University of Glasgow

School of Computing Science

60 YEARS OF COMPUTING AT GLASGOW

Virtualisation

Advanced Operating Systems

Tutorial 9

Virtualisation and Hypervisors

- Full system virtualisation
  - Concepts
  - Hypervisor mode; CPU support
  - Paravirtualisation
  - Type 1 and Type 2 hypervisors
  - Systems management and live migration
- Example: Xen

Discussion: Xen

- P. Barham et al, “Xen and the art of virtualization”, Proc. 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?
  - How do Dom0 and device drivers work?

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

SOSP'03, October 19–22, 2003, Bolton Landing, New York, USA.

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-

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-

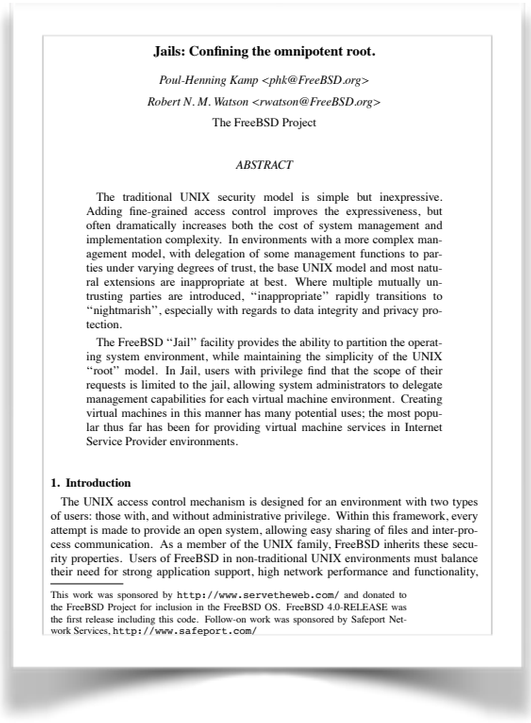
joint applications.

Jails and Unikernels

- Alternatives to full virtualisation
  - Jails, containers, and sandboxes
  - Benefits: lightweight, straightforward administration, portable
  - Disadvantages: imperfect virtualisation and security; tied to a physical machine
- Container management
  - Docker: standardised way of packaging an image to run in a container
  - Unikernels: library operating systems; high-level languages; customised to the application

Discussion: Jails and Unikernels

- P.-H. Kamp and R. Watson, “Jails: Confining the omnipotent root”, Proc. 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?
  - Benefits of Docker-style configuration of images



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.

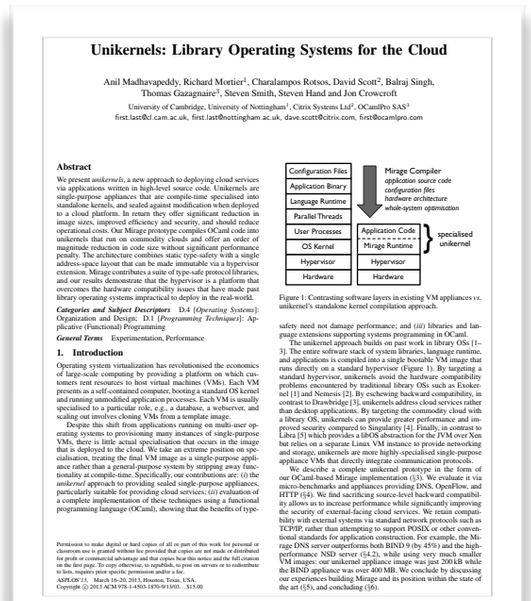
The FreeBSD ‘‘Jail’’ facility provides the ability to partition the operating system environment, while maintaining the simplicity of the UNIX ‘‘root’’ model. In Jail, users with privilege find that the scope of their requests is limited to the jail, allowing system administrators to delegate management 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/

- A. Madhavapeddy et al., “Unikernels: Library Operating Systems for the Cloud”, Proc. ACM ASPLOS, Houston, TX, USA, March 2013. DOI:10.1145/2451116.2451167
  - Is optimising an operating system for a single application going too far?
  - Are unikernels maintainable?
  - Relation to containers and hypervisors?



