Virtual Machine Monitors: Disco and Xen

Robert Grimm
New York University
The Three Questions

- What is the problem?
- What is new or different?
- What are the contributions and limitations?
Background: ccNUMA

- Cache-coherent non-uniform memory architecture
  - Multi-processor with high-performance interconnect
- Non-uniform memory
  - Global address space
  - But memory distributed amongst processing elements
- Cache-coherence
  - How to ensure consistency between processor caches?
  - Solutions: Bus snooping, directory
- Targeted system: FLASH, Stanford's own ccNUMA
The Challenge

- Commodity OS's not well-suited for ccNUMA
  - Do not scale
    - Lock contention, memory architecture
  - Do not isolate/contain faults
    - More processors → more failures
- Customized operating systems
  - Take time to build, lag hardware
  - Cost a lot of money
Add a virtual machine monitor (VMM)

- Commodity OS's run in their own virtual machines (VMs)
  - Communicate through distributed protocols
- VMM uses global policies to manage resources
  - Moves memory between VMs to avoid paging
  - Schedules virtual processors to balance load
Virtual Machine Challenges

- Overheads
  - Instruction execution, exception processing, I/O
  - Memory
    - Code and data of hosted operating systems
    - Replicated buffer caches
- Resource management
  - Lack of information
    - Idle loop, lock busy-waiting
  - Page usage
- Communication and sharing
  - Not really a problem anymore b/c of distributed protocols
**Interface**

- MIPS R10000 processor
  - All instructions, the MMU, trap architecture
  - Memory-based interface to VMM
    - Enabling/disabling interrupts, accessing privileged registers
- Physical memory
  - Contiguous, starting at address 0
- I/O devices
  - Virtual devices exclusive to VM
  - Physical devices multiplexed by Disco
  - Idealized interface to SCSI disks and network
    - Virtual subnet across all virtual machines
Virtual CPUs

- **Three modes**
  - Kernel mode: Disco
    - Provides full access to hardware
  - Supervisor mode: Guest operating system
    - Provides access to protected memory segment
  - User mode: Applications
    - Emulated in VMM with per VM data structures (registers, TLB)
  - Syscalls, page faults handled by guest OS's trap handlers

- **Emulation by direct execution**
  - Not for privileged instructions, access to physical memory, and I/O devices
Virtual Physical Memory

- Adds level of translation: physical-to-machine
  - Performed in software-reloaded TLB
  - Based on pmap data structure: entry per physical page
- Requires changes in Irix memory layout
- Flushes TLB when scheduling different virtual CPUs
  - MIPS TLB is tagged with address space ID
- Increases number of misses, but adds software TLB
  - Guest operating system now mapped through TLB
  - TLB is flushed on virtual CPU switches
- Virtualization introduces overhead
NUMA Memory Management

- Two optimizations
  - Heavily accessed pages relocated to using node
  - Read-only shared pages replicated across nodes
  - Based on cache miss counting facility of FLASH
  - Supported by memmap data structure
    - For each machine page, points to physical addr's, VMs, copies
Virtual I/O Devices

- Specialized interface for common devices
  - Special drivers for guest OS's: one trap per operation
- DMA requests are modified
  - From physical to machine memory
- Copy-on-write disks
  - Page with same contents requires only one copy
Virtual Network Interface

* Issue: Different VMs communicate through standard distributed protocols (here, NFS)
  * May lead to page duplication in memory

* Solution: virtual subnet
  * Ethernet-like addresses, no maximum transfer unit
  * Read-only mapping instead of copying
  * Supports scatter/gather

* What about NUMA?
Irix 5.3 on Disco

- Changed memory layout to make all pages mapped
- Added device drivers to special I/O devices
  - Disco's drivers are the same as those in Irix
- Patched HAL to use memory loads/stores instead of privileged instructions
- Added new calls
  - Request zeroed-out memory pages, inform about freed page
- Changed mbuf management to be page-aligned
- Changed bcopy to use remap (with copy-on-write)
Disco Evaluation
Experimental Methodology

- FLASH machine "unfortunately not yet available"
- Use SimOS
  - Models hardware in enough detail to run unmodified code
  - Supports different levels of accuracy, checkpoint/restore
- Workloads
  - pmake, engineering, scientific computing, database
Uniprocessor configuration comparing Irix and Disco

- Disco overhead between 3% and 16% (!)
- Mostly due to trap emulation and TLB reload misses
### What does this table tell us?

- What is the problem with entering/exiting the kernel?
- What is the problem with placing OS in mapped memory?
Memory Overhead

- Workload: 8 instances of pmake
- Memory partitioned across virtual machines
- NFS configuration uses more memory than available
**Scalability**

- Irix: High synchronization and memory overheads
  - memlock: spinlock for memory management data structures
- Disco: Partitioning reduces overheads
- What about RADIX experiment?
Page Migration and Replication

What does this figure tell us?
Xen in (Some) Detail
How does this differ from Disco?
Xen by Comparison

Three main differences

- Less complete virtualization
- *Domain0* to initialize/manage VMs, incl. to set policies
- Strong performance isolation

Primary contribution

- Interface that is pretty close to hardware *and* enables low-overhead virtualization
- Need to change more OS code than in Disco
- Yet still not too much: 3000 lines for Linux
CPU Interface

- Xen runs in ring 0
- Guest OS's run in the otherwise unused ring 1
- Privileged instructions need to be processed by Xen
  - Though, x86 makes life a little difficult — how?
- Fast exception handlers do *not* require Xen interaction
  - Must execute outside ring 0 — what does Xen need to do?
Memory Interface

- x86 has hardware-accessed page tables and TLB
- Guest OS's responsible for managing page tables
  - Provide machine memory to Xen (from guest's reservation)
  - Have direct read-only access
  - Defer to Xen for performing (batched) updates
Device Driver Interface

- Shockingly, idealized hardware abstraction
  - Virtual firewall-router
  - Virtual network interfaces
  - Virtual block devices
VM/VMM Communication

- Control transfer
  - Hypercalls: synchronous software traps to VMM
  - Events: asynchronous, possibly batched upcalls to VMs
- Data transfer through I/O rings
  - Separate descriptors from actual data
  - Zero-copy transfer during device I/O
  - Support batching and re-ordering

![Diagram](image-url)

- **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

Legend:
- **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**
Memory Management

- Virtual memory
  - Guest OS's manage page tables (not shadows)
    - Expose names and allocation
  - Validated by types and reference counts
    - Page directory/table, local/global descriptor table, writable
  - Page directory and tables pinned
    - I.e., they cannot be swapped — why?

- Physical memory
  - Controlled through balloon driver in guest OS
    - Requests and pins pages, which are then returned to VMM
  - May be mapped into hardware memory — why?
    - Xen publishes machine-to-physical mapping
Performance Isolation

- Four "domains"
  - PostgreSQL
  - SPECweb99
  - Disk bandwidth hog (sustained `dd`)
  - Fork bomb

- Results: Xen sees only 2-4% loss, Linux locks up

- What is the key ingredient?
What Do You Think?