EECS 262a
Advanced Topics in Computer Systems
Lecture 13

Virtual Machines
October 9th, 2023

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs262

Today's Papers

- Disco: Running Commodity Operating Systems on Scalable Multiprocessors". *ACM Transactions on Computer Systems* **15** (4).
  Edouard Bugnion; Scott Devine; Kinshuk Govil; Mendel Rosenblum (November 1997).
- Xen and the Art of Virtualization
  P. Barham, B. Dragovic, K Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield. Appears in *Symposium on Operating System Principles* (SOSP), 2003

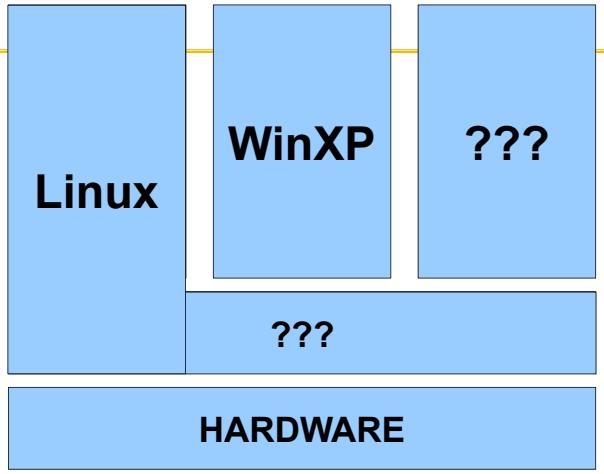
- Thoughts?

Why Virtualize?

- Consolidate machines
  - Huge energy, maintenance, and management savings
- Isolate performance, security, and configuration
  - Stronger than process-based
- Stay flexible
  - Rapid provisioning of new services
  - Easy failure/disaster recovery (when used with data replication)
- Cloud Computing
  - Huge economies of scale from multiple tenants in large datacenters
  - Savings on mgmt, networking, power, maintenance, purchase costs
- Corporate employees
  - Can choose own devices

Virtual Machines Background

- Observation: instruction-set architectures (ISA) form some of the relatively few well-documented complex interfaces we have in world
  - Machine interface includes meaning of interrupt numbers, programmed I/O, DMA, etc.
- Anything that implements this interface can execute software for that platform
- A *virtual machine* is a software implementation of this interface (often using the same underlying ISA, but not always)
  - Original paper on whether or not a machine is virtualizable: Gerald J. Popek and Robert P. Goldberg (1974). "Formal Requirements for Virtualizable Third Generation Architectures". *Communications of the ACM* 17 (7): 412 –421.



Many VM Examples

- IBM initiated VM idea to support legacy binary code
  - Support an old machine's on a newer machine (e.g., CP-67 on System 360/67)
  - Later supported multiple OS's on one machine (System 370)
- Apple's Rosetta ran old PowerPC apps on newer x86 Macs
- MAME is an emulator for old arcade games (5800+ games!!)
  - Actually executes the game code straight from a ROM image
- Modern VM research started with Stanford's Disco project
  - Ran multiple VM's on large shared-memory multiprocessor (since normal OS's couldn't scale well to lots of CPUs)
- VMware (founded by Disco creators):
  - Customer support (many variations/versions on one PC using a VM for each)
  - Web and app hosting (host many independent low-utilization servers on one machine – "server consolidation")

