UNIT – 2

Virtual Machines and Virtualization of Clusters and Data Centers :-

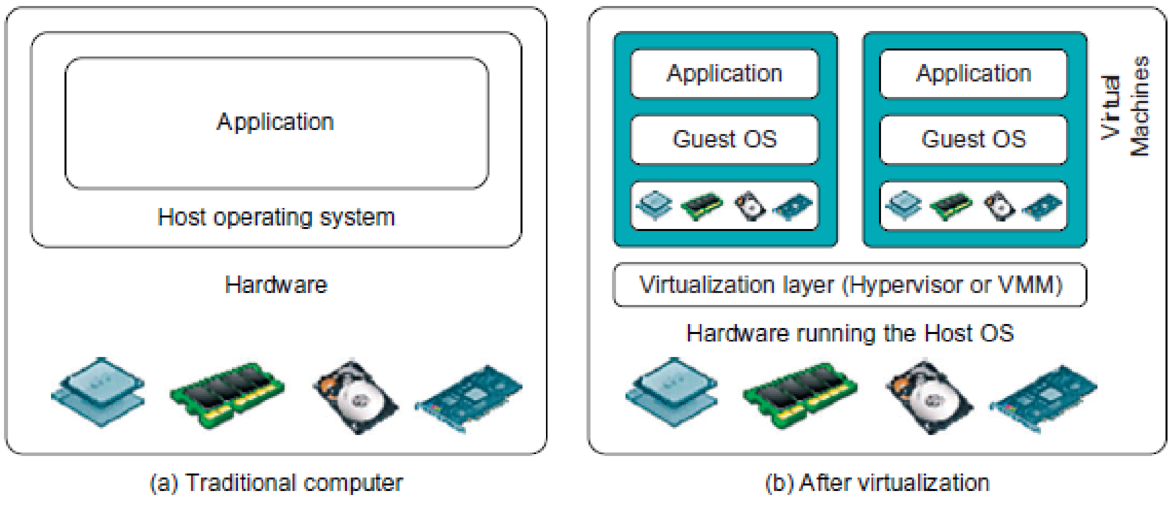
Implementation Levels of Virtualization, Virtualization Structures/ Tools and mechanisms, virtualization of CPU, Memory and I/O Devices, Virtual Clusters and Resource Management, Virtualization for Data Center Automation.

1. Implementation Levels Of Virtualization:

- Virtualization is a computer architecture technology by which multiple virtual machines (VMs) are multiplexed in the same hardware machine.
- The purpose of a VM is to enhance resource sharing by many users and improve computer performance in terms of resource utilization and application flexibility.
- The idea is to separate the hardware from the software to yield better system efficiency.
- For example, computer users gained access to much enlarged memory space when the concept of virtual memory was introduced.
- Similarly, virtualization techniques can be applied to enhance the use of compute engines, networks, and storage.

1.1 Levels of Virtualization Implementation:

A traditional computer runs with a host operating system specially tailored for its hardware architecture, as shown in Figure . After virtualization, different user applications managed by their own operating systems (guest OS) can run on the same hardware, independent of the host OS. This is often done by adding additional software, called a virtualization layer.



- This virtualization layer is known as hypervisor or virtual machine monitor (VMM).
- The virtualization software creates the abstraction of VMs by interposing a virtualization layer at various levels of a computer system.
- Common virtualization layers include the
    1. instruction set architecture (ISA) level
    2. hardware level
    3. operating system level
    4. library support level
    5. application level

1.2 Instruction Set Architecture Level:

- At the ISA level, virtualization is performed by emulating a given ISA by the ISA of the host machine.
- With this approach, it is possible to run a large amount of legacy binary code written for various processors on any given new hardware host machine.
- Instruction set emulation leads to virtual ISAs created on any hardware machine.
- The basic emulation method is through code interpretation.
- An interpreter program interprets the source instructions to target instructions one by one.
- One source instruction may require tens or hundreds of native target instructions to perform its function. Obviously, this process is relatively slow.
- For better performance, dynamic binary translation is desired. This approach translates basic blocks of dynamic source instructions to target instructions.
- The basic blocks can also be extended to program traces or super blocks to increase translation efficiency.

1.2.1 Hardware Abstraction Level:

- Hardware-level virtualization is performed right on top of the bare hardware.
- On the one hand, this approach generates a virtual hardware environment for a VM.
- On the other hand, the process manages the underlying hardware through virtualization.
- The intention is to upgrade the hardware utilization rate by multiple users concurrently.

1.2.2 Operating System Level:

- This refers to an abstraction layer between traditional OS and user applications.
- OS-level virtualization creates isolated containers on a single physical server and the OS instances to utilize the hardware and software in data centers.
- The containers behave like real servers.

- OS-level virtualization is commonly used in creating virtual hosting environments to allocate hardware resources among a large number of mutually distrusting users.

1.2.3 Library Support Level:

- Most applications use APIs exported by user-level libraries rather than using lengthy system calls by the OS.
- Virtualization with library interfaces is possible by controlling the communication link between applications and the rest of a system through API hooks.
- The software tool WINE has implemented this approach to support Windows applications on top of UNIX hosts.

1.2.4 User-Application Level:

- Virtualization at the application level virtualizes an application as a VM.
- On a traditional OS, an application often runs as a process.
- Therefore, application-level virtualization is also known as process-level virtualization.
- In this scenario, the virtualization layer sits as an application program on top of the operating system, and the layer exports an abstraction of a VM that can run programs written and compiled to a particular abstract machine definition.

1.3 Relative Merits of Different Approaches:

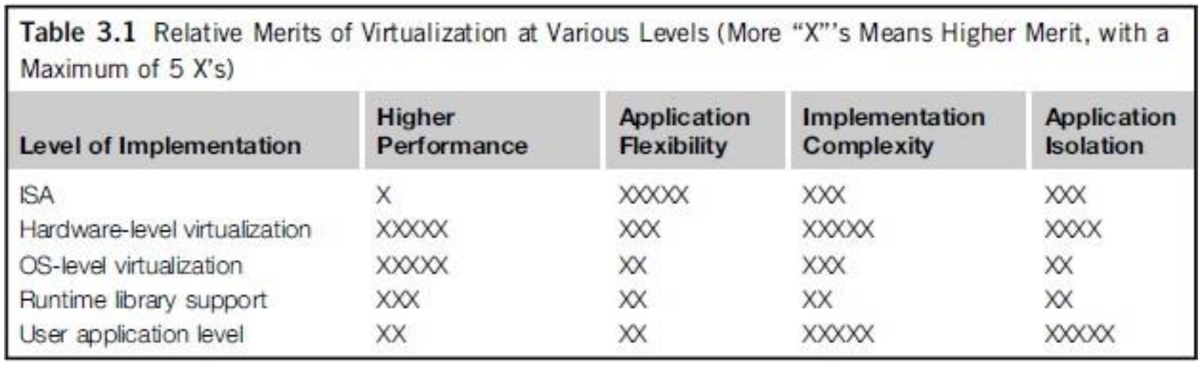


Table 3.1 Relative Merits of Virtualization at Various Levels (More "X"'s Means Higher Merit, with a Maximum of 5 X's)

Level of Implementation	Higher Performance	Application Flexibility	Implementation Complexity	Application Isolation
ISA	X	XXXXX	XXX	XXX
Hardware-level virtualization	XXXXX	XXX	XXXXX	XXXX
OS-level virtualization	XXXXX	XX	XXX	XX
Runtime library support	XXX	XX	XX	XX
User application level	XX	XX	XXXXX	XXXXX

1.4 VMM Design Requirements and Providers: There are three requirements for a VMM.

- First, a VMM should provide an environment for programs which is essentially identical to the original machine.
- Second, programs run in this environment should show, at worst, only minor decreases in speed.
- Third, a VMM should be in complete control of the system resources.