Unikernels: Library Operating Systems for the Cloud

Anil Madhavapeddy, Richard Mortier[1], Charalampos Rotsos, David Scott[2], Balraj Singh, Thomas Gazagnaire[3], Steven Smith, Steven Hand and Jon Crowcroft

University of Cambridge, University of Nottingham[1], Citrix Systems Ltd[2], OCamlPro SAS[3]

first.last@cl.cam.ac.uk, first.last@nottingham.ac.uk, dave.scott@citrix.com, first@ocamlpro.com

Abstract

We present *unikernels*, a new approach to deploying cloud services via applications written in high-level source code. Unikernels are single-purpose appliances that are compile-time specialised into standalone kernels, and sealed against modification when deployed to a cloud platform. In return they offer significant reduction in image sizes, improved efficiency and security, and should reduce operational costs. Our Mirage prototype compiles OCaml code into unikernels that run on commodity clouds and offer an order of magnitude reduction in code size without significant performance penalty. The architecture combines static type-safety with a single address-space layout that can be made immutable via a hypervisor extension. Mirage contributes a suite of type-safe protocol libraries, and our results demonstrate that the hypervisor is a platform that overcomes the hardware compatibility issues that have made past library operating systems impractical to deploy in the real-world.

Categories and Subject Descriptors D.4 [*Operating Systems*]: Organization and Design; D.1 [*Programming Techniques*]: Applicative (Functional) Programming

General Terms Experimentation, Performance

1. Introduction

Operating system virtualization has revolutionised the economics of large-scale computing by providing a platform on which customers rent resources to host virtual machines (VMs). Each VM presents as a self-contained computer, booting a standard OS kernel and running unmodified application processes. Each VM is usually specialised to a particular role, e.g., a database, a webserver, and scaling out involves cloning VMs from a template image.

Despite this shift from applications running on multi-user operating systems to provisioning many instances of single-purpose VMs, there is little actual specialisation that occurs in the image that is deployed to the cloud. We take an extreme position on specialisation, treating the final VM image as a single-purpose appliance rather than a general-purpose system by stripping away functionality at compile-time. Specifically, our contributions are: (*i*) the *unikernel* approach to providing sealed single-purpose appliances, particularly suitable for providing cloud services; (*ii*) evaluation of a complete implementation of these techniques using a functional programming language (OCaml), showing that the benefits of type-safety need not damage performance; and (*iii*) libraries and language extensions supporting systems programming in OCaml.

Figure 1: Contrasting software layers in existing VM appliances *vs.* unikernel’s standalone kernel compilation approach.

The unikernel approach builds on past work in library OSs [1–3]. The entire software stack of system libraries, language runtime, and applications is compiled into a single bootable VM image that runs directly on a standard hypervisor (Figure 1). By targeting a standard hypervisor, unikernels avoid the hardware compatibility problems encountered by traditional library OSs such as Exokernel [1] and Nemesis [2]. By eschewing backward compatibility, in contrast to Drawbridge [3], unikernels address cloud services rather than desktop applications. By targeting the commodity cloud with a library OS, unikernels can provide greater performance and improved security compared to Singularity [4]. Finally, in contrast to Libra [5] which provides a libOS abstraction for the JVM over Xen but relies on a separate Linux VM instance to provide networking and storage, unikernels are more highly-specialised single-purpose appliance VMs that directly integrate communication protocols.

We describe a complete unikernel prototype in the form of our OCaml-based Mirage implementation (§3). We evaluate it via micro-benchmarks and appliances providing DNS, OpenFlow, and HTTP (§4). We find sacrificing source-level backward compatibility allows us to increase performance while significantly improving the security of external-facing cloud services. We retain compatibility with external systems via standard network protocols such as TCP/IP, rather than attempting to support POSIX or other conventional standards for application construction. For example, the Mirage DNS server outperforms both BIND 9 (by 45%) and the high-performance NSD server (§4.2), while using very much smaller VM images: our unikernel appliance image was just 200 kB while the BIND appliance was over 400 MB. We conclude by discussing our experiences building Mirage and its position within the state of the art (§5), and concluding (§6).