Virtual Machine Monitors

Mendel Rosenblum

Associate Professor of Computer Science
Stanford University
and
Co-Founder and Chief Scientist
VMware Inc.

October 2001
Talk Goal

- Understand the capabilities of Virtual Machine Monitors
  - What they can (and can’t) do.

- Observation:
  Virtual machine monitor
  + Smart person with problem
    Solution using virtual machine monitors
Talk Outline

- What is a virtual machine monitor (VMM)?
  - IBM mainframes
  - VMware Inc. – Commodity x86 computers

- How do they work?
  - A quick look under the hood.

- VMM characteristics, attributes, and capabilities
  - Compatibility, performance, isolation, encapsulation

- Some applications of VMMs to increase reliability

- Detailed examples

- Conclusion

October 2001
Virtual Machine Monitor

- Thin software layer that virtualizes the hardware
  Exports abstraction of virtual machines
Old idea from the 1960s

- IBM VM/370 – A VMM for IBM mainframe
  - Multiplex multiple OS environments on expensive hardware.
  - Desirable when few machine around.

- Popular research idea in 1960s and 1970s
  - Entire conferences on virtual machine monitor

- Interest died out in the 1980s and 1990s.
  - Hardware got cheap.

- Why did we think this was interesting again?
  - Difference problems today – complex software
  - VMM attributes still relevant

October 2001
Virtual Machine Monitor Implementation

- Need to virtualize all the hardware of the machine
  - CPU, Memory, I/O
- Need to use hardware to directly support virtual machines
  - Virtual CPU execution directly uses real CPU
  - Memory of virtual machine uses real memory
  - I/O devices supported using real I/O devices
- Definition: virtualizability
  - Hardware is virtualizable if a virtual machine monitor can control it enough to directly support a virtual machine.
- Use to be a standard part of architecture curriculum
  - Not even mentioned in standard textbooks
  - Most architectures today are not strictly virtualizable (x86, RISC)

October 2001
CPU virtualization

- Monitor needs to **safely** give CPU to virtual machine
  - Virtual machine should be able to damage monitor or other VMs
  - Monitor should be able to get control and switch running VM
    - *Monitor needs to multiplex virtual CPUs on real CPUs*
      - Like an OS context switch: Save one state/Restore another
- Basic trick: Run monitor in most privileged mode
  - VMs always run in less privileged modes.
    - *For speed, use direct execution to run VMs*
    - *For protection, use protection mechanisms of CPU*
  - Monitor needs to hide and fake-out software to run correctly.
    - *VMs traps into the monitor to do any privileged operations.*
- Done right, software can’t tell virtual CPUs from real CPU

October 2001
CPU virtualization requirements

- Need protection levels to run VMs and monitors
- All unsafe/privileged operations should trap
  - Example: disable interrupt, access I/O dev, …
  - x86 problem: POPF (different semantics in different rings)
- Privilege level should not be visible to software
  - Software in VM should be able to query and find its in a VM
  - x86 problem: MOV ax, cs
- Trap should be transparent to software in VM
  - Software in VM should be able to tell if instruction trapped.
  - x86 problem: traps can destroy machine state.
Physical memory virtualization

- Need to implement a *virtual* physical memory
  - Logically need additional level of indirection
    - VM’s VA -> VM’s PA -> machine address
  - Can be folded into page tables: VA->machine address

- Trick: Monitor keeps shadow of VM’s page table
  - Contains mapping to physical memory allocated for that VM
  - Monitor can demand page the virtual machine

- Again, uses hardware protection
  - Monitor never maps itself into VM’s page table
  - Monitor never maps memory allocated to other VMs in VM’s page table

- Need a place to load monitor in memory
I/O device virtualization

- Need to either emulate or map through I/O device
  - Example map through: virtual disk == physical disk
  - Example emulate: virtual disk == blocks on physical disk
- Monitor needs to interpose on all I/O device request
  - Intercept or redirect all program input/output
  - Relocate all DMA requests
- Hard to do for an arbitrary I/O device
  - How do you know what’s an DMA address?
- Multiplex devices: Keyboard, mouse, video, ..
Virtual machine monitor attributes

- **Software compatibility**
  - Runs pretty much all software

- **Low overheads/High performance**
  - Near “raw” machine performance

- **Complete isolation**
  - Total data isolation between virtual machines

- **Encapsulation**
  - Virtual machines are not tied to physical machines

October 2001
Software compatibility

• Key: Make virtual machine abstraction match real HW
  • All software that runs on real HW runs in VM
  • Compare to abstract virtual machines such as Java

• Example: VMware runs
  DOS, Win 3.1, 95, 98, NT, 2000, ME, XP; Linux, OS/2, Freebsd, etc.