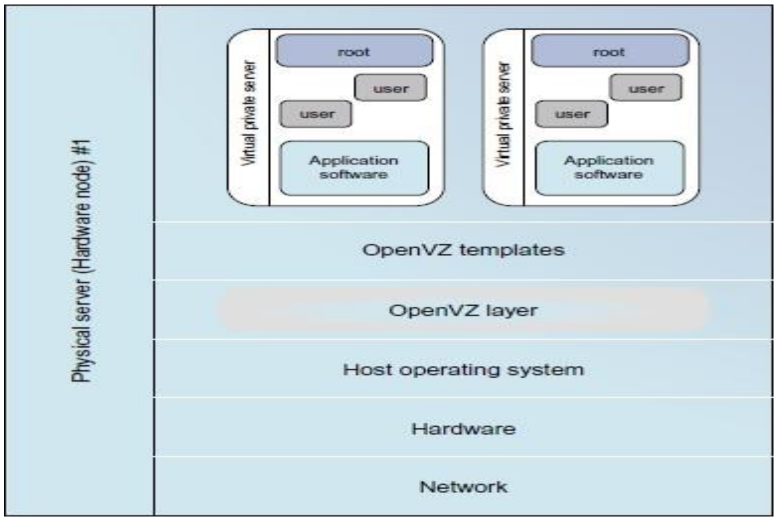
1.5 Virtualization Support at the OS Level:

Cloud computing is transforming the computing landscape by shifting the hardware and staffing costs of managing a computational center to third parties, just like banks.
cloud computing has at least two challenges.

- The first is the ability to use a variable number of physical machines and VM instances depending on the needs of a problem.
- The second challenge concerns the slow operation of instantiating new VMs. Currently, new VMs originate either as fresh boots or as replicates of a template VM, unaware of the current application state.

1.6 Why OS-Level Virtualization?

To reduce the performance overhead of hardware-level virtualization, even hardware modification is needed.
OS-level virtualization provides a feasible solution for these hardware-level virtualization issues.
Operating system virtualization inserts a virtualization layer inside an operating system to partition a machine's physical resources.
It enables multiple isolated VMs within a single operating system kernel.
This kind of VM is often called a virtual execution environment (VE), Virtual Private System (VPS), or simply container.



1.7 Advantages of OS Extensions: Compared to hardware-level virtualization, the benefits of OS extensions are twofold:
VMs at the operating system level have minimal startup/shutdown costs, low resource requirements, and high scalability for an OS-level VM, it is possible for a VM and its host environment to synchronize state changes when necessary.

These benefits can be achieved via two mechanisms of OS-level virtualization:

- All OS-level VMs on the same physical machine share a single operating system kernel
- The virtualization layer can be designed in a way that allows processes in VMs to access as many resources of the host machine as possible, but never to modify them.

The first and second benefits can be used to overcome the defects of slow initialization of VMs at the hardware level

1.8 Disadvantages of OS Extensions

- The main disadvantage of OS extensions is that all the VMs at operating system level on a single container must have the same kind of guest operating system.
- That is, although different OS-level VMs may have different operating system distributions, they must pertain to the same operating system family.
- The virtualization layer is inserted inside the OS to partition the hardware resources for multiple VMs to run their applications in multiple virtual environments.

1.9 Virtualization on Linux or Windows Platforms:

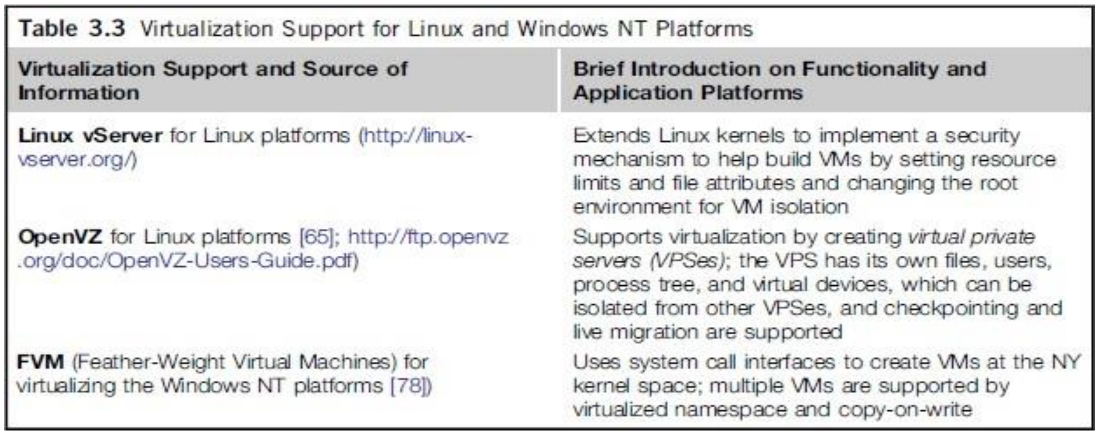


Table 3.3 Virtualization Support for Linux and Windows NT Platforms

Virtualization Support and Source of Information	Brief Introduction on Functionality and Application Platforms
Linux vServer for Linux platforms (http://linux-vserver.org/)	Extends Linux kernels to implement a security mechanism to help build VMs by setting resource limits and file attributes and changing the root environment for VM isolation
OpenVZ for Linux platforms [65]; http://ftp.openvz.org/doc/OpenVZ-Users-Guide.pdf)	Supports virtualization by creating *virtual private servers (VPSes)*; the VPS has its own files, users, process tree, and virtual devices, which can be isolated from other VPSes, and checkpointing and live migration are supported
FVM (Feather-Weight Virtual Machines) for virtualizing the Windows NT platforms [78])	Uses system call interfaces to create VMs at the NY kernel space; multiple VMs are supported by virtualized namespace and copy-on-write

1.10 Middleware Support for Virtualization:

Library-level virtualization is also known as user-level Application Binary Interface or API emulation.

This type of virtualization can create execution environments for running alien programs on a platform rather than creating a VM to run the entire operating system.

API call interception and remapping are the key functions performed.

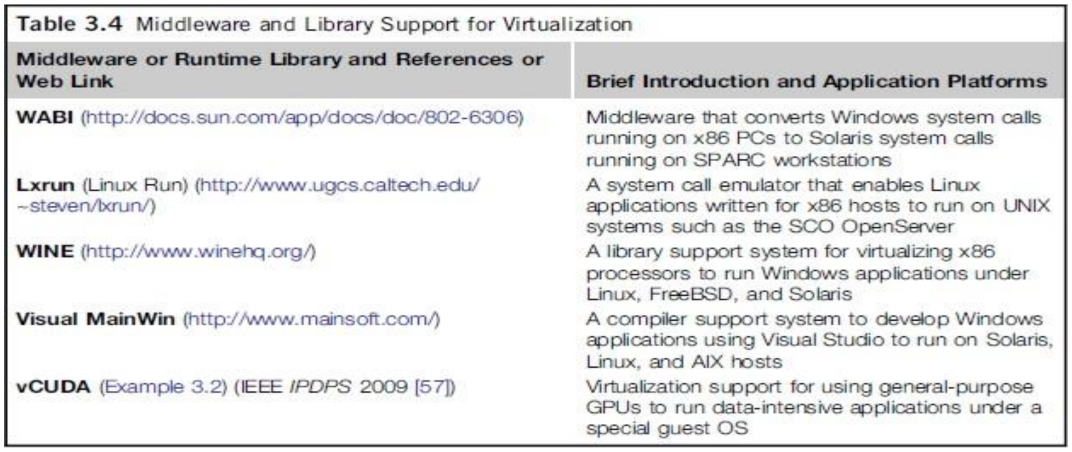


Table 3.4 Middleware and Library Support for Virtualization

Middleware or Runtime Library and References or Web Link	Brief Introduction and Application Platforms
WABI (http://docs.sun.com/app/docs/doc/802-6306)	Middleware that converts Windows system calls running on x86 PCs to Solaris system calls running on SPARC workstations
Lxrun (Linux Run) (http://www.ugcs.caltech.edu/~steven/lxrun/)	A system call emulator that enables Linux applications written for x86 hosts to run on UNIX systems such as the SCO OpenServer
WINE (http://www.winehq.org/)	A library support system for virtualizing x86 processors to run Windows applications under Linux, FreeBSD, and Solaris
Visual MainWin (http://www.mainsoft.com/)	A compiler support system to develop Windows applications using Visual Studio to run on Solaris, Linux, and AIX hosts
vCUDA (Example 3.2) (IEEE *IPDPS* 2009 [57])	Virtualization support for using general-purpose GPUs to run data-intensive applications under a special guest OS

