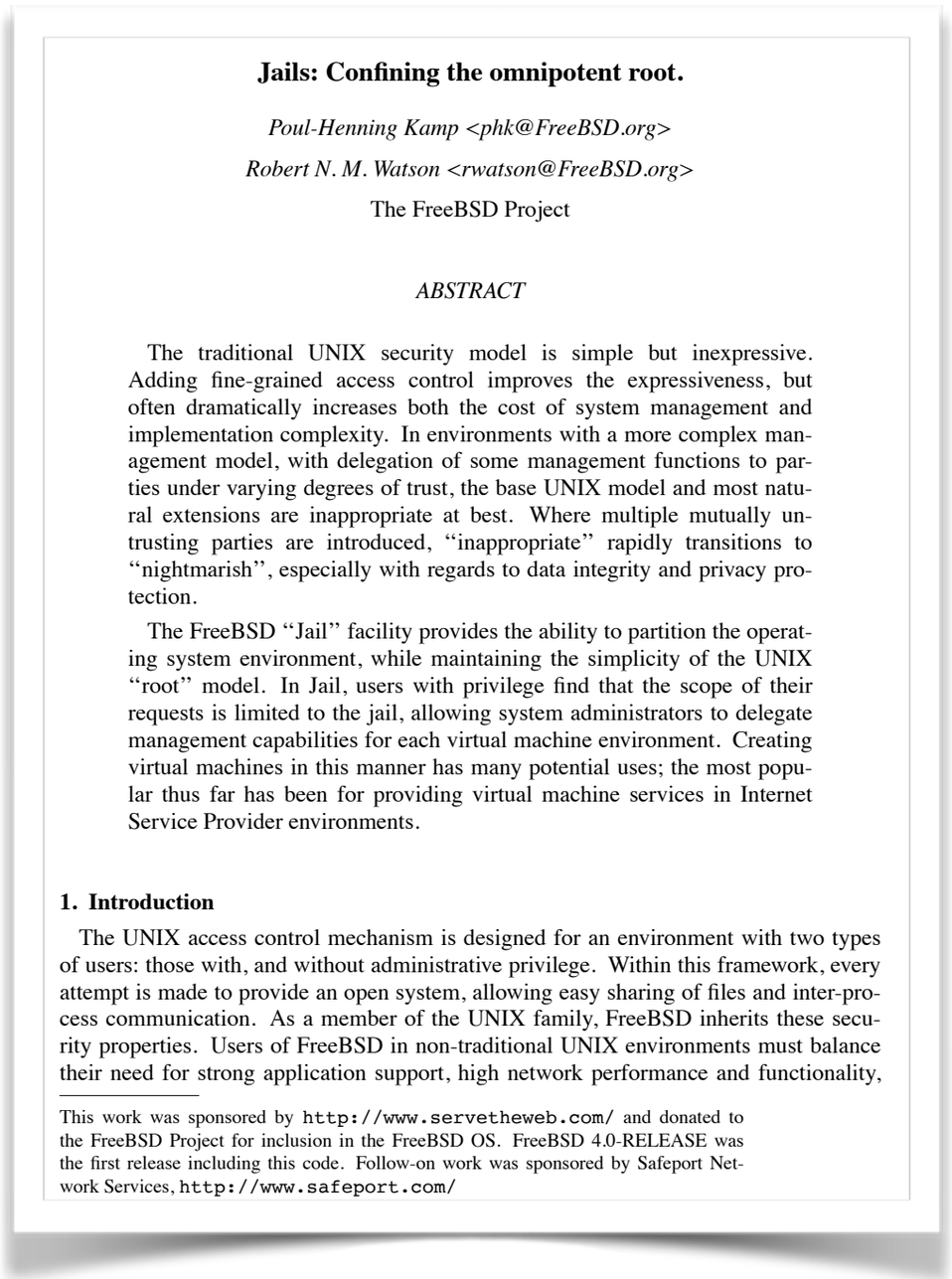


Figure 1: The structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including *Domain0* running control software in a XenoLinux environment.

Imperfect Virtualisation: Jails & Containers

- A full virtual machine is effective, but heavy-weight
- Alternatives offer lower cost, but less effective, virtualisation:
  - FreeBSD Jails
  - Linux containers – Docker, etc.
- Different implementations of the same basic idea: constrain processes to see a restricted subset of system resources
  - Jailed processes run within the existing operating system, and can tell that they are running in a jail

FreeBSD Jail subsystem



Jails: Confining the omnipotent root.

Poul-Henning Kamp <phk@FreeBSD.org>

Robert N. M. Watson <rwatson@FreeBSD.org>

The FreeBSD Project

ABSTRACT

The traditional UNIX security model is simple but inexpressive. Adding fine-grained access control improves the expressiveness, but often dramatically increases both the cost of system management and implementation complexity. In environments with a more complex management model, with delegation of some management functions to parties under varying degrees of trust, the base UNIX model and most natural extensions are inappropriate at best. Where multiple mutually untrusting parties are introduced, "inappropriate" rapidly transitions to "nightmarish", especially with regards to data integrity and privacy protection.