• Easiest way to maintain backward compatibility
  • Avoid death to due cost of backward compatibility

• What doesn’t run in VMM:
  • Super timing sensitive:
    • e.g. syscall trap should take exactly 60 cycles
  • Hardware I/O device dependences:
    • Code assumes it can talk to latest nvidia graphics processors

• **Everything** else runs regardless of SW complexity!

October 2001
Low overhead/High Performance

• Key: Configure HW to directly run Virtual Machines
  • Use CPU to emulate a virtual CPU
  • Use real physical memory to emulate virtual physical memory
  • Emulate a disk with a disk, etc.

• Trick from 1960s:
  • Configure hardware to safely give it to virtual machine
  • VMM gets control on any privileged operation

• Virtual machine runs within a few percent of native
Isolation capability

• **Key:** Use HW protection mechanisms to isolate VMs
  • Example: Protection rings, MMU protection bits
  • Unbreakable security

• **Complete isolation**
  • Code running in a VM can’t read, modify, break, etc:
    - *Other virtual machines*
    - *The virtual machine monitor*

• **Isolation comparable to separate physical machines**
  • Handle accidents (e.g. software bugs)
  • Malicious attacks (e.g. hackers)
Encapsulation

- **Key: VMM sits between VM and hardware**
  - Virtual machine is not tied to physical hardware

- **VMM can completely isolate VM from hardware:**
  - Support for checkpoint/restore operations
  - Virtual machine migration
  - Undo execution

- **Hardware independence**
  - VMM maps “standard” virtual HW to real HW
    - *VMM provides the “conversion” layer*
VMMs capabilities

- An application of an level of indirection
  - Between software and hardware
  - Transparently interpose on all communication:
    - CPU, memory, I/O devices (disk, net, etc.)

- Standard argument:
  - Additional overheads
  + Glorious benefits

- Simple examples:
  - Resource management
  - OS enhancement
Resource management

- Key: VMM controls all HW: CPU, Memory, I/O devices
  - VMM can decide how much of each each VM gets

- Can overcommit resources
  - More virtual CPUs than physical CPUs
  - More virtual physical memory than physical memory

- VMM can do performance isolation
  - Make guarantees to the VMs
  - Limit the resources used by a VM

October 2001
OS enhancements

• Key: VMM layer can enhance the OS running in the VM
  • Some functionality easier to do in VMM than OS

• Example: Stanford Disco VMM
  • Run OSes on scalable multiprocessors

• Examples from VMware product:
  • Check & fast restore capabilities
  • Undoable disks

• Many more
  • Internet Edge deployment
  • Application Service Provider environment
  • Software fault tolerance

• Sometimes faster and easiest way of improvement.
Virtual Machines Vision

- All software runs in a virtual machine.
- All hardware runs a virtual machine monitor.

![Diagram showing hardware configuration and virtual machine monitors (VMMs) for Win2kVM, Linux VM, and WinXP VM.]

Configuration:
- vDisk#1 backed by Disk#5
- vDisk#2 backed by remote Disk#1
- vNet#1 attached to VLAN#4

The Matrix analogy

October 2001
Examples of using VMs for reliability

• **Lower cost replication**
  • Replicate virtual machines not physical machines

• **Faster recovery**
  • Use restore to skip boot, etc.

• **Partitioning for reliability**
  • Fault containment, reduce concurrency demands on OS

October 2001
Low cost replication example#1

- Assumption: Hardware doesn’t fail, but software does
  - Replicate software not hardware

October 2001
Low cost replication #2

- Assumption: Unlikely that more than one fail at a time

October 2001
Partitioning for reliability

- Assumption: An OS doing one thing is more stable than OS doing more than one thing.
Detailed Examples

- Moving content to the network edge
- Computing utility