2. VIRTUALIZATION STRUCTURES/TOOLS AND MECHANISMS

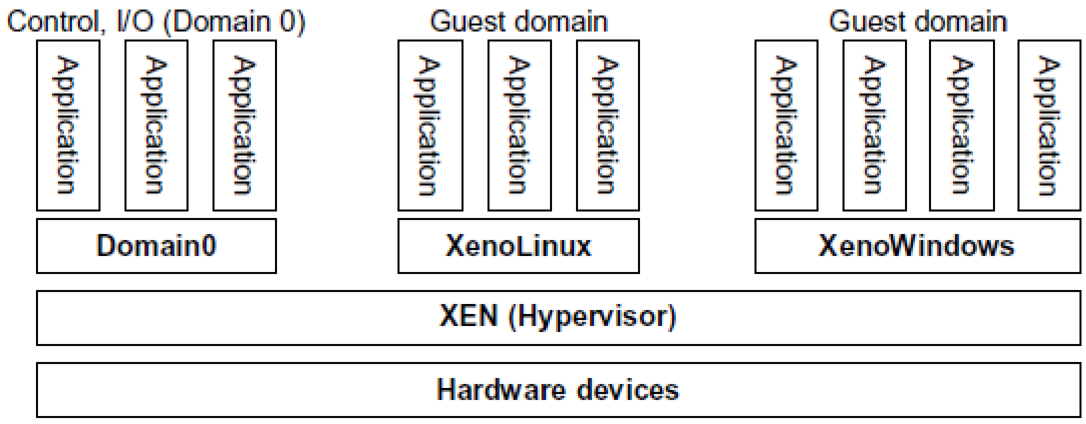
Depending on the position of the virtualization layer, there are several classes of VM architectures, namely the hypervisor architecture, para-virtualization, and host-based virtualization. The hypervisor is also known as the VMM (Virtual Machine Monitor)

2.1 HYPERVISOR AND XEN ARCHITECTURE:

- The hypervisor supports hardware-level virtualization on bare metal devices like CPU, memory, disk and network interfaces.
- The hypervisor software sits directly between the physical hardware and its OS. This virtualization layer is referred to as either the VMM or the hypervisor. The hypervisor provides hypercalls for the guest OSes and applications.
- Depending on the functionality, a hypervisor can assume a micro-kernel architecture like the Microsoft Hyper-V. Or it can assume a monolithic hypervisor architecture like the VMware ESX for server virtualization.

2.2 The Xen Architecture:

- Xen is an open source hypervisor program developed by Cambridge University. Xen is a microkernel hypervisor, which separates the policy from the mechanism.
- The Xen hypervisor implements all the mechanisms, leaving the policy to be handled by Domain 0, as shown in Figure
- Xen does not include any device drivers natively. It just provides a mechanism by which a guest OS can have direct access to the physical devices.
- Xen provides a virtual environment located between the hardware and the OS.
- The core components of a Xen system are the hypervisor, kernel, and applications.



- The guest OS, which has control ability, is called Domain 0, and the others are called Domain U.
- Domain 0 is a privileged guest OS of Xen. It is first loaded when Xen boots without any file system drivers being available. Domain 0 is designed to access hardware directly and manage devices.
- Therefore, one of the responsibilities of Domain 0 is to allocate and map hardware resources for the guest domains

2.3 Binary Translation with Full Virtualization:

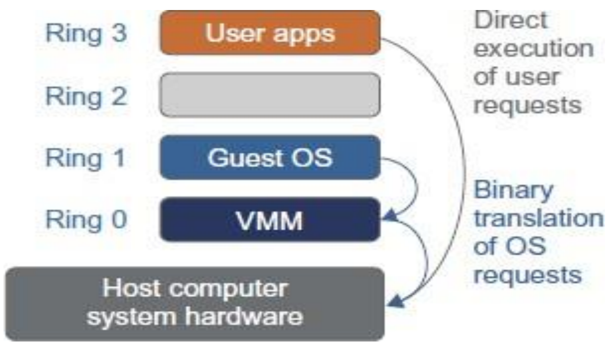
- Depending on implementation technologies, hardware virtualization can be classified into two categories: full virtualization and host-based virtualization.
- Full virtualization does not need to modify the host OS. It relies on binary translation to trap and to virtualize the execution of certain sensitive, nonvirtualizable instructions.
- In a host-based system, both a host OS and a guest OS are used. A virtualization software layer is built between the host OS and guest OS.

2.4 Full Virtualization:

- With full virtualization, noncritical instructions run on the hardware directly.
- while critical instructions are discovered and replaced with traps into the VMM to be emulated by software. Both the hypervisor and VMM approaches are considered full virtualization.
- **Note:**Why are only critical instructions trapped into the VMM? This is because binary translation can incur a large performance overhead. Noncritical instructions do not control hardware or threaten the security of the system, but critical instructions do.

Therefore, running noncritical instructions on hardware not only can promote efficiency, but also can ensure system security.

2.5 Binary Translation of Guest OS Requests Using a VMM


Ring 3
User apps
Ring 2
Ring 1
Guest OS
Ring 0
VMM
Host computer system hardware
Direct execution of user requests
Binary translation of OS requests

- As shown in the above Figure , VMware puts the VMM at Ring 0 and the guest OS at Ring 1.
- The VMM scans the instruction stream and identifies the privileged, control and behavior-sensitive instructions.
- When these instructions are identified, they are trapped into the VMM, which emulates the behavior of these instructions.
- The method used in this emulation is called binary translation. Therefore, full virtualization combines binary translation and direct execution.
- The guest OS is completely decoupled from the underlying hardware. Consequently, the guest OS is unaware that it is being virtualized.

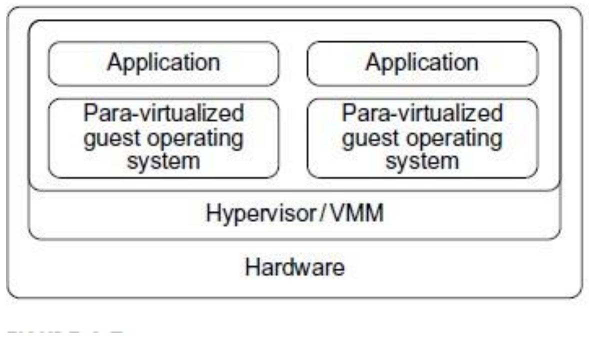
2.6 Host-Based Virtualization

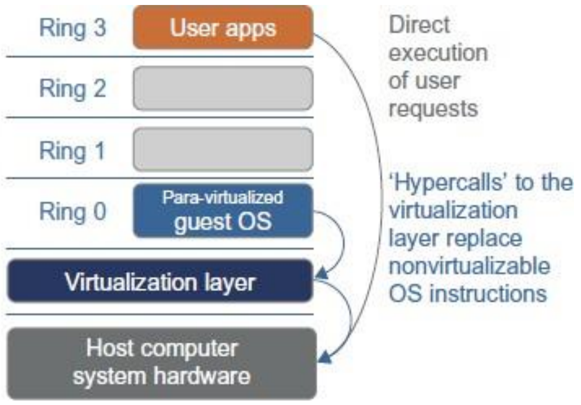
- An alternative VM architecture is to install a virtualization layer on top of the host OS. This host OS is still responsible for managing the hardware.
- The guest OSes are installed and run on top of the virtualization layer. Dedicated applications may run on the VMs. Certainly, some other applications can also run with the host OS directly. This host-based architecture has some distinct advantages, as enumerated next.
- First, the user can install this VM architecture without modifying the host OS. The virtualizing software can rely on the host OS to provide device drivers and other low-level services. This will simplify the VM design and ease its deployment.
- Second, the host-based approach appeals to many host machine configurations. Compared to the hypervisor/VMM architecture, the performance of the host-based

architecture may also be low. When an application requests hardware access, it involves four layers of mapping which downgrades performance significantly.

