Types of Virtualization

- **Emulation**
  - VM emulates/simulates complete hardware
  - Unmodified guest OS for a different PC can be run
    - Bochs, VirtualPC for Mac, QEMU
- **Full/native Virtualization**
  - VM simulates “enough” hardware to allow an unmodified guest OS to be run in isolation
    - Same hardware CPU
    - IBM VM family, VMWare Workstation, Parallels,…

Types of virtualization

- **Para-virtualization**
  - VM does not simulate hardware
  - Use special API that a modified guest OS must use
  - Hypercalls trapped by the Hypervisor and serviced
    - Xen, VMWare ESX Server
- **OS-level virtualization**
  - OS allows multiple secure virtual servers to be run
  - Guest OS is the same as the host OS, but appears isolated
    - apps see an isolated OS
  - Solaris Containers, BSD Jails, Linux Vserver
- **Application level virtualization**
  - Application is gives its own copy of components that are not shared
    - (E.g., own registry files, global objects) - VE prevents conflicts
    - JVM
**Type 1 hypervisor**

- Unmodified OS is running in user mode (or ring 1)
  - But it thinks it is running in kernel mode (virtual kernel mode)
  - privileged instructions trap; sensitive inst-> use VT to trap
  - Hypervisor is the “real kernel”
    - Upon trap, executes privileged operations
    - Or emulates what the hardware would do

**Type 2 Hypervisor**

- VMWare example
  - Upon loading program: scans code for basic blocks
  - If sensitive instructions, replace by Vmware procedure
    - Binary translation
    - Cache modified basic block in VMWare cache
    - Execute; load next basic block etc.

- Type 2 hypervisors work without VT support
  - Sensitive instructions replaced by procedures that emulate them.
Paravirtualization

- Both type 1 and 2 hypervisors work on unmodified OS
- Paravirtualization: modify OS kernel to replace all sensitive instructions with hypercalls
  - OS behaves like a user program making system calls
  - Hypervisor executes the privileged operation invoked by hypercall.

Virtual machine Interface

- Standardize the VM interface so kernel can run on bare hardware or any hypervisor
Memory virtualization

- OS manages page tables
  - Create new pagetable is sensitive -> traps to hypervisor
- hypervisor manages multiple OS
  - Need a second shadow page table
  - OS: VM virtual pages to VM’s physical pages
  - Hypervisor maps to actual page in shadow page table
  - Two level mapping
  - Need to catch changes to page table (not privileged)
    - Change PT to read-only - page fault
    - Paravirtualized - use hypercalls to inform

I/O Virtualization

- Virtualize I/O devices:
  - Network interface card, disk
- Create virtual interfaces that are multiplexed onto a physical interface
  - Network: multiple virtual NICs multiplexed onto a physical Nic
  - Disk: each VM has its own partition; hypervisor translates I/O requests to actual disk blocks
- Type 2 hypervisor: use host OS device drivers
- Type 1: implement drivers or use a special VM (dom-0)
Server Design Issues

- Server Design
  - Iterative versus concurrent
- How to locate an end-point (port #)?
  - Well known port #
  - Directory service (port mapper in Unix)
  - Super server (inetd in Unix)

Stateful or Stateless?

- Stateful server
  - Maintain state of connected clients
  - Sessions in web servers
- Stateless server
  - No state for clients
- Soft state
  - Maintain state for a limited time; discarding state does not impact correctness
Scaling via Server Clusters

- Web applications use tiered architecture
  - Each tier may be optionally replicated; uses a dispatcher
  - Use TCP splicing or handoffs

Case Study: PlanetLab

- Distributed cluster across universities
  - Used for experimental research by students and faculty in networking and distributed systems
- Uses a virtualized architecture
  - Linux Vservers
  - Node manager per machine
  - Obtain a “slice” for an experiment: slice creation service
Code and Process Migration

- Motivation
- How does migration occur?
- Resource migration
- Agent-based system
- Details of process migration

Motivation

- Key reasons: performance and flexibility
- Process migration (aka strong mobility)
  - Improved system-wide performance – better utilization of system-wide resources
  - Examples: Condor, DQS
- Code migration (aka weak mobility)
  - Shipment of server code to client – filling forms (reduce communication, no need to pre-link stubs with client)
  - Ship parts of client application to server instead of data from server to client (e.g., databases)
  - Improve parallelism – agent-based web searches
Motivation

- Flexibility
  - Dynamic configuration of distributed system
  - Clients don’t need preinstalled software – download on demand

Migration models

- Process = code seg + resource seg + execution seg
- Weak versus strong mobility
  - Weak => transferred program starts from initial state
- Sender-initiated versus receiver-initiated
- Sender-initiated (code is with sender)
  - Client sending a query to database server
  - Client should be pre-registered
- Receiver-initiated
  - Java applets
  - Receiver can be anonymous
Who executes migrated entity?

- Code migration:
  - Execute in a separate process
  - [Applets] Execute in target process
- Process migration
  - Remote cloning
  - Migrate the process

Models for Code Migration
Do Resources Migrate?

- Depends on resource to process binding
  - By identifier: specific web site, ftp server
  - By value: Java libraries
  - By type: printers, local devices
- Depends on type of “attachments”
  - Unattached to any node: data files
  - Fastened resources (can be moved only at high cost)
    - Database, web sites
  - Fixed resources
    - Local devices, communication end points

Resource Migration Actions

- Actions to be taken with respect to the references to local resources when migrating code to another machine.
- GR: establish global system-wide reference
- MV: move the resources
- CP: copy the resource
- RB: rebind process to locally available resource
Migration in Heterogeneous Systems

- Systems can be heterogeneous (different architecture, OS)
  - Support only weak mobility: recompile code, no run time information
  - Strong mobility: recompile code segment, transfer execution segment [migration stack]
  - Virtual machines - interpret source (scripts) or intermediate code [Java]

Case study: Agents/Worms

- Software agents
  - Autonomous process capable of reacting to, and initiating changes in its environment, possibly in collaboration
  - More than a “process” – can act on its own

- Mobile agent
  - Capability to move between machines
  - Needs support for strong mobility
  - Example: D’Agents (aka Agent TCL)
    - Support for heterogeneous systems, uses interpreted languages
Case study: Condor

- Condor: use idle cycles on workstations in a LAN
- Used to run large batch jobs, long simulations
- Idle machines contact condor for work
- Condor assigns a waiting job
- User returns to workstation => suspend job, migrate
- Flexible job scheduling policies

Case Study: VM Migration

- “Process migration” in the modern era: VM migration
- Migrate the entire VM (OS + all resident processes)
  - Network connections also migrate since OS moves
- Live migration supported: migrate while apps are running
  - Near-zero downtimes
  - IP address of VM does not change (provided migration within LAN)
- VMWare, Xen support VM migration
Case Study: ISOS

- Internet scale operating system
  - Harness compute cycles of thousands of PCs on the Internet
  - PCs owned by different individuals
  - Donate CPU cycles/storage when not in use (pool resources)
  - Contact coordinator for work
  - Coordinator: partition large parallel app into small tasks
  - Assign compute/storage tasks to PCs
- Examples: Seti@home, P2P backups