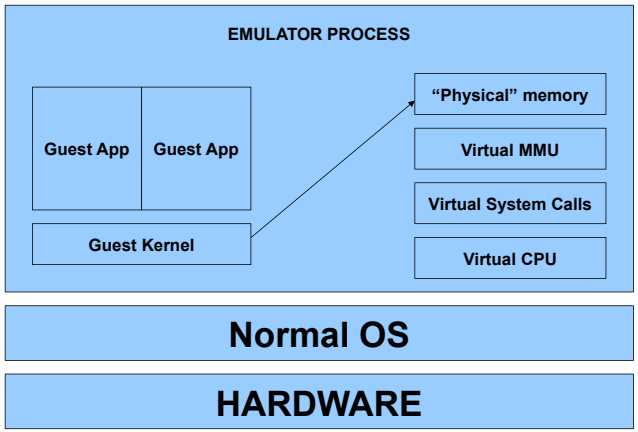
VM Basics

- The real master is no longer the OS but the "Virtual Machine Monitor" (VMM) or "Hypervisor" (hypervisor > supervisor)
- OS no longer runs in most privileged mode (reserved for VMM)
  - x86 has four privilege "rings" with ring 0 having full access
  - VMM = ring 0, OS = ring 1, app = ring 3
  - x86 rings come from Multics (also x86 segment model)
    - » (Newer x86 has fifth "ring" for hypervisor, but not available at time of papers)
- But OS thinks it is running in most privileged mode and still issues those instructions?
  - Ideally, such instructions should cause traps and the VMM then emulates the instruction to keep the OS happy
  - But in (old) x86, some such instructions fail silently!
  - Four solutions: SW emulation (Disco), dynamic binary code rewriting (VMware), slightly rewrite OS (Xen), hardware virtualization (IBM System/370, IBM LPAR, Intel VT-x, AMD-V, SPARC T-series)

Virtualization Approaches

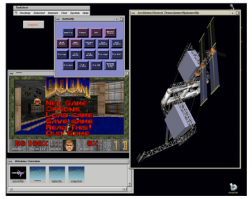
- Disco/VMware/IBM: Complete virtualization – runs unmodified OSs and applications
  - Use software emulation to shadow system data structures,
  - Dynamic binary rewriting of OS code that modifies system structures, or
  - Hardware virtualization support
- Denali introduced the idea of "paravirtualization" – change interface some to improve VMM performance/simplicity
  - Must change OS and some apps (e.g., those using segmentation) – easy for Linux, hard for MS (requires their help!)
  - But can support 1,000s of VMs on one machine...
  - Great for web hosting
- Xen: change OS but not applications – support the full *Application Binary Interface* (ABI)
  - Faster than a full VM – supports ~100 VMs per machine
  - Moving to a paravirtual VM is essentially porting the software to a very similar machine

How to Build a VMM 1: SW Emulation (Disco)

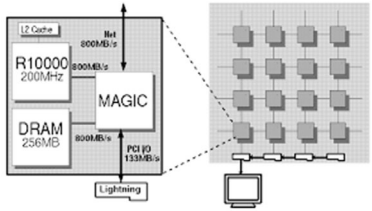


Disco

- Extend modern OS to run efficiently on shared memory multiprocessors with minimal OS changes
- A VMM built to run multiple copies of Silicon Graphics IRIX operating system on Stanford Flash multiprocessor

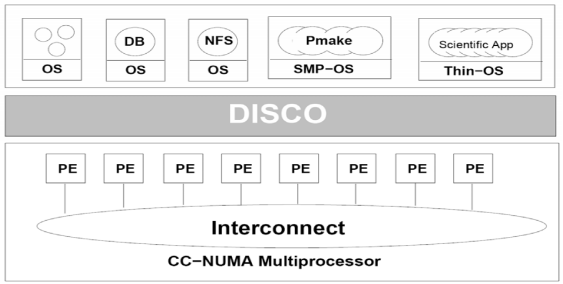


IRIX Unix based OS



Stanford FLASH: cache coherent NUMA

Disco Architecture



Disco's Interface

- Processors
  - MIPS R10000 processor: emulates all instructions, MMU, trap architecture
  - Extension to support common processor operations
    - » Enabling/disabling interrupts, accessing privileged registers
- Physical memory
  - Contiguous, starting at address 0
- I/O devices
  - Virtualize devices like I/O, disks
  - Physical devices multiplexed by Disco
  - Special abstractions for SCSI disks and network interfaces
    - » Virtual disks for VMs
    - » Virtual subnet across all virtual machines