2.7 Para-Virtualization with Compiler Support:

Para-virtualization needs to modify the guest operating systems. A para-virtualized VM provides special APIs requiring substantial OS modifications in user applications. Performance degradation is a critical issue of a virtualized system. No one wants to use a VM if it is much slower than using a physical machine.


Application
Application
Para-virtualized guest operating system
Para-virtualized guest operating system
Hypervisor/VMM
Hardware


Ring 3
User apps
Ring 2
Ring 1
Ring 0
Para-virtualized guest OS
Virtualization layer
Host computer system hardware
Direct execution of user requests
'Hypercalls' to the virtualization layer replace nonvirtualizable OS instructions

- ➢ The virtualization layer can be inserted at different positions in a machine software stack. Para-virtualization attempts to reduce the virtualization overhead, and thus improve performance by modifying only the guest OS kernel.
- ➢ Above figure illustrates the concept of a para-virtualized VM architecture. The guest operating systems are para-virtualized.
- ➢ They are assisted by an intelligent compiler to replace the nonvirtualizable OS instructions by hypercalls .The traditional x86 processor offers four instruction execution rings: Rings 0, 1, 2, and 3. The lower the ring number, the higher the privilege of instruction being executed.
- ➢ The OS is responsible for managing the hardware and the privileged instructions to execute at Ring 0, while user-level applications run at Ring 3. The best example of para-virtualization is the KVM to be described below.

2.8 Para-Virtualization Architecture

- When the x86 processor is virtualized, a virtualization layer is inserted between the hardware and the OS.
- According to the x86 ring definition, the virtualization layer should also be installed at Ring 0. Different instructions at Ring 0 may cause some problems.
- In the above figure-b, we show that para-virtualization replaces nonvirtualizable instructions with hyper calls that communicate directly with the hypervisor or VMM. When the guest OS kernel is modified for virtualization, it can no longer run on the hardware directly.
- Although para-virtualization reduces the overhead, it has incurred other problems.
- First, its compatibility and portability may be in doubt, because it must support the unmodified OS as well.
- Second, the cost of maintaining para-virtualized OSes is high, because they may require deep OS kernel modifications.

2.9 KVM (Kernel-Based VM):

- This is a Linux para-virtualization system. Memory management and scheduling activities are carried out by the existing Linux kernel.
- The KVM does the rest, which makes it simpler than the hypervisor that controls the entire machine.
- KVM is a hardware-assisted para-virtualization tool, which improves performance and supports unmodified guest OSes such as Windows, Linux, Solaris, and other UNIX variants.

3. VIRTUALIZATION OF CPU, MEMORY, AND I/O DEVICES

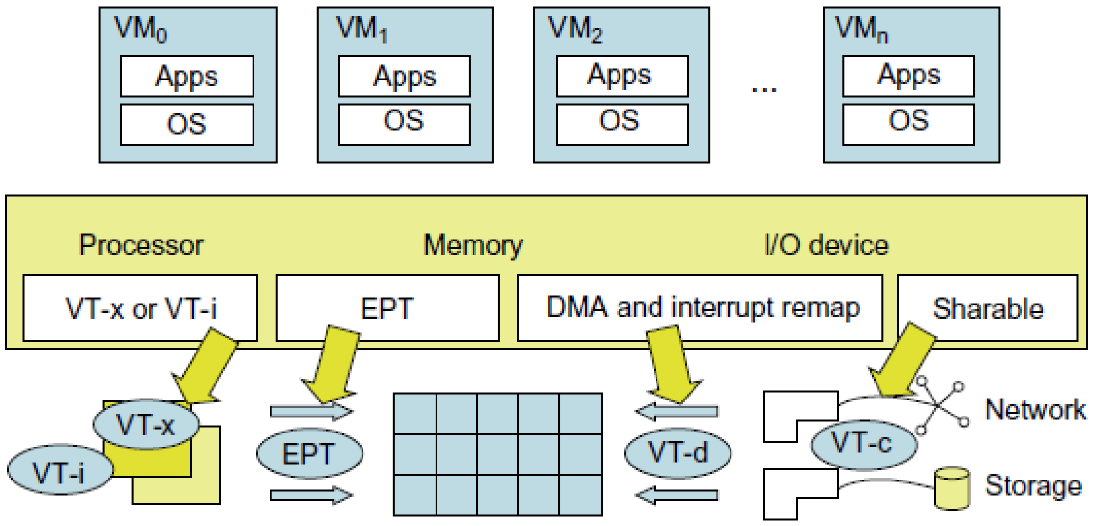
3.1 Hardware Support for Virtualization:

- Modern operating systems and processors permit multiple processes to run simultaneously.
- If there is no protection mechanism in a processor, all instructions from different processes will access the hardware directly and cause a system crash.
- Therefore, all processors have at least two modes, user mode and supervisor mode, to ensure controlled access of critical hardware.

- Instructions running in supervisor mode are called privileged instructions. Other instructions are unprivileged instructions.
- In a virtualized environment, it is more difficult to make OSes and applications run correctly because there are more layers in the machine stack. Example discusses Intel's hardware support approach.
- One or more guest OS can run on top of the hypervisor. KVM (Kernel-based Virtual Machine) is a Linux kernel virtualization infrastructure.
- KVM can support hardware-assisted virtualization and paravirtualization by using the Intel VT-x or AMD-v and VirtIO framework, respectively.

3.1.1 Hardware Support for Virtualization in the Intel x86 Processor:

Since software-based virtualization techniques are complicated and incur performance overhead, Intel provides a hardware-assist technique to make virtualization easy and improve performance.



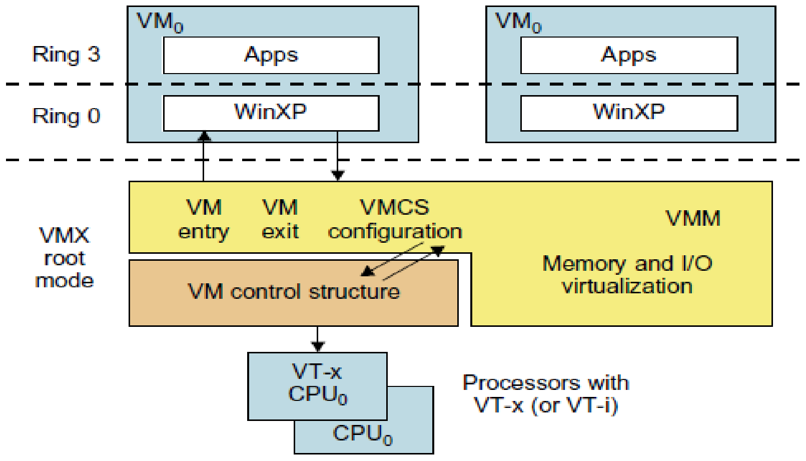
The above figure provides an overview of Intel's full virtualization techniques. For processor virtualization, Intel offers the VT-x or VT-i technique. VT-x adds a privileged mode (VMX Root Mode) and some instructions to processors. This enhancement traps all sensitive instructions in the VMM automatically. For memory virtualization, Intel offers the EPT, which translates the virtual address to the machine's physical addresses to improve performance. For I/O virtualization, Intel implements VT-d and VT-c to support this.

3.2 CPU Virtualization:

- A CPU architecture is virtualizable if it supports the ability to run the VM's privileged and unprivileged instructions in the CPU's user mode while the VMM runs in supervisor mode.
- When the privileged instructions including control- and behavior-sensitive instructions of a VM are executed, they are trapped in the VMM.
- In this case, the VMM acts as a unified mediator for hardware access from different VMs to guarantee the correctness and stability of the whole system.
- However, not all CPU architectures are virtualizable. RISC CPU architectures can be naturally virtualized because all control- and behavior-sensitive instructions are privileged instructions.
- On the contrary, x86 CPU architectures are not primarily designed to support virtualization. This is because about 10 sensitive instructions, such as SGDT and SMSW, are not privileged instructions. When these instructions execute in virtualization, they cannot be trapped in the VMM.

