Introduction to Virtual Machines

Kartik Gopalan

From
“HSSV”, Bugnion, Neih, Tsafrir
“Virtual Machines”, Smith and Nair, Chapter 1
Also, Chapter 7 Andrew Tanenbaum’s book
“Virtual machines have finally arrived. Dismissed for a number of years as merely academic curiosities, they are now seen as cost-effective techniques for organizing computer systems resources to provide extraordinary system flexibility and support for certain unique applications”.
The Rise and Fall of Virtual Machines

• 1970s:
  • IBM mainframes run VMs and hardware supports not just single-level, but also nested virtualization.
  • Hundreds of papers on virtualization.
  • Goldberg and Popek’s seminal paper on formal requirements of virtualization support.

• 1980s and 1990s:
  • Personal computers (temporarily) obviate the need for VMs.
  • x86. MIPS, Sparc processors have no hardware support for virtualization
  • UNIX, Windows, Linux don’t include virtualization support.
VMs — Back from Dead (or the Second Coming)

• 1997: DISCO paper revisits for running commodity OS on SMP machines
• 1999: VMWare releases VMWare workstation for x86 CPUs.
• Early 2000s:
  • VMWare ESX server, VirtualPC, Xen, Denali, Cells.
  • Still no hardware support for virtualization
• Late 2000s
  • AMD-V, Intel VT-x, VT-d etc.
  • Turtles paper: Nested Virtualization revisited
• 2010s:
  • Multi-billion $ industry.
  • All cloud platforms are virtualized.

• But wait…the cycle may be turning back.
  • Bare-metal clouds
  • “Say Goodbye to Virtualization for a Safer Cloud”, HotCloud 2018
Definitions

- **Virtualization** is the application of the layering principle through enforced modularity, [whereby the exposed virtual resource is identical to the underlying physical resource being virtualized.]

- A **virtual machine** is an abstraction of a complete compute environment through the combined virtualization of the processor, memory, and I/O components of a computer.

- The **hypervisor** is a specialized piece of system software that manages and runs virtual machines.

- The **virtual machine monitor (VMM)** refers to the portion of the hypervisor that focuses on the CPU and memory virtualization.
Forms of Virtualization

Note that the three are not mutually exclusive.

- Many-to-many virtualization
- Emulating many I/O devices on a single physical disk
Virtual Machines

- Logical/Emulated representations of full computing system environment
  - CPU + memory + I/O
  - Implemented by adding layers of software to the real machine to support the desired VM architecture.

- Uses:
  - Multiple OSes on one machine, including legacy OSes
  - Isolation
  - Enhanced security
  - Live migration of servers
  - Virtual environment for testing and development
  - Platform emulation
  - On-the-fly optimization
  - Realizing ISAs not found in physical machines
Interfaces of a computer system

- User ISA: 7
- System ISA: 8
- Syscalls: 3
- ABI: 3, 7
- API: 2, 7
Two Types of VMs

- **Process VM**
  - Virtualizes the ABI
  - Virtualization software = Runtime
    - Runs in non-privileged mode (user space)
    - Performs binary translation.
  - Terminates when guest process terminates.

- **System VM**
  - Virtualizes the ISA
  - Virtualization software = Hypervisor
    - Runs in privileged mode
    - Traps and emulates privileged instructions
  - Lasts as long as physical host is alive
Process Virtual Machines

Process in a multiprogramming OS
- Standard OS syscall interface + instruction set
- Multiple processes, each with its own address space and virtual machine view.

Emulators
- Support one ISA on hardware designed for another ISA
- Interpreter:
  - Fetches, decodes and emulates individual instructions. Slow.
- Dynamic Binary Translator:
  - Blocks of source instructions converted to target instructions.
  - Translated blocks cached to exploit locality.

Same ISA Binary Optimizers
- Optimize code on the fly
- Same as emulators except source and target ISAs are the same.

Language-based VMs
- Virtual ISA (bytecode) designed for platform independence
- Platform-dependent VM executes virtual ISA
- E.g. Sun’s JVM and Microsoft’s CLI (part of .NET)
- Both are stack-based VMs that run on register-based m/c.
System Virtual Machines

(focus of this lecture)
Hypervisor

- Also called Virtual Machine Monitor (VMM)

A hypervisor is an operating system for operating systems
- Provides a virtual execution environment for an entire OS and its applications
- Controls access to hardware resources
- When guest OS executes a privileged instruction, Hypervisor intercepts the instruction, checks for correctness and emulates the instruction.