Disco Implementation

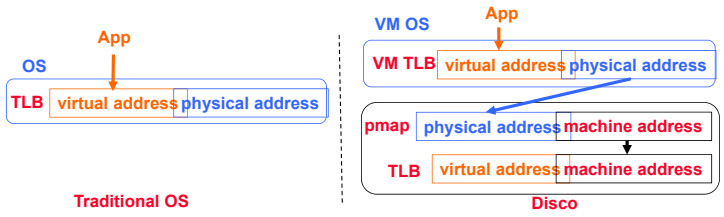
- Multi threaded shared memory program
- Attention to NUMA memory placement, cache aware data structures and IPC patterns
- Disco code copied to each flash processor
- Communicate using shared memory

Virtual CPUs

- Direct execution on real CPU:
  - Set real CPU registers to those of virtual CPU/Jump to current PC of VCPU
  - Privileged instructions must be emulated, since we won't run the OS in privileged mode
  - Disco runs privileged, OS runs supervisor mode, apps in user mode
- An OS privileged instruction causes a trap which causes Disco to emulated the intended instruction
- Maintains data structure for each virtual CPU for trap emulation
  - Trap examples: page faults, system calls, bus errors
- Scheduler multiplexes virtual CPU on real processor

Virtual Physical Memory

- Extra level of indirection: physical-to-machine address mappings
  - VM sees contiguous physical addresses starting from 0
  - Map physical addresses to the 40 bit machine addresses used by FLASH
  - When OS inserts a virtual-to-physical mapping into TLB, translates the physical address into corresponding machine address



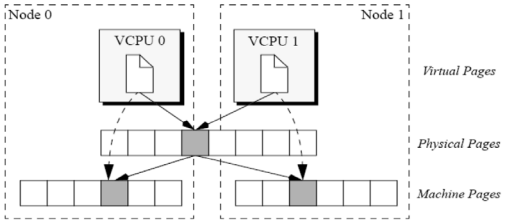
Virtual Physical Memory

- Extra level of indirection: physical-to-machine address mappings
  - VM sees contiguous physical addresses starting from 0
  - Map physical addresses to the 40 bit machine addresses used by FLASH
  - When OS inserts a virtual-to-physical mapping into TLB, translates the physical address into corresponding machine address
  - To quickly compute corrected TLB entry, Disco keeps a per VM pmap data structure that contains one entry per VM physical page
  - Each entry in TLB tagged with address space identifier (ASID) to avoid flushing TLB on MMU context switches (within VM)
- Must flush the real TLB on VM switch
- Somewhat slower:
  - OS now has TLB misses (not direct mapped)
  - TLB flushes are frequent
  - TLB instructions are now emulated
- Disco maintains a second-level cache of TLB entries:
  - This makes the VTLB seem larger than a regular R10000 TLB
  - Disco can absorb many TLB faults without passing them through to the real OS

NUMA Memory Management

- Dynamic Page Migration and Replication
  - Pages frequently accessed by one node are migrated
  - Read-shared pages are replicated among all nodes
  - Write-shared are not moved, since maintaining consistency requires remote access anyway
  - Migration and replacement policy is driven by cache-miss-counting facility provided by the FLASH hardware

Transparent Page Replication



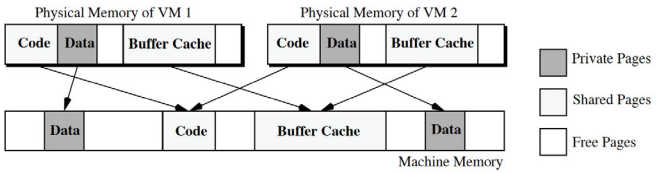
1. **Two different virtual processors of same virtual machine logically read-share same physical page, but each virtual processor accesses a local copy**
2. ***memmap* maintains an entry for each machine page that contains which virtual page reference it; used during TLB shootdown***