3.3 Hardware-Assisted CPU Virtualization:

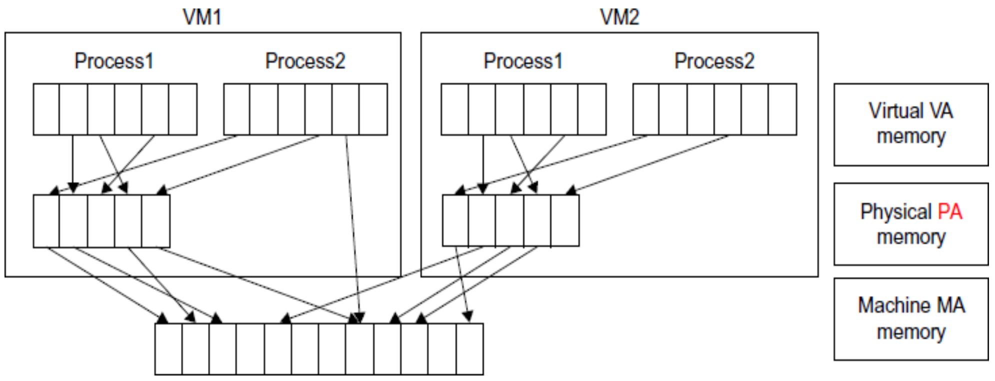
- This technique attempts to simplify virtualization because full or paravirtualization is complicated.



- Intel and AMD add an additional mode called privilege mode level (some people call it Ring-1) to x86 processors. Therefore, operating systems can still run at Ring 0 and the hypervisor can run at Ring 1. All the privileged and sensitive instructions are trapped in the hypervisor automatically. This technique removes the difficulty of implementing binary translation of full virtualization. It also lets the operating system run in VMs without modification.

3.4 Memory Virtualization:

- Virtual memory virtualization is similar to the virtual memory support provided by modern operating systems.
- In a traditional execution environment, the operating system maintains mappings of virtual memory to machine memory using page tables, which is a one-stage mapping from virtual memory to machine memory.
- All modern x86 CPUs include a memory management unit (MMU) and a translation lookaside buffer (TLB) to optimize virtual memory performance.
- However, in a virtual execution environment, virtual memory virtualization involves sharing the physical system memory in RAM and dynamically allocating it to the physical memory of the VMs.
- That means a two-stage mapping process should be maintained by the guest OS and the VMM, respectively: virtual memory to physical memory and physical memory to machine memory.
- Furthermore, MMU virtualization should be supported, which is transparent to the guest OS.
- The guest OS continues to control the mapping of virtual addresses to the physical memory addresses of VMs.
- But the guest OS cannot directly access the actual machine memory.
- The VMM is responsible for mapping the guest physical memory to the actual machine memory. The below figure shows the two-level memory mapping procedure.

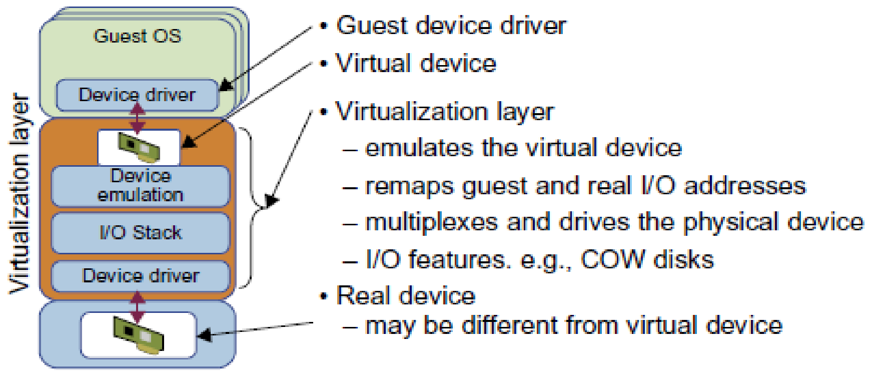


3.5 I/O Virtualization:

- I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware.
- There are three ways to implement I/O virtualization: full device emulation, para-virtualization, and direct I/O.

1. **Full device emulation** is the first approach for I/O virtualization. Generally, this approach emulates well-known, real-world devices. All the functions of a device or bus infrastructure, such as device enumeration, identification, interrupts, and DMA, are replicated in software. This software is located in the VMM and acts as a virtual device. The I/O access requests of the guest OS are trapped in the VMM which interacts with the I/O devices. The full device emulation approach is shown in Fig.

   A single hardware device can be shared by multiple VMs that run concurrently. However, software emulation runs much slower than the hardware it emulates.


Guest OS
Device driver
Device emulation
I/O Stack
Device driver
Virtualization layer
• Guest device driver
• Virtual device
• Virtualization layer
– emulates the virtual device
– remaps guest and real I/O addresses
– multiplexes and drives the physical device
– I/O features. e.g., COW disks
• Real device
– may be different from virtual device

2. **Para-virtualization method:** Para virtualization method of I/O virtualization is typically used in Xen. It is also known as the split driver model consisting of a frontend driver and a backend driver.
   The frontend driver is running in Domain U and the backend driver is running in Domain 0. They interact with each other via a block of shared memory.
   The frontend driver manages the I/O requests of the guest OSes and the backend driver is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs.
   Although para-I/O-virtualization achieves better device performance than full device emulation, it comes with a higher CPU overhead.
3. **Direct I/O virtualization** lets the VM access devices directly. It can achieve close-to-native performance without high CPU costs.

Current direct I/O virtualization implementations focus on networking for mainframes. There are a lot of challenges for commodity hardware devices.

3.6 Virtualization in Multi-Core Processors

- Virtualizing a multi-core processor is relatively more complicated than virtualizing a uni-core processor.
- Though multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip, muti-core virtualiuzation has raised some new challenges to computer architects, compiler constructors, system designers, and application programmers.
- There are mainly two difficulties: Application programs must be parallelized to use all cores fully, and software must explicitly assign tasks to the cores, which is a very complex problem.
- Concerning the first challenge, new programming models, languages, and libraries are needed to make parallel programming easier.
- The second challenge has spawned research involving scheduling algorithms and resource management policies.
- Yet these efforts cannot balance well among performance, complexity, and other issues.
- What is worse, as technology scales, a new challenge called dynamic heterogeneity is emerging to mix the fat CPU core and thin GPU cores on the same chip, which further complicates the multi-core or many-core resource management.
- The dynamic heterogeneity of hardware infrastructure mainly comes from less reliable transistors and increased complexity in using the transistors.

4. VIRTUAL CLUSTERS AND RESOURCE MANAGEMENT

A physical cluster is a collection of servers (physical machines) interconnected by a physical network such as a LAN. We introduce virtual clusters and study its properties as well as explore their potential applications. There are three critical design issues of virtual clusters: live migration of Vms, memory and file migrations, and dynamic deployment of virtual clusters.

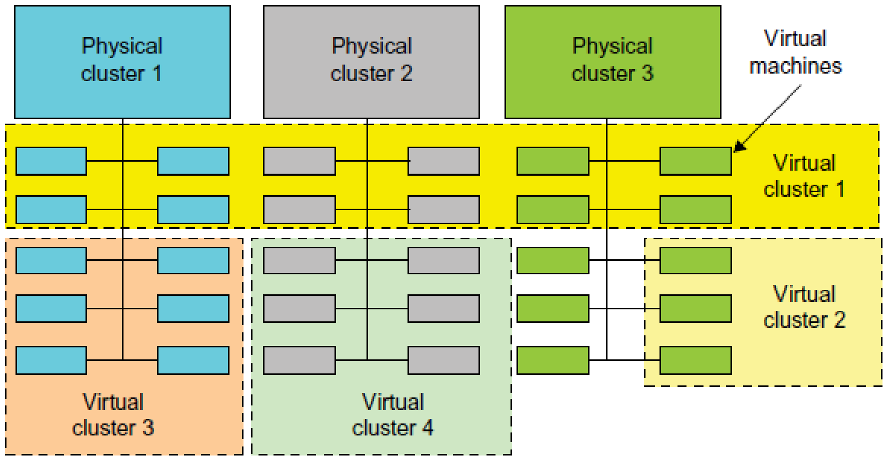
4.1 Physical versus Virtual Clusters

- Virtual clusters are built with VMs installed at distributed servers from one or more physical clusters.
- The VMs in a virtual cluster are interconnected logically by a virtual network across several physical networks. Figure 3.18 illustrates the concepts of virtual clusters and physical clusters.