<table>
<thead>
<tr>
<th>Apps</th>
<th>Apps</th>
<th>Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest OS 1</td>
<td>Guest OS 2</td>
<td>Guest OS 3</td>
</tr>
</tbody>
</table>

Hypervisor

Hardware
Type 1 Hypervisors (Classical System VMs)

- Hypervisor executes natively on the host ISA
- Hypervisor directly controls hardware and provides all device drivers
- Hypervisor emulates sensitive instructions executed by the Guest OS
- E.g. KVM and VMWare ESX Server
Type-2 Hypervisors (Hosted VMs)

- A host OS controls the hardware
- The Hypervisor runs partly in process space and partly in the host kernel
- Hypervisor Relies on host OS to provide drivers
- E.g. VMWare Desktop Client
Para-virtualized VMs

- Modify guest OS for better performance

- Traditional Hypervisors provide full-virtualization
  - They expose to VMs virtual hardware that is functionally identical to the underlying physical hardware.
  - Advantage: allows unmodified guest OS to execute
  - Disadvantage: Sensitive instructions must be trapped and emulated by Hypervisor.
  - E.g. KVM and VMWare ESX provide full virtualization

- Para-virtualized VM
  - Sees a virtual hardware abstraction that is similar, but not identical to the real hardware.
  - Guest OS is modified to replace sensitive instructions with “hypercalls” to the Hypervisor.
  - Advantage: Results in lower performance overhead
  - Disadvantage: Needs modification to the guest OS.
  - E.g. Xen provides both para-virtual as well as full-virtualization

- Often traditional Hypervisors are partially para-virtualized
  - Device drivers in guest OS may be para-virtualized whereas CPU and Memory may be fully virtualized.
Whole System VMs: Emulation

- Host and Guest ISA are different
- So emulation is required
- Hosted VM + emulation
- E.g. Virtual PC (Windows on MAC)
Co-designed VMs

- The hypervisor is designed closely with (and possibly built into) a specific type of hardware ISA (or native ISA).

- **Goal:** Performance improvement of existing ISA (or guest ISA) during runtime.

- Hypervisor performs Emulation from Guest ISA to Native ISA.

- E.g. Transmeta Crusoe
  - Native ISA based on VLIW
  - Guest ISA = x86
  - Goal power savings
Taxonomy

Process VMs
- same ISA
  - Multi programmed Systems
  - Dynamic Optimizers
- different ISA
  - Dynamic Translators
  - HLL VMs

System VMs
- same ISA
  - Classic OS VMs
  - Hosted VMs
- different ISA
  - Whole System VMs
  - Co-Designed VMs

Para-virtualized VMs
Hardware Virtual Machines
Versatility

Java App

JVM

Linux IA-32

VMWare

Windows IA-32

Code Morphing

Crusoe VLIW
What can you do with system VMs?

- Emulation: Mix-and-match cross-platform portability
- Optimization: Usually done with emulation for platform-specific performance improvement
- Replication: Multiple VMs on single platform
- Composition: form more complex flexible systems
Virtualizing individual resources in System VMs
CPU Virtualization for VMs

- Each VM sees a set of “virtual CPUs”
- Hypervisors must emulate privileged instructions issued by guest OS.
- Modern ISAs provide special interfaces for Hypervisors to run VMs
  - Intel provides the VTx interface
  - AMD provides the AMD-v interface

- These special ISA interfaces allow the Hypervisors to efficiently emulate privileged instructions executed by the guest OS.

- When guest OS executes a privileged instruction
  - Hardware traps the instruction to the hypervisor
  - Hypervisor checks whether instruction must be emulated.
  - If so, Hypervisor reproduces the effect of the privileged operation.
Execution of Privileged Instruction by Guest

Instruction trap occurs

Dispatcher

- Privileged Instruction
- Interpreter Routine 1
- Interpreter Routine 2
- ... (multiple instances)
- Interpreter Routine n

Allocator

These instructions desire to change machine resources, e.g., load relocation bounds register

Guest OS code in VM (user mode)

- Privileged instruction (LPSW)
- Next instruction (target of LPSW)

VMM code (privileged mode)

- Dispatcher
- LPSW Routine:
  - Change mode to privileged
  - Check privilege level in VM
  - Emulate instruction
  - Compute target
  - Restore mode to user
  - Jump to target

These instructions do not change machine resources but access privileged resources, e.g., IN, OUT, Write TLB
Resource Control

- Issue: How to retain control of resources in the Hypervisor?
- Timer interval control performed by Hypervisor
- Also, guest OS is not allowed to read the timer value
  - Guest OS sees a virtual interval timer
- Hypervisor also gains control whenever guest OS executes privileged instructions.
Memory Virtualization for VMs

**Traditional virtual memory**

- Virtual Address Space
- Page Table
- Physical Address Space

**Virtual memory for VMs**

- Virtual Address Space
- First-level Page Table
- Guest Physical Address Space
- Second-level Page Table
- Physical Address Space
- Shadow Page Table (optional)

- Guest OS in each VM sees a “guest”-physical address (GPA) space instead of the physical addresses
- Often hardware supports two-level page tables
  - EPT in Intel VT-x and NPT in AMD-v
- When hardware doesn’t, then Hypervisor needs to emulate two-level page tables using “shadow page tables”.
I/O Virtualization for VMs

• Hypervisor provides a virtual version of each physical device
• I/O activity directed at the virtual device is trapped by Hypervisor and converted to equivalent request for the physical device.
• Options:
  • Device emulation
    • Hypervisor traps and emulates each I/O instruction from Guest in Hypervisor.
    • Very slow.
    • Difficult to emulate the effect of combinations of I/O instructions.
  • Para-virtual devices
    • Special device drivers inserted in guest OS to talk to Hypervisor.
    • Most common.
  • Direct device access
    • Allow the VM to directly access physical device.
    • Fastest option but not scalable.
    • Requires IOMMU and VT-d support from hardware.