2.1 VIRTUALIZATION

- Virtualization is a technique, which allows sharing single physical instance of an application or resource among multiple organizations or tenants (customers).
- Virtualization is a proved technology that makes it possible to run multiple operating system and applications on the same server at same time.
- Virtualization is the process of creating a logical(virtual) version of a server operating system, a storage device, or network services.
- The technology that work behind virtualization is known as a virtual machine monitor(VM), or virtual manager which separates compute environments from the actual physical infrastructure.
- Virtualization -- the abstraction of computer resources.
- Virtualization hides the physical characteristics of computing resources from their users, applications, or end users.
- This includes making a single physical resource (such as a server, an operating system, an application, or storage device) appear to function as multiple virtual resources.
- It can also include making multiple physical resources (such as storage devices or servers) appear as a single virtual resource.
- In computing, virtualization refers to the act of creating a virtual (rather than actual) version of something, like computer hardware platforms, operating systems, storage devices, and computer network resources
- Creation of a virtual machine over existing operating system and hardware.
- **Host machine:** The machine on which the virtual machine is created.
- **Guest machine:** virtual machines referred as a guest machine.
- **Hypervisor:** Hypervisor is a firmware or low-level program that acts as a Virtual Machine Manager.

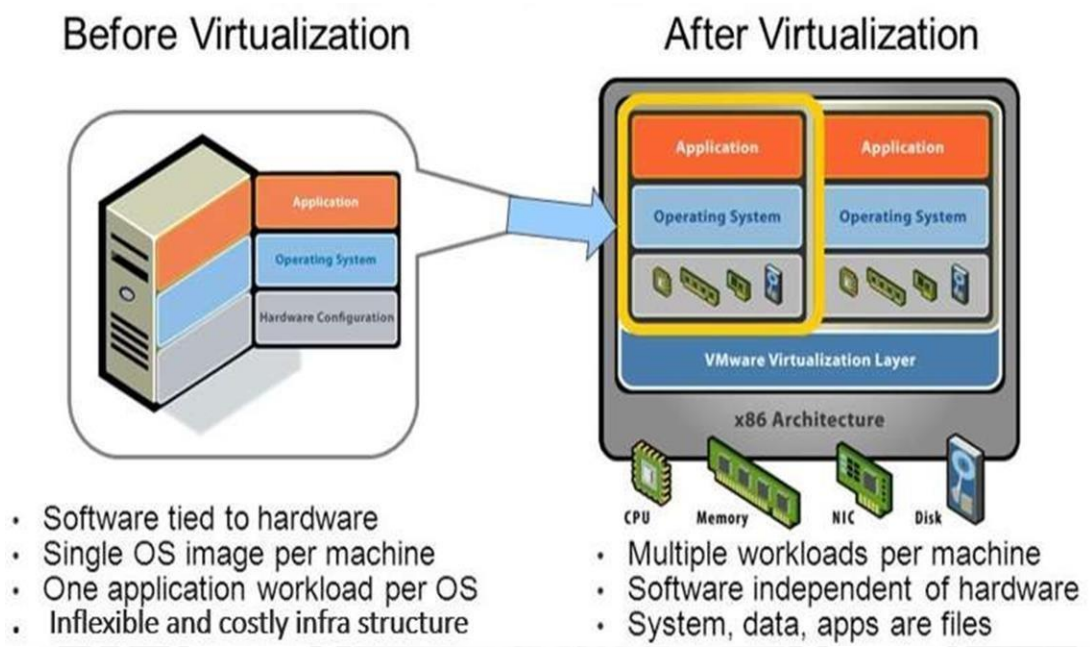


Figure 2.10 Virtualization Example

Advantages of Virtualization:

1. Reduced Costs.
2. Efficient hardware Utilization.
3. Virtualization leads to better resource Utilization and increase performance
4. Testing for software development.
5. Increase Availability
6. Save energy
7. Shifting all your Local Infrastructure to Cloud in a day
8. Possibility to Divide Services
9. Running application not supported by the host.

Disadvantages of Virtualization:

1. Extra Costs.
2. Software Licensing.

2.2 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.

Hardware resources (CPU, memory, I/O devices, etc.) or software resources(operating system and software libraries) can be virtualized in various functional layers.

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.

2.6.1 Levels of Virtualization:

A traditional computer runs with host operating system specially tailored for its hardware architecture, as shown in Figure 2.11 (a). 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 as shown in Figure 2.11 (b). This virtualization layer is