Virtual Machine Monitors:

Technology and Trends

Zhefu Jiang

zj46@cornell.edu

Xen and the Art of Virtualization

- Paul Barham
  - "I am currently a Principal Researcher at Microsoft Research Cambridge, where I lead the Systems and Networking group. "
  - Finished his Ph.D. degree at University of Cambridge, for the Nemenisis Operating System (QoS friendly OS, microkernel)

- Keir Frase
  - Senior architect at Citrix and has worked on the Xen hypervisor since 2002.
  - Received a PhD for work on concurrency management and lock-free data structures in 2004
  - Was a lecturer in the University of Cambridge Computer Laboratory

- Ian Pratt
  - Used to be a senior Lecturer at the University of Cambridge Computer Lab
  - Co-Founder of Xensource (developing Xen) --> CTO at Citrix (after the acquire) --> Start Bromium

Virtualization: Requirement

- Popek and Goldberg virtualization requirements
  - From the paper *Formal Requirements for Virtualizable Third Generation Architectures*
  - Theorem 1. For any conventional third generation computer, a virtual machine monitor (Or hypervisor) may be constructed if the set of **sensitive instruction** for that computer is a subset of the set of **privileged instructions.**
  - Theorem 2. A conventional third-generation computer is recursively virtualizable if it is virtualizable and a VMM without any timing dependencies can be constructed for it.

Virtualization on X86: challenges

- Sensitive but non-privilege instructions:
  - cli, popf, cpuid, etc. Totally 17 such instructions. (Can be emulated?)
  - If pagetable is to be changed, even mov is such kind of instruction. (Isolation?)

Virtualization on X86: Solution

- To the other extreme: emulation
  - QEMU
  - Not efficient enough (at least 5x - 10x performance degradation)
- Selective Emulation - Binary Translation
  - replace sensitive instructions with emulated ones from instruction translation buffer
  - Can be counted as Virtualization: VMware VirtualBox, etc
- (Current) Patch X86 architecture
  - Force the 17 instructions to trap: Intel VT-x/AMD SVM, EPT/NPT, VT-D, IO-Vector
  - KVM
- Patch OS - Paravirtualization
  - **Xen: the art of virtualization**
  - Also support hardware-assist Virtualization