***Processors flushing their TLBs when another processor restrict access to a shared page.**

I/O (disk and network)

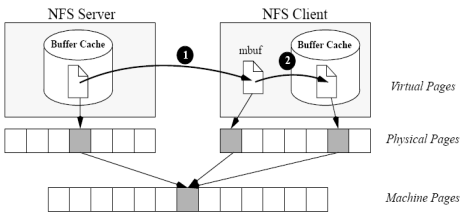
- Emulated all programmed I/O instructions
- Can also use special Disco-aware device drivers (simpler)
- Main task: translate all I/O instructions from using PM addresses to MM addresses
- Optimizations:
  - Larger TLB
  - Copy-on-write disk blocks
    - » Track which blocks already in memory
    - » When possible, reuse these pages by marking all versions read-only and using copy-on-write if they are modified
    - » => shared OS pages and shared executables can really be shared.
- Zero-copy networking along fake “subnet” that connect VMs within an SMP
  - Sender and receiver can use the same buffer (copy on write)

Transparent Page Sharing



- Global buffer cache that can be transparently shared between virtual machines

Transparent Page Sharing over NFS



1. The monitor's networking device remaps data page from source's machine address to destination's
2. The monitor remaps data page from driver's to client's buffer cache

Impact

- Revived VMs for the next 20 years
  - Now VMs are commodity
  - Every cloud provider, and virtually every enterprise uses VMs today
- VMWare successful commercialization of this work
  - Founded by authors of Disco in 1998, $6B revenue today
  - Initially targeted developers
  - Killer application: workload consolidation and infrastructure management in enterprises

Summary

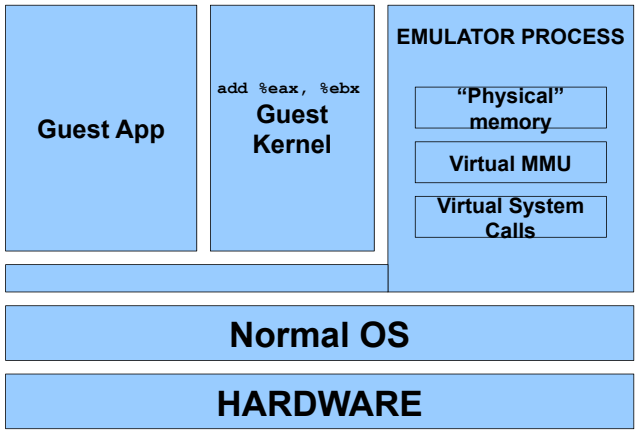
- Disco VMM hides NUMA-ness from non-NUMA aware OS
- Disco VMM is low effort
  - Only 13K LoC
- Moderate overhead due to virtualization
  - Only 16% overhead for uniprocessor workloads
  - System with eight virtual machines can run some workloads 40% faster

Is this a good paper?

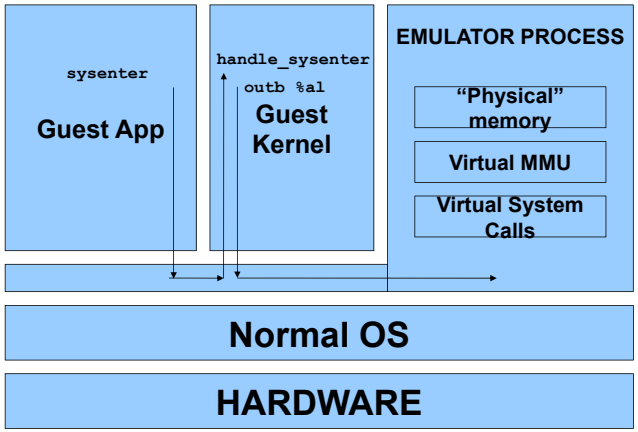
- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?