- Each virtual cluster is formed with physical machines or a VM hosted by multiple physical clusters.
- The virtual cluster boundaries are shown as distinct boundaries.

The **provisioning of VMs** to a virtual cluster is done dynamically to have the following **properties:**

- The virtual cluster nodes can be either physical or virtual machines. Multiple VMs running with different OSes can be deployed on the same physical node.
- A VM runs with a guest OS, which is often different from the host OS, that manages the resources in the physical machine, where the VM is implemented.
- The purpose of using VMs is to consolidate multiple functionalities on the same server. This will greatly enhance server utilization and application flexibility.
- VMs can be colonized (replicated) in multiple servers for the purpose of promoting distributed parallelism, fault tolerance, and disaster recovery.
- The size (number of nodes) of a virtual cluster can grow or shrink dynamically, similar to the way an overlay network varies in size in a peer-to-peer (P2P) network.
- The failure of any physical nodes may disable some VMs installed on the failing nodes. But the failure of VMs will not pull down the host system.



4.2 Fast Deployment and Effective Scheduling:

- The system should have the capability of fast deployment. Here, deployment means two things:

- to construct and distribute software stacks (OS, libraries, applications) to a physical node inside clusters as fast as possible, and to quickly switch runtime environments from one user's virtual cluster to another user's virtual cluster.
- If one user finishes using his system, the corresponding virtual cluster should shut down or suspend quickly to save the resources to run other VMs for other users.

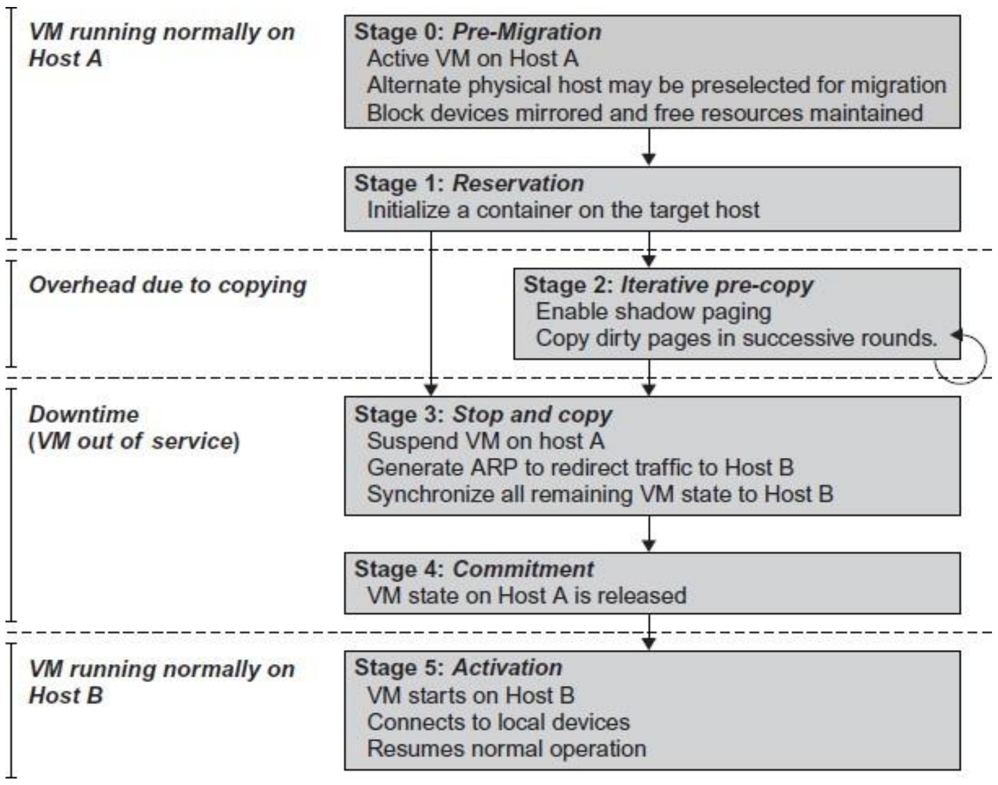
4.3 High-Performance Virtual Storage:

- The template VM can be distributed to several physical hosts in the cluster to customize the VMs. In addition, existing software packages reduce the time for customization as well as switching virtual environment.
- It is important to efficiently manage the disk space occupied by template software packages. Some storage architecture design can be applied to reduce duplicated blocks in a distributed file system of virtual clusters. Hash values are used to compare the contents of data blocks.
- Users have their own profiles which store the identification of the data blocks for corresponding VMs in a user specific virtual cluster. New blocks are created when users modify the corresponding data. Newly created blocks are identified in the user's profiles.
- Basically, there are four steps to deploy a group of VMs onto target cluster:
    1. preparing the disk image
    2. configuring the VM
    3. choosing the destination nodes
    4. executing the VM deployment command on every host
- Many systems use templates to simplify the disk image preparation process. A template is a disk image that includes a preinstalled operating system with or without certain application software.
- Users choose a proper template according to their requirements and make a duplicate of it as their own disk image.
- Templates could implement the COW(Copy-On-Write) format. A new COW backup file is very small and easy to create and transfer. Therefore, it definitely reduces disk space consumption. In addition, VM deployment time is much shorter than that of copying the whole raw image file.

4.4 Live VM Migration Steps and Performance Effects:

- In a cluster built with mixed nodes of host and guest systems, the normal method of operation is to run everything on the physical machine.

- When a VM fails, its role could be replaced by another VM on a different node, as long as they both run with the same guest OS.
- In other words, a physical node can fail over to a VM on another host. This is different from physical-to-physical failover in a traditional physical cluster. The advantage is enhanced failover flexibility.
- The potential drawback is that a VM must stop playing its role if its residing host node fails. However, this problem can be mitigated with VM life migration. Figure shows the process of life migration of a VM from host A to host B.
- The migration copies the VM state file from the storage area to the host machine.


VM running normally on Host A
Stage 0: Pre-Migration
Active VM on Host A
Alternate physical host may be preselected for migration
Block devices mirrored and free resources maintained
Stage 1: Reservation
Initialize a container on the target host
Overhead due to copying
Stage 2: Iterative pre-copy
Enable shadow paging
Copy dirty pages in successive rounds.
Downtime (VM out of service)
Stage 3: Stop and copy
Suspend VM on host A
Generate ARP to redirect traffic to Host B
Synchronize all remaining VM state to Host B
Stage 4: Commitment
VM state on Host A is released
VM running normally on Host B
Stage 5: Activation
VM starts on Host B
Connects to local devices
Resumes normal operation

Steps 0 and 1: **Start migration**. This step makes preparations for the migration, including determining the migrating VM and the destination host. Although users could manually make a VM migrate to an appointed host, in most circumstances, the migration is automatically started by strategies such as load balancing and server consolidation.

Steps 2: **Transfer memory**. Since the whole execution state of the VM is stored in memory, sending the VM's memory to the destination node ensures continuity of the service provided by the VM. All of the memory data is transferred in the first round, and then the migration controller recopies the memory data which is changed in the last round. These steps keep iterating until the dirty portion of the memory is small enough to handle the final copy. Although precopying memory is performed iteratively, the execution of programs is not obviously interrupted.

Step 3: Suspend the VM and copy the last portion of the data. The migrating VM's execution is suspended when the last round's memory data is transferred. Other nonmemory data such as CPU and network states should be sent as well. During this step, the VM is stopped and its applications will no longer run. This "service unavailable" time is called the "downtime" of migration, which should be as short as possible so that it can be negligible to users.