The FreeBSD "Jail" facility provides the ability to partition the operating system environment, while maintaining the simplicity of the UNIX "root" model. In Jail, users with privilege find that the scope of their requests is limited to the jail, allowing system administrators to delegate management capabilities for each virtual machine environment. Creating virtual machines in this manner has many potential uses; the most popular thus far has been for providing virtual machine services in Internet Service Provider environments.

1. Introduction

The UNIX access control mechanism is designed for an environment with two types of users: those with, and without administrative privilege. Within this framework, every attempt is made to provide an open system, allowing easy sharing of files and inter-process communication. As a member of the UNIX family, FreeBSD inherits these security properties. Users of FreeBSD in non-traditional UNIX environments must balance their need for strong application support, high network performance and functionality,

This work was sponsored by `http://www.servetheweb.com/` and donated to the FreeBSD Project for inclusion in the FreeBSD OS. FreeBSD 4.0-RELEASE was the first release including this code. Follow-on work was sponsored by Safeport Network Services, `http://www.safeport.com/`

- Unix has long had `chroot()`
  - Limits a process to see a subset of the filesystem
  - Used in, e.g., web servers, to prevent buggy code from serving files outside the chosen directory
- Jails extend this to also restrict access to other resources for jailed processes:
  - Limits a process to see a subset of the processes (the children of the initially jailed process)
  - Limits a process to see a specific network interface
  - Prevents a process from mounting/unmounting file systems, creating device nodes, loading kernel modules, changing kernel parameters, configuring network interfaces, etc.
  - Even if running with `root` privilege
- Processes outside the jail can see into the jail – from outside, jailed processes look like regular processes

Benefits of Jails

- Extremely lightweight implementation
  - Processes in the jail are regular Unix processes, with some additional privilege checks
  - FreeBSD implementation added/modified ~1,000 lines of code in total
  - Contents of the jail are limited: it's not a full operating system
- Straightforward system administration
  - Few new concepts: no need to learn how to operate the hypervisor
  - Visibility into the jail eases debugging and administration; no new tools
- Portable
  - No separate hypervisor, with its own device drivers, needed
  - If the base system runs, so do the jails

Disadvantages of Jails

- Virtualisation is imperfect
  - Full virtual machines can provide stronger isolation and performance guarantees – processes in jail can compete for resources with other processes in the system, and observe the effects of that competition
  - A process can discover that it's running in a jail – a process in a VM cannot determine that it's in a VM
- Jails are tied to the underlying operating system
  - A virtual machine can be paused, migrated to a different physical system, and resumed, without processes in the VM being aware – a jail cannot

Further Reading

- P. Barham *et al,* “Xen and the art of virtualization”, Proceedings of the ACM Symposium on Operating Systems Principles, October 2003. DOI:10.1145/945445.945462
  - Trade-offs of paravirtualisation vs. full virtualisation?
  - What needs to be done to port an OS to Xen?
  - Is paravirtualisation worthwhile, when compared to full system virtualisation?



Xen and the Art of Virtualization

Paul Barham[*], Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris,
Alex Ho, Rolf Neugebauer[†], Ian Pratt, Andrew Warfield

University of Cambridge Computer Laboratory
15 JJ Thomson Avenue, Cambridge, UK, CB3 0FD
{firstname.lastname}@cl.cam.ac.uk

ABSTRACT

Numerous systems have been designed which use virtualization to subdivide the ample resources of a modern computer. Some require specialized hardware, or cannot support commodity operating systems. Some target 100% binary compatibility at the expense of performance. Others sacrifice security or functionality for speed. Few offer resource isolation or performance guarantees; most provide only best-effort provisioning, risking denial of service.

This paper presents Xen, an x86 virtual machine monitor which allows multiple commodity operating systems to share conventional hardware in a safe and resource managed fashion, but without sacrificing either performance or functionality. This is achieved by providing an idealized virtual machine abstraction to which operating systems such as Linux, BSD and Windows XP, can be *ported* with minimal effort.

Our design is targeted at hosting up to 100 virtual machine instances simultaneously on a modern server. The virtualization approach taken by Xen is extremely efficient: we 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 unvirtualized case. We considerably outperform competing commercial and freely available solutions in a range of microbenchmarks and system-wide tests.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management; D.4.2 [**Operating Systems**]: Storage Management; D.4.8 [**Operating Systems**]: Performance

General Terms

Design, Measurement, Performance

Keywords

Virtual Machine Monitors, Hypervisors, Paravirtualization

[*]Microsoft Research Cambridge, UK
[†]Intel Research Cambridge, UK

1. INTRODUCTION

Modern computers are sufficiently powerful to use virtualization to present the illusion of many smaller *virtual machines* (VMs), each running a separate operating system instance. This has led to a resurgence of interest in VM technology. In this paper we present Xen, a high performance resource-managed virtual machine monitor (VMM) which enables applications such as server consolidation [42, 8], co-located hosting facilities [14], distributed web services [43], secure computing platforms [12, 16] and application mobility [26, 37].