BREAK

How to Build a VMM 2: Trap and Emulate



How to Build a VMM 2: Trap and Emulate

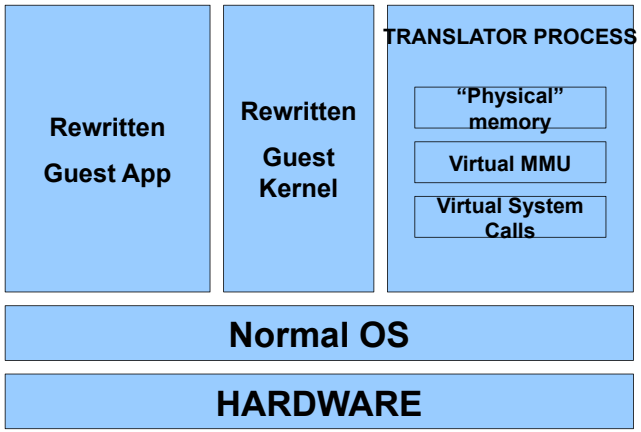


How to Build a VMM 2: Trap and Emulate

```
for(i = 0; i < 256; i++)
  mangle_pagetable_entry(&ptes[i]);
```

- 256 traps into the emulator
- Severe performance penalty

How to Build a VMM 3: Dynamic Binary Translation (VMware)



How to Build a VMM 3: Dynamic Binary Translation

```
for(i = 0; i < 256; i++)
  mangle_pagetable_entry(&ptes[i]);
```


How to Build a VMM 3: Dynamic Binary Translation

```
pte_t new_ptes[256];
for(i = 0; i < 256; i++)
  new_ptes[i] = mangled_entry(&ptes[i]);
register_new_ptes(new_ptes, 256);
```

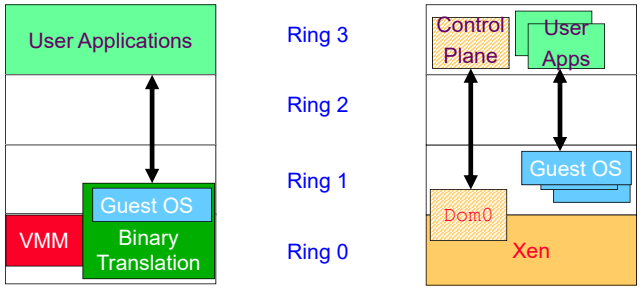
- But when is this a safe alteration?

How to Build a VMM 4: Paravirtualization (Xen)

Q. But when is this a safe alteration?

A. Let the humans worry about it

- Manually hack the OS: "paravirtualization"

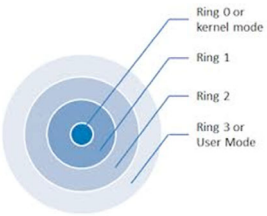


Full Virtualization

Paravirtualization

CPU

- X86 supports 4 privilege levels
  - 0 for OS, and 3 for applications
  - Xen downgrades OS to level 1
  - System-call and page-fault handlers registered to Xen
  - "fast handlers" for most exceptions, doesn't involve Xen



Xen: Founding Principles

- Key idea: Minimally alter guest OS to make VMs simpler and higher performance
  - Called paravirtualization (due to Denali project)
- Don't disguise multiplexing
- Execute faster than the competition
  - Note: VMWare does that too as "guest additions" are basically paravirtualization through specialized drivers (disk, I/O, video, …)

Xen: Emulate x86 (mostly)

- Xen paravirtualization:
  - Required less than 2% of the total lines of code to be modified
  - Pros: better performance on x86, some simplifications in VM implementation, OS might want to know that it is virtualized! (e.g. real time clocks)
  - Cons: must modify the guest OS (but not its applications!)
- Aims for performance isolation (why is this hard?)
- Philosophy:
  - Divide up resources and let each OS manage its own
  - Ensures that real costs are correctly accounted to each OS (essentially zero shared costs, e.g., no shared buffers, no shared network stack, etc.)

x86 Virtualization

- x86 harder to virtualize than Mips (as in Disco):
  - MMU uses hardware page tables
  - Some privileged instructions fail silently rather than fault
  - VMWare fixed this using binary rewrite
  - Xen by modifying the OS to avoid them
- Step 1: reduce the privilege of the OS
  - "Hypervisor" runs with full privilege instead (ring 0), OS runs in ring 1, Apps in ring 3
  - Xen must intercept interrupts and convert them to events posted to shared region with OS
  - Need both real and virtual time (and wall clock)

Virtualizing Virtual Memory

- Unlike MIPS, x86 does not have software TLB
- Good performance requires that all valid translations should be in HW page table
- TLB not "tagged", which means address space switch must flush TLB
  1) Map Xen into top 64MB in all address spaces (limit guest OS access) to avoid TLB flush
  2) Guest OS manages the hardware page table(s), but entries must be validated by Zen on updates; guest OS has read-only access to its own page table
- Page frame states:
  - PD=page directory, PT=page table, LDT=local descriptor table, GDT=global descriptor table, RW=writable page
  - The type system allows Xen to make sure that only validated pages are used for the HW page table
- Each guest OS gets a dedicated set of pages, although size can grow/shrink over time
- Physical page numbers (those used by the guest OS) can differ from the actual hardware numbers
  - Xen has a table to map HW➔Phys
  - Each guest OS has a Phy➔HW map
  - This enables the illusion of physically contiguous pages

Network

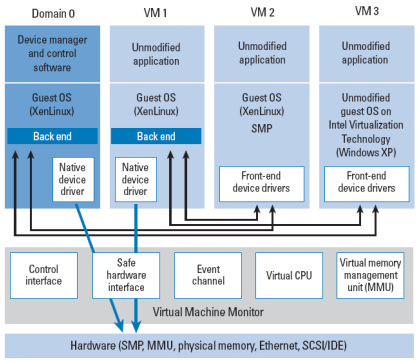
- Model:
  - Each guest OS has a virtual network interface connected to a virtual firewall/ router (VFR)
  - The VFR both limits the guest OS and also ensure correct incoming packet dispatch
- Exchange pages on packet receipt (to avoid copying)
  - No frame available ➔ dropped packet
- Rules enforce no IP spoofing by guest OS
- Bandwidth is round robin (is this "isolated"?)

Disk

- Virtual block devices (VBDs): similar to SCSI disks
- Management of partitions, etc. done via domain 0
- Could also use NFS or network-attached storage instead

Domain 0 (`dom0`)

- Nice idea: run the VMM management at user level
  - Given special access to control interface for platform management
  - Has back-end device drivers
- Much easier to debug a user-level process than an OS
- Narrow hypercall API and checks can catch potential errors



CPU Scheduling: Borrowed Virtual Time Scheduling

- Fair sharing scheduler: effective isolation between domains
- However, allows temporary violations of fair sharing to favor recently-woken domains
  - Reduce wake-up latency → improves interactivity

Times and Timers

- Times:
  - Real time since machine boot: always advances regardless of the executing domain
  - Virtual time: time that only advances within the context of the domain
  - Wall-clock time
- Each guest OS can program timers for both:
  - Real time
  - Virtual time

Control Transfer: Hypercalls and Events

- Hypercalls: synchronous calls from a domain to Xen
  - Allows domains to perform privileged operation via software traps
  - Similar to system calls
- Events: asynchronous notifications from Xen to domains
  - Replace device interrupts

Exceptions

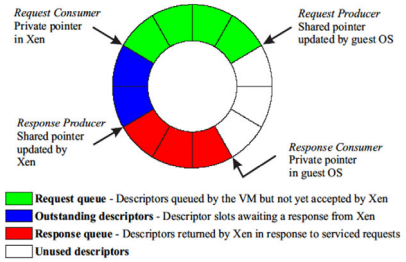
- Memory faults and software traps
  - Virtualized through Xen's event handler

Memory

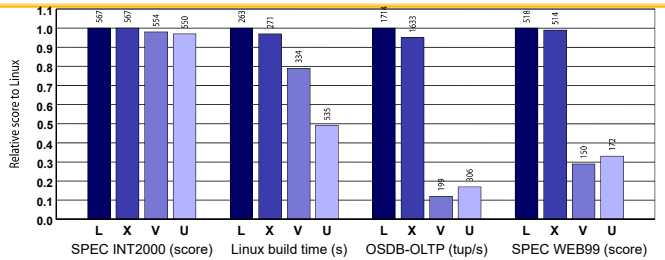
- Physical memory
  - Reserved at domain creation time
  - Statically partitioned among domains
- Virtual memory:
  - OS allocates a page table from its own reservation and registers it with Xen
  - OS gives up all direct privileges on page table to Xen
  - All subsequent updates to page table must be validated by Xen
  - Guest OS typically batch updates to amortize hypervisor calls

Data Transfer: I/O Rings

- Zero-copy semantics
- Data is transferred to/from domains via Xen through a buffer ring



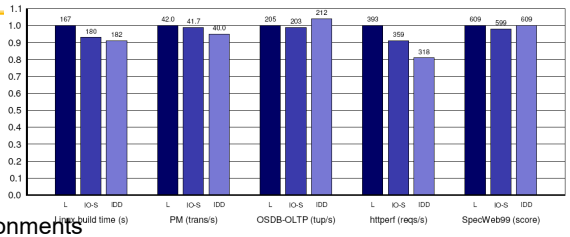
Benchmark Performance



Benchmark suite running on Linux (L), Xen (X), VMware Workstation (V), and UML (U)

- Benchmarks
  - Spec INT200: compute intensive workload
  - Linux build time: extensive file I/O, scheduling, memory management
  - OSBD-OLTP: transaction processing workload, extensive synchronous disk I/O
  - Spec WEB99: web-like workload (file and network traffic)
- Fair and reasonable comparisons?

I/O Performance



- Environments
  - L: Linux
  - IO-S: Xen using IO-Space access
  - IDD: Xen using isolated device driver
- Benchmarks
  - Linux build time: file I/O, scheduling, memory management
  - PM: file system benchmark
  - OSDB-OLTP: transaction processing workload, extensive synchronous disk I/O
  - httperf: static document retrieval
  - SpecWeb99: web-like workload (file and network traffic)

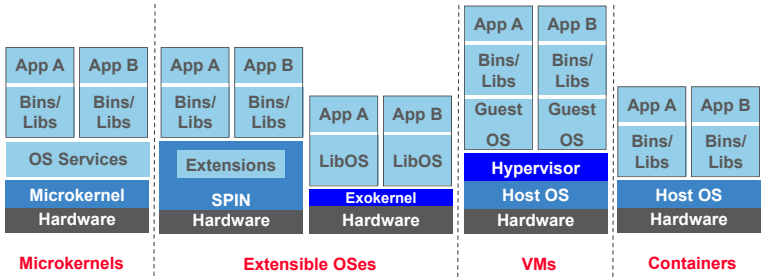
Xen Summary

- Performance overhead of only 2-5%
- Available as open source but owned by Citrix since 2007
  - Modified version of Xen powers Amazon EC2
  - Widely used by web hosting companies
- Many security benefits
  - Multiplexes physical resources with performance isolation across OS instances
  - Hypervisor can isolate/contain OS security vulnerabilities
  - Hypervisor has smaller attack surface
    - » Simpler API than OS – narrow interfaces ➔ tractable security
    - » Less overall code than an OS
- BUT hypervisor vulnerabilities compromise everything…

History and Impact

- Released in 2003 (Cambridge University)
  - Authors founded XenSource, acquired by Citrix for $500M in 2007
- Widely used today:
  - Linux supports dom0 in Linux's mainline kernel
  - AWS EC2 based on Xen

Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?

Discussion: OSes, VMs, Containers