Steps 4 and 5: Commit and activate the new host. After all the needed data is copied, on the destination host, the VM reloads the states and recovers the execution of programs in it, and the service provided by this VM continues. Then the network connection is redirected to the new VM and the dependency to the source host is cleared. The whole migration process finishes by removing the original VM from the source host.

4.5 Migration of Memory, Files, and Network Resources

4.5.1 Memory Migration

- The techniques employed for Memory Migration purpose depend upon the characteristics of application/workloads supported by the guest OS.
- Memory migration can be in a range of hundreds of megabytes to a few gigabytes in a typical system today, and it needs to be done in an efficient manner.
- The **Internet Suspend-Resume** (ISR) technique exploits temporal locality as memory states are likely to have considerable overlap in the suspended and the resumed instances of a VM.
- Temporal locality refers to the fact that the memory states differ only by the amount of work done since a VM was last suspended before being initiated for migration.
- The ISR technique deals with situations where the migration of live machines is not a necessity.

4.5.2 File System Migration

- To support VM migration, a system must provide each VM with a consistent, location-independent view of the file system that is available on all hosts.
- A simple way to achieve this is to provide each VM with its own virtual disk which the file system is mapped to and transport the contents of this virtual disk along with the other states of the VM.
- However, due to the current trend of high capacity disks, migration of the contents of an entire disk over a network is not a viable solution.
- Another way is to have a global file system across all machines where a VM could be located. This way removes the need to copy files from one machine to another because all files are network accessible.
- In smart copying, the VMM exploits spatial locality. Typically, people often move between the same small number of locations, such as their home and office.
- In these conditions, it is possible to transmit only the difference between the two file systems at suspending and resuming locations.
- This technique significantly reduces the amount of actual physical data that has to be moved.

4.6 Network Migration

- A migrating VM should maintain all open network connections without relying on forwarding mechanisms on the original host or on support from mobility or redirection mechanisms.
- To enable remote systems to locate and communicate with a VM, each VM must be assigned a virtual IP address known to other entities. This address can be distinct from the IP address of the host machine where the VM is currently located. Each VM can also have its own distinct virtual MAC address.
- The VMM maintains a mapping of the virtual IP and MAC addresses to their corresponding VMs. In general, a migrating VM includes all the protocol states and carries its IP address with it.

4.7 Dynamic Deployment of Virtual Clusters

Table summarizes four virtual cluster research projects. We briefly introduce them here just to identify their design objectives and reported results.

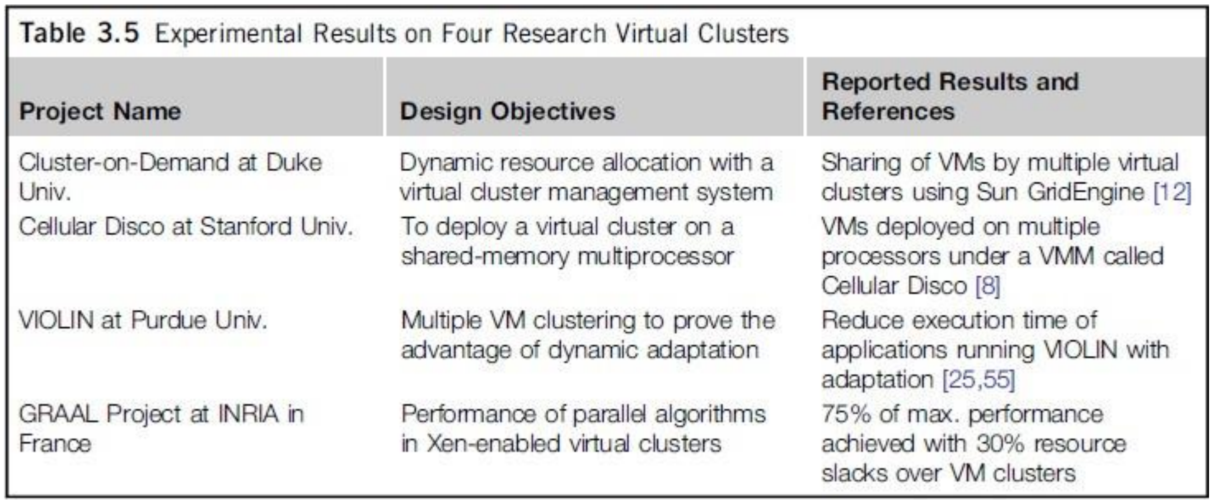


Table 3.5 Experimental Results on Four Research Virtual Clusters

Project Name	Design Objectives	Reported Results and References
Cluster-on-Demand at Duke Univ.	Dynamic resource allocation with a virtual cluster management system	Sharing of VMs by multiple virtual clusters using Sun GridEngine [12]
Cellular Disco at Stanford Univ.	To deploy a virtual cluster on a shared-memory multiprocessor	VMs deployed on multiple processors under a VMM called Cellular Disco [8]
VIOLIN at Purdue Univ.	Multiple VM clustering to prove the advantage of dynamic adaptation	Reduce execution time of applications running VIOLIN with adaptation [25,55]
GRAAL Project at INRIA in France	Performance of parallel algorithms in Xen-enabled virtual clusters	75% of max. performance achieved with 30% resource slacks over VM clusters

5. VIRTUALIZATION FOR DATA-CENTER AUTOMATION

- Data-center automation means that huge volumes of hardware, software, and database resources in these data centers can be allocated dynamically to millions of Internet users simultaneously, with guaranteed QoS and cost-effectiveness.
- Data centers have grown rapidly in recent years, and all major IT companies are pouring their resources into building new data centers.
- In addition, Google, Yahoo!, Amazon, Microsoft, HP, Apple, and IBM are all in the game. All these companies have invested billions of dollars in data center construction and automation.

5.1 Server Consolidation in Data Centers

- In general, the use of VMs increases resource management complexity. This causes a challenge in terms of how to improve resource utilization as well as guarantee QoS in data centers.

  Server virtualization has the following side effects:
- Consolidation enhances hardware utilization. Many underutilized servers are consolidated into fewer servers to enhance resource utilization. Consolidation also facilitates backup services and disaster recovery.
- This approach enables more agile provisioning and deployment of resources. In a virtual environment, the images of the guest OSes and their applications are readily cloned and reused.

- The total cost of ownership is reduced. In this sense, server virtualization causes deferred purchases of new servers, a smaller data-center footprint, lower maintenance costs, and lower power, cooling, and cabling requirements.
- This approach improves availability and business continuity. The crash of a guest OS has no effect on the host OS or any other guest OS. It becomes easier to transfer a VM from one server to another, because virtual servers are unaware of the underlying hardware.

5.2 Virtual Storage Management

- The term “storage virtualization” was widely used before the renaissance of system virtualization. Yet the term has a different meaning in a system virtualization environment.
- Previously, storage virtualization was largely used to describe the aggregation and repartitioning of disks at very coarse time scales for use by physical machines.
- In system virtualization, virtual storage includes the storage managed by VMMs and guest OSes.
- Generally, the data stored in this environment can be classified into two categories: VM images and application data.
- The VM images are special to the virtual environment, while application data includes all other data which is the same as the data in traditional OS environments.
- In virtualization environments, a virtualization layer is inserted between the hardware and traditional operating systems or a traditional operating system is modified to support virtualization. This procedure complicates storage operations.
- On the one hand, storage management of the guest OS performs as though it is operating in a real hard disk while the guest OSes cannot access the hard disk directly.
- On the other hand, many guest OSes contest the hard disk when many VMs are running on a single physical machine.
- Therefore, storage management of the underlying VMM is much more complex than that of guest OSes (traditional Oses).

5.3 Cloud OS for Virtualized Data Centers

Data centers must be virtualized to serve as cloud providers. Table 3.6 summarizes four virtual infrastructure (VI) managers and OSes. These VI managers and OSes are specially tailored for virtualizing data centers which often own a large number of servers in clusters. Nimbus, Eucalyptus, and OpenNebula are all open source software available to the general public. Only vSphere 4 is a proprietary OS for cloud resource virtualization and management over data centers.