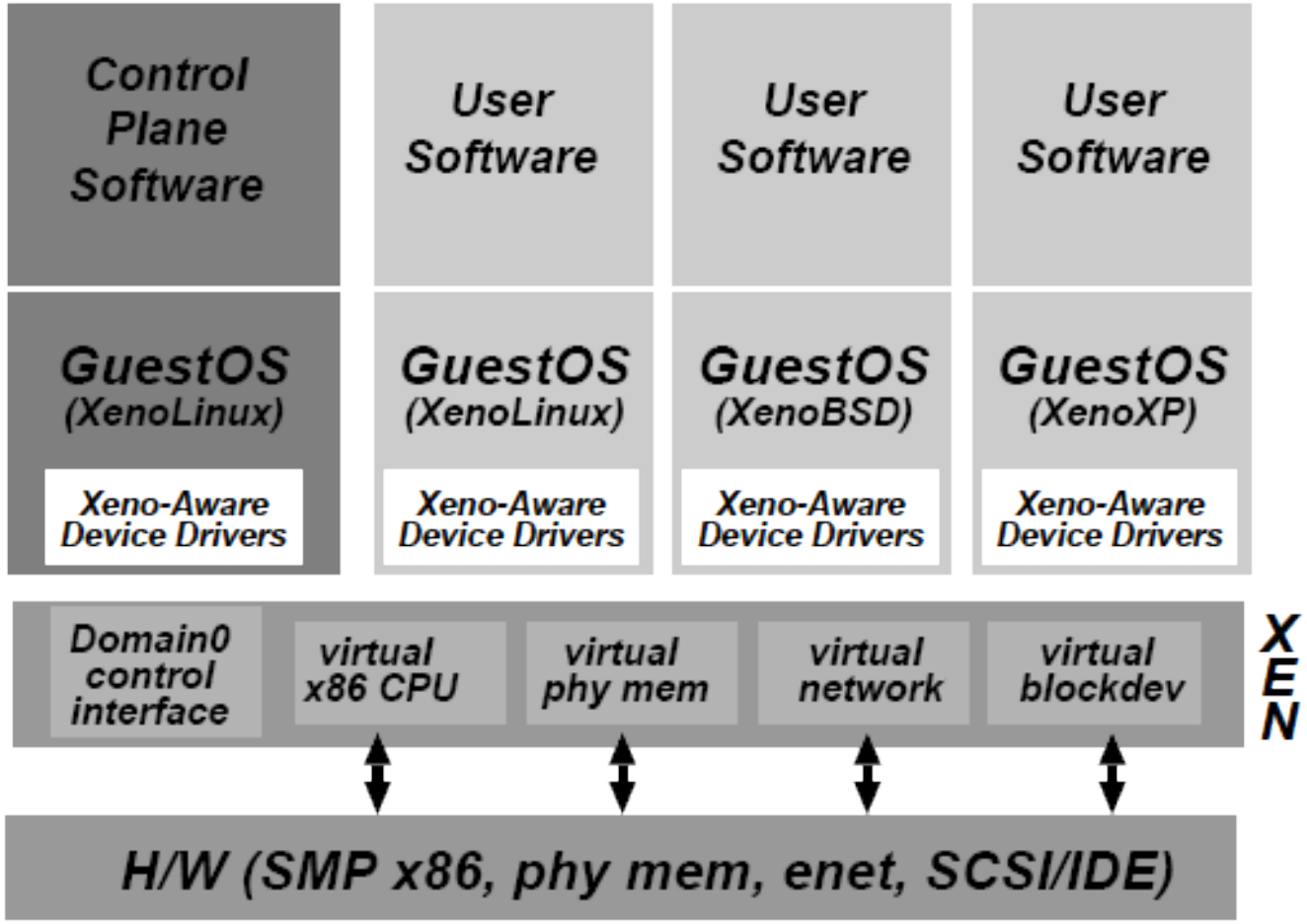
The basic idea of Xen

- Replace sensitive but non-privileged instructions with "Hypercalls"
  - Multiplexing CPU
- Using "Event Channel" to simulate interrupts
  - Multiplexing interrupt
- Protect page table to manage memory
  - Multiplexing memory
  - Balloon driver
- Using "Splitted Device" to virtualize I/O
  - Multiplexing I/O devices

Terminology in Xen

- Xen
  - The hypervisor
- Domains
  - The Virtual Machines
  - DomainU: general guest virtual machine
  - Domain0: privileged guest virtual machine which has the control interface of xen and manages devices
  - Driver Domain: Domain which can access devices