Successful partitioning of a machine to support the concurrent execution of multiple operating systems poses several challenges. Firstly, virtual machines must be isolated from one another: it is not acceptable for the execution of one to adversely affect the performance of another. This is particularly true when virtual machines are owned by mutually untrusting users. Secondly, it is necessary to support a variety of different operating systems to accommodate the heterogeneity of popular applications. Thirdly, the performance overhead introduced by virtualization should be small.

Xen hosts commodity operating systems, albeit with some source modifications. The prototype described and evaluated in this paper can support multiple concurrent instances of our XenoLinux guest operating system; each instance exports an application binary interface identical to a non-virtualized Linux 2.4. Our port of Windows XP to Xen is not yet complete but is capable of running simple user-space processes. Work is also progressing in porting NetBSD.

Xen enables users to dynamically instantiate an operating system to execute whatever they desire. In the XenoServer project [15, 35] we are deploying Xen on standard server hardware at economically strategic locations within ISPs or at Internet exchanges. We perform admission control when starting new virtual machines and expect each VM to pay in some fashion for the resources it requires. We discuss our ideas and approach in this direction elsewhere [21]; this paper focuses on the VMM.

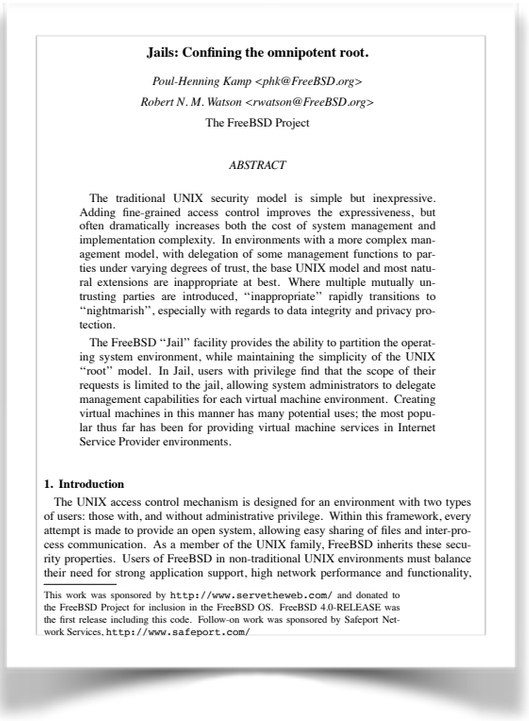
There are a number of ways to build a system to host multiple applications and servers on a shared machine. Perhaps the simplest is to deploy one or more hosts running a standard operating system such as Linux or Windows, and then to allow users to install files and start processes — protection between applications being provided by conventional OS techniques. Experience shows that system administration can quickly become a time-consuming task due to complex configuration interactions between supposedly dis-

joint applications.

More importantly, such systems do not adequately support performance isolation; the scheduling priority, memory demand, network traffic and disk accesses of one process impact the performance of others. This may be acceptable when there is adequate provisioning and a closed user group (such as in the case of com-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SOSP'03, October 19–22, 2003, Bolton Landing, New York, USA.
Copyright 2003 ACM 1-58113-757-5/03/0010 ...$5.00.

- P.-H. Kamp and R. Watson, “Jails: Confining the omnipotent root”, Proceedings of the System Administration and Network Engineering Conference, May 2000. http://www.sane.nl/events/sane2000/papers/kamp.pdf
  - Trade-offs vs. complete system virtualisation?
  - Overheads vs. flexibility vs. ease of management?
  - Jails vs. Docker vs. …?



Jails: Confining the omnipotent root.

Poul-Henning Kamp <phk@FreeBSD.org>
Robert N. M. Watson <rwatson@FreeBSD.org>
The FreeBSD Project

ABSTRACT

The traditional UNIX security model is simple but inexpressive. Adding fine-grained access control improves the expressiveness, but often dramatically increases both the cost of system management and implementation complexity. In environments with a more complex management model, with delegation of some management functions to parties under varying degrees of trust, the base UNIX model and most natural extensions are inappropriate at best. Where multiple mutually untrusting parties are introduced, “inappropriate” rapidly transitions to “nightmarish”, especially with regards to data integrity and privacy protection.

The FreeBSD “Jail” facility provides the ability to partition the operating system environment, while maintaining the simplicity of the UNIX “root” model. In Jail, users with privilege find that the scope of their requests is limited to the jail, allowing system administrators to delegate management capabilities for each virtual machine environment. Creating virtual machines in this manner has many potential uses; the most popular thus far has been for providing virtual machine services in Internet Service Provider environments.

1. Introduction

The UNIX access control mechanism is designed for an environment with two types of users: those with, and without administrative privilege. Within this framework, every attempt is made to provide an open system, allowing easy sharing of files and inter-process communication. As a member of the UNIX family, FreeBSD inherits these security properties. Users of FreeBSD in non-traditional UNIX environments must balance their need for strong application support, high network performance and functionality,

This work was sponsored by `http://www.servetheweb.com/` and donated to the FreeBSD Project for inclusion in the FreeBSD OS. FreeBSD 4.0-RELEASE was the first release including this code. Follow-on work was sponsored by Safeport Network Services, `http://www.safeport.com/`