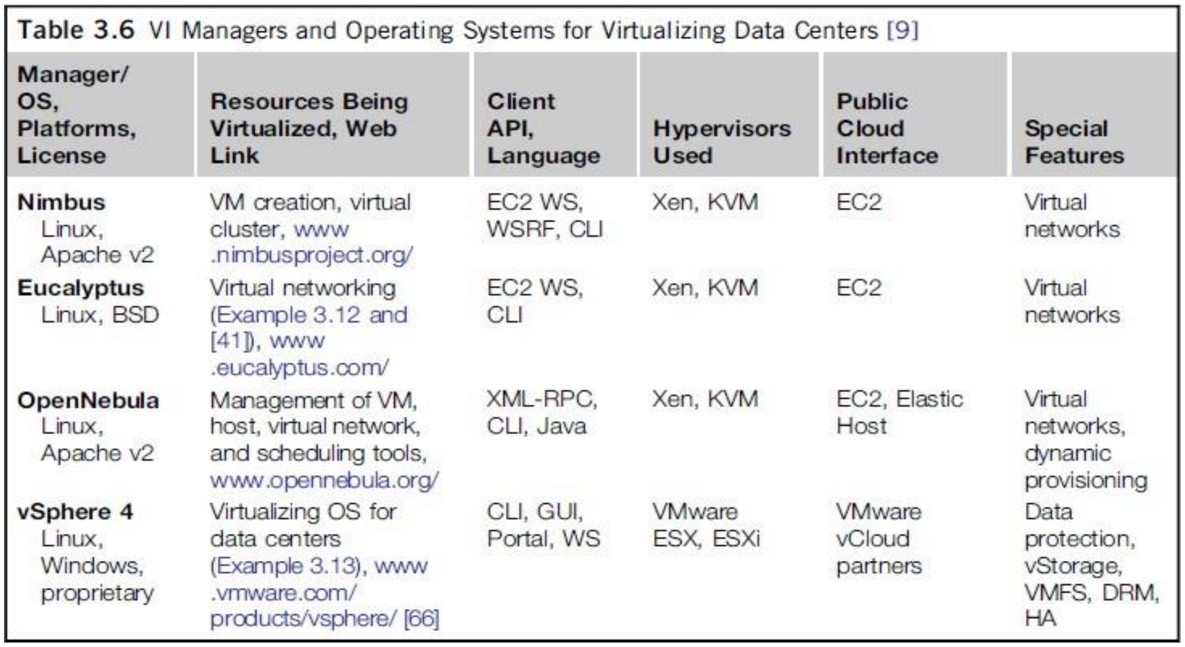


Table 3.6 VI Managers and Operating Systems for Virtualizing Data Centers [9]

Manager/ OS, Platforms, License	Resources Being Virtualized, Web Link	Client API, Language	Hypervisors Used	Public Cloud Interface	Special Features
Nimbus Linux, Apache v2	VM creation, virtual cluster, www .nimbusproject.org/	EC2 WS, WSRF, CLI	Xen, KVM	EC2	Virtual networks
Eucalyptus Linux, BSD	Virtual networking (Example 3.12 and [41]), www .eucalyptus.com/	EC2 WS, CLI	Xen, KVM	EC2	Virtual networks
OpenNebula Linux, Apache v2	Management of VM, host, virtual network, and scheduling tools, www.opennebula.org/	XML-RPC, CLI, Java	Xen, KVM	EC2, Elastic Host	Virtual networks, dynamic provisioning
vSphere 4 Linux, Windows, proprietary	Virtualizing OS for data centers (Example 3.13), www .vmware.com/ products/vsphere/ [66]	CLI, GUI, Portal, WS	VMware ESX, ESXi	VMware vCloud partners	Data protection, vStorage, VMFS, DRM, HA

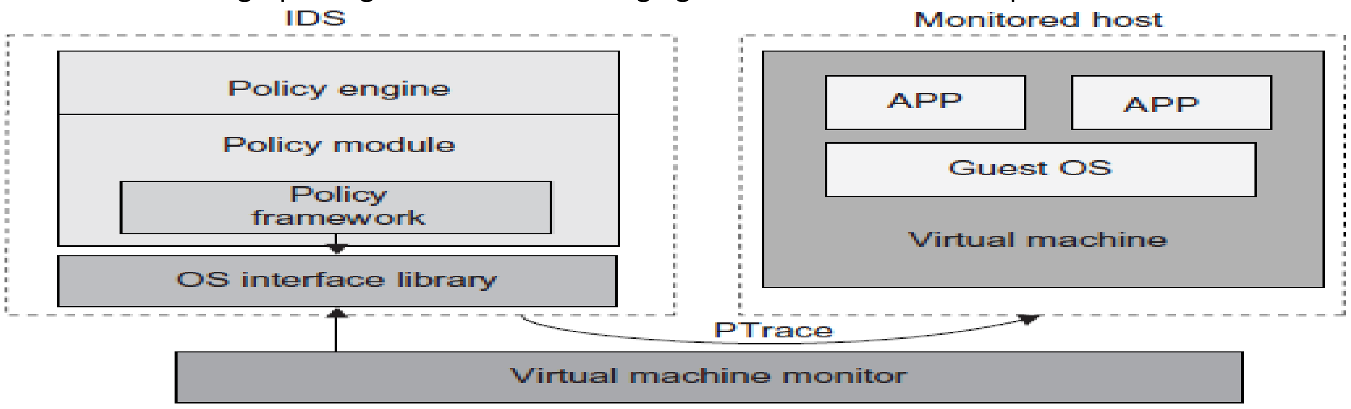
5.4 Trust Management in Virtualized Data Centers

- A VMM changes the computer architecture. It provides a layer of software between the operating systems and system hardware to create one or more VMs on a single physical platform.
- A VM entirely encapsulates the state of the guest operating system running inside it.
- Encapsulated machine state can be copied and shared over the network and removed like a normal file, which proposes a challenge to VM security.
- In general, a VMM can provide secure isolation and a VM accesses hardware resources through the control of the VMM, so the VMM is the base of the security of a virtual system. Normally, one VM is taken as a management VM to have some privileges such as creating, suspending, resuming, or deleting a VM.
- Once a hacker successfully enters the VMM or management VM, the whole system is in danger.
- A subtler problem arises in protocols that rely on the “freshness” of their random number source for generating session keys.
- Considering a VM, rolling back to a point after a random number has been chosen, but before it has been used, resumes execution; the random number, which must be “fresh” for security purposes, is reused.

- With a stream cipher, two different plaintexts could be encrypted under the same key stream, which could, in turn, expose both plaintexts if the plaintexts have sufficient redundancy.
- Noncryptographic protocols that rely on freshness are also at risk. For example, the reuse of TCP initial sequence numbers can raise TCP hijacking attacks

5.5 VM-Based Intrusion Detection

- Intrusions are unauthorized access to a certain computer from local or network users and intrusion detection is used to recognize the unauthorized access.
- An intrusion detection system (IDS) is built on operating systems, and is based on the characteristics of intrusion actions.
- A typical IDS can be classified as a host-based IDS (HIDS) or a network-based IDS (NIDS), depending on the data source.
- A HIDS can be implemented on the monitored system. When the monitored system is attacked by hackers, the HIDS also faces the risk of being attacked.
- A NIDS is based on the flow of network traffic which can't detect fake actions.
- Virtualization-based intrusion detection can isolate guest VMs on the same hardware platform.
- Even some VMs can be invaded successfully; they never influence other VMs, which is similar to the way in which a NIDS operates.
- Furthermore, a VMM monitors and audits access requests for hardware and system software. This can avoid fake actions and possess the merit of a HIDS.
- There are two different methods for implementing a VM-based IDS:
- Either the IDS is an independent process in each VM or a high-privileged VM on the VMM; or the IDS is integrated into the VMM and has the same privilege to access the hardware as well as the VMM. Garfinkel and Rosenblum have proposed an IDS to run on a VMM as a high-privileged VM. The following figure illustrates the concept.


IDS
Policy engine
Policy module
Policy framework
OS interface library
Monitored host
APP
APP
Guest OS
Virtual machine
PTrace
Virtual machine monitor