Running on top of Xen

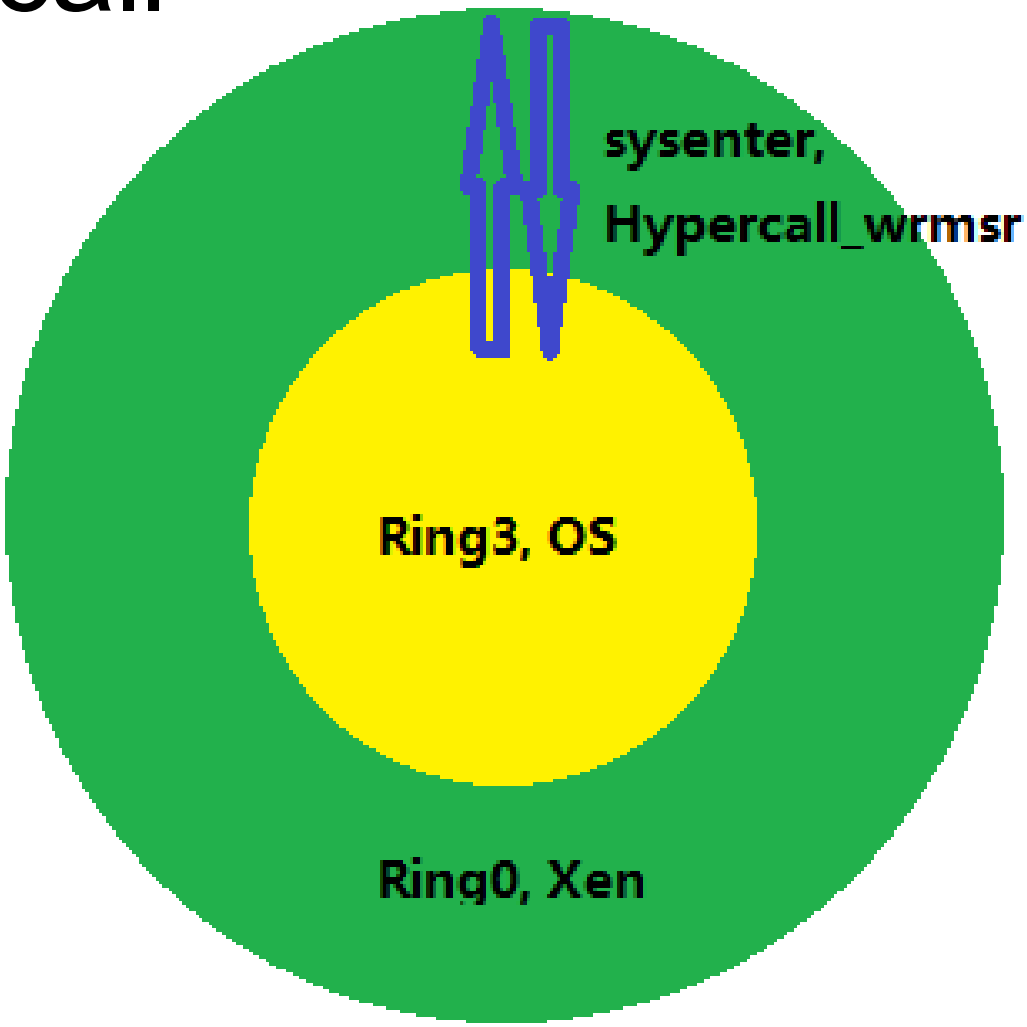


Hypercall

- Ring compression
  - Xen itself occupies the privileged ring
  - Move guest OS to other ring (ring1 for x86_32 and ring3 for x86_64)
  - applications run at lowest ring
  - maintains Hypervisor's integrity

- Replace non-privileged sensitive instruction with Hypercall
  - For x86_32: int82 with arguments
  - For x86_64: sysenter with arguments
  - Xen service the hypercall and maintain the symantics as if the OS was executing a real non-privileged sensitive instruction

Hypercall



Event Channel & Upcall

- Multiplex interrupts for Virtual Machine
  - Event Channel is essentially a bitmap marking pending events
  - To the guest OS, event channel is equal to interrupt lines
- Xen can also call a Domain's pre-installed handler directly
  - To implement system call for virtual machine

Xen's memory management

- Guest kernel should update pagetable using hypercall
  - guest OS initally runs in protected mode, with readonly pagetable
  - guest OS update pagetable by hypercalls, Xen keep track of page's type (L1/L2/L3 page table or normal page)
  - guest OS can touch page table directly, but it shouldn't because of the cost of emulation

Xen's memory management

- P2M mapping
  - Xen designate the mapping of guest physical memory to machine physical address in P2M table
  - Xen also have a global table which maps machine pages to each VM

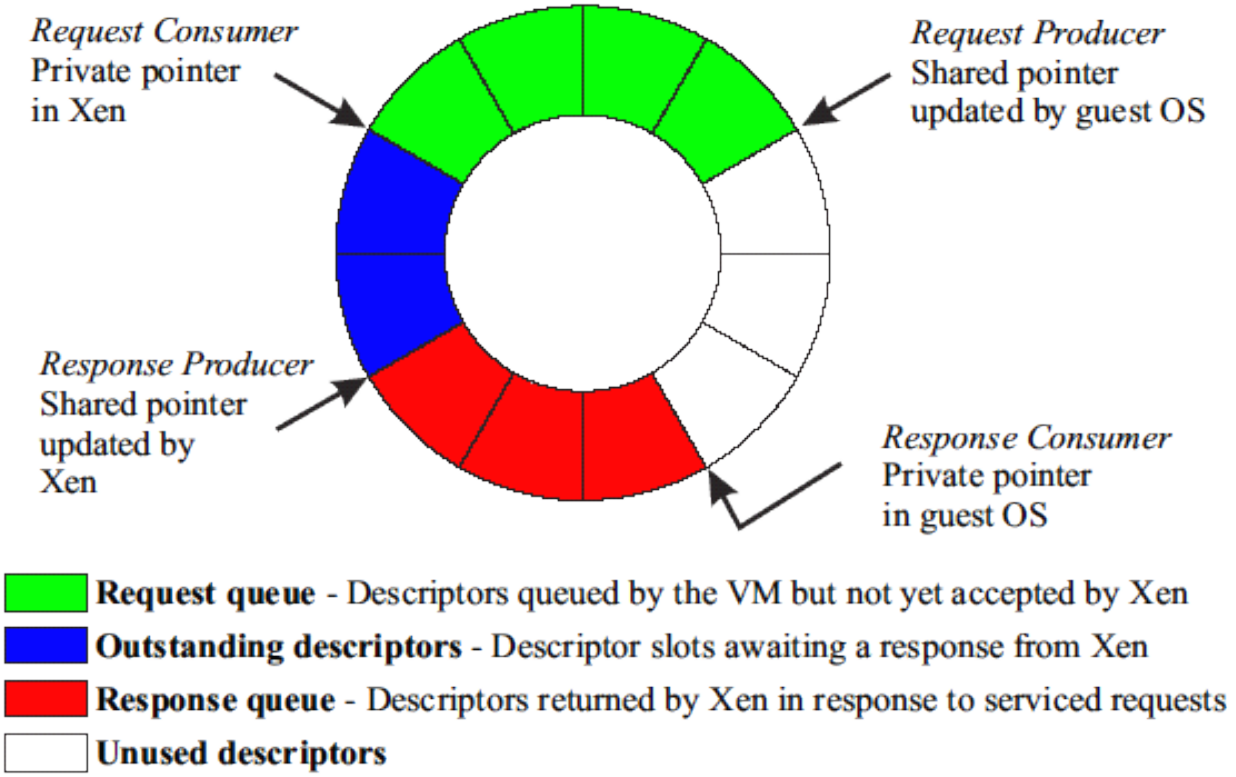
Xen's memory management

- Balloon driver
  - Runs in DomU, sqeeze DomU's memory in an aggressive manner
  - Returns the pages to Xen by hypercalls
  - may cause swapping in DomU

Xen's splitted device model

- Special Disk and Network device (VBD and XENNET) are created and are enumerable on a special virtual BUS -- The XENBUS
  - XENBUS is actually a block of shared memory passing formated messages
  - The special devices are called "Frontends", which are drived by special drivers.
  - On Dom0, there are respective Backends which performs the real IO
    - Frontends talks with Backends by event channel and **Ring buffer**
    - BLKTAP2 for VBD
    - NETBACK for XENNET

RingBuffer


Request Consumer
Private pointer
in Xen
Request Producer
Shared pointer
updated by guest OS
Response Producer
Shared pointer
updated by
Xen
Response Consumer
Private pointer
in guest OS
Request queue - Descriptors queued by the VM but not yet accepted by Xen
Outstanding descriptors - Descriptor slots awaiting a response from Xen
Response queue - Descriptors returned by Xen in response to serviced requests
Unused descriptors

Xen's splitted device model

- Data Flows
  - Frontend to Backend
    - Frontend put data pointers into Ringbuffer
    - Frontend use event channel to notify Backend
    - Backend maps data to its address space and perform I/O
  - Backend to Frontend
    - Vise-Visa

Evaluation

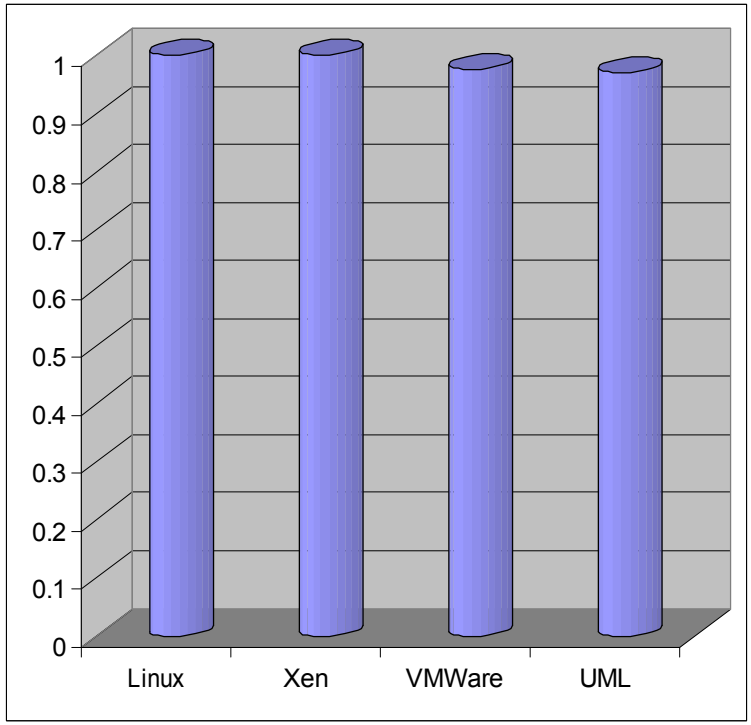
- Cost of adapting an OS: is it unacceptably expensive to modify an commodity OS to run on XEN?
  - Architecture Independent (78 lines)
  - Virtual Block Device driver (1070 lines)
  - Virtual Network driver (484 lines)
  - Xen specific (1363 lines)

  < 2% of code-base
- Paravirtualize mode and Xen virtual device drivers are actually part of today's Linux system
  - No more than 10000 lines, which is highly structual

Evaluation

- Against other virtualization techniques
  - Vmware, User Mode Linux(UML)
- Single Native OS vs Virtual Machine
  - Running multiple applications on a native OS vs a guest OS
- Performance Isolation between Guest OSs
- Overhead of running large number of OSs

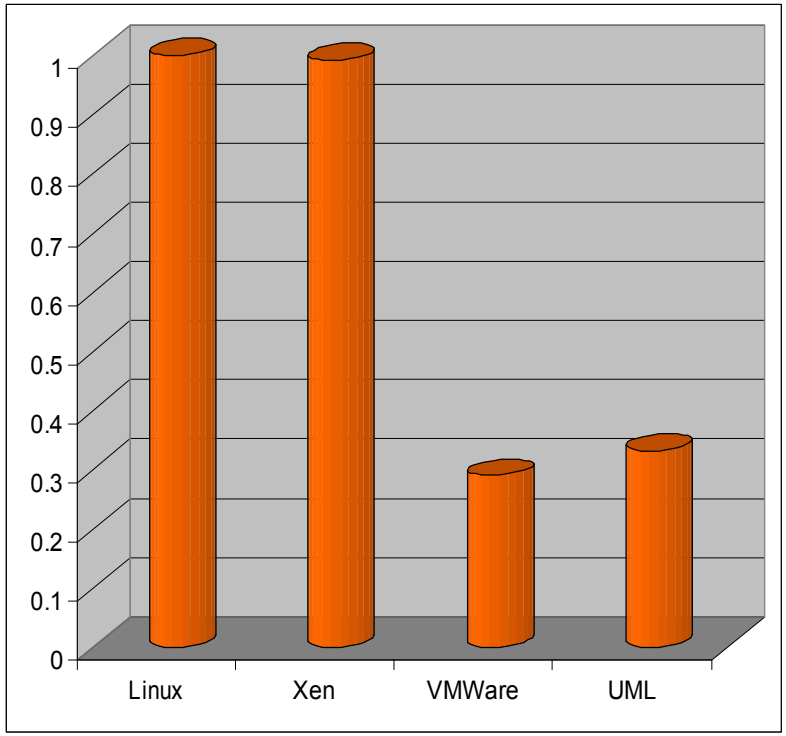
Relative Performance



SPEC INT2000 score

CPU Intensive

Little I/O and OS interaction



SPEC WEB99

180Mb/s TCP traffic

Disk read-write on 2GB dataset

O.S Benchmarks

Context switching times – extra overhead due to hypercall required to change the page table base.

Config	2p 0K	2p 16K	2p 64K	8p 16K	8p 64K	16p 16K	16p 64K
L-SMP	1.69	1.88	2.03	2.36	26.8	4.79	38.4
L-UP	0.77	0.91	1.06	1.03	24.3	3.61	37.6
Xen	**1.97**	**2.22**	**2.67**	**3.07**	**28.7**	**7.08**	39.4
VMW	18.1	17.6	21.3	22.4	51.6	41.7	72.2
UML	15.5	14.6	14.4	16.3	36.8	23.6	52.0

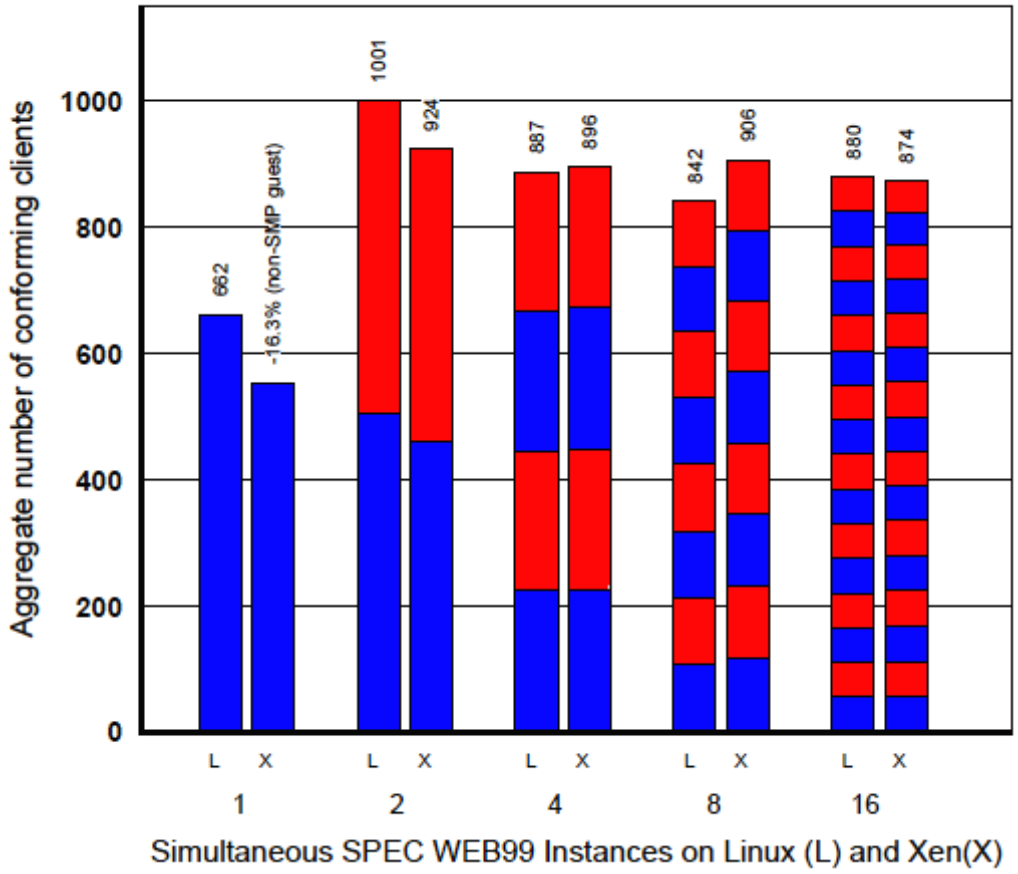
Table 4: `lmbench`: Context switching times in μs

Concurrent Virtual Machines

Multiple Apache processes in Linux

vs.

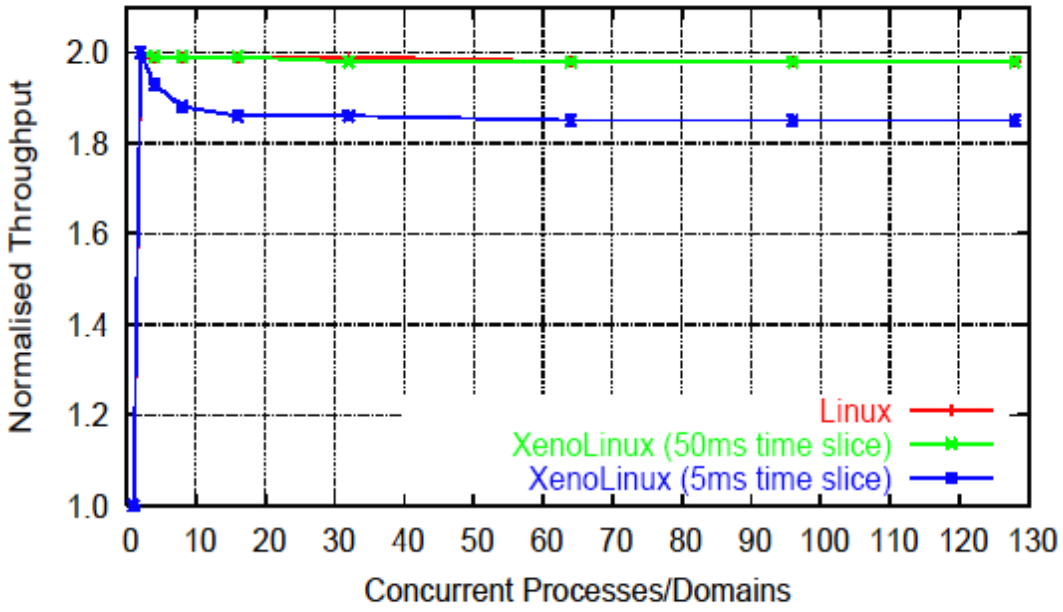
One Apache process in each guest OS



Performance Isolation

- 4 Domains
- 2 running benchmarks
- 1 running dd
- 1 running a fork bomb in the background
- 2 antisocial domains contributed only 4% performance degradation

Scalability



Normalized aggregate performance of a subset of SPEC CINT2000 running concurrently on 1-128 domains

Issues

- Extra effort is required to port every version of every OS to Xen
  - Demonstrated by the ‘ongoing effort’ to port Windows XP and BSD
  - During 2003-2010, the painful efforts to be accepted into Linux Kernel (Finally, succeeded, and Xen survives)
- Dom0 may be the key issue of security
  - Dom0 is actually a Linux and because Dom0 can actually controll Xen, that means the security of the whole system is no better than original Linux
- Coarse Cache management
  - TLB
  - CPU instruction/data cache
- Schedule issues

Are Virtual Machine Monitor Microkernels Done Right?

•Steven Hand, Keir Fraser, Evangelos Kotsovinos

Cambridge University, UK

•Andrew Warfield

University of British Columbia

•Dan Magenheimer

HP labs, Fort Collins

Wrote the first PA-RISC simulator

Developed Vblades, the first Itanium VMM

Virtualization vs Microkernel

	Platform-abstraction	Interdomain communication	Security	Scalability
Virtualization	Simulated machine architecture	Shared memory/Virtual Network/Block devices	High. Malicious actions are limited within a VM	Similar with distributed systems
Microkernel	simplified system calls	Mainly IPC	High. Malicious actions are limited within an application or a service	Can achieve best effect on single machine: Tornado

The Arguments (Open to discussion)

- Avoid Liability Inversion
  - E.g. for Microkernel, the kernel itself relies on user-mode pager, which may cause the kernel to wait(Is that so? Virtual Machine doesn't have liability inversion? How about Xen's Dom0?)
- Make IPC performance Irrelevant
  - Virtualization leave the problem to distributed systems.
  - Microkernel's effort may still be effective on single machine
- Treat the OS as a component
  - compatibility

Current Trends

- Virtualization towards Microkernel
  - SOSP 2011 *Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor,* they split the functionality of Dom0 into several VMs
  - OS as a service
- Microkernel towards Virtualization
  - L4 is so flexible that some variations of L4 are actually paravirtualization
- Evolution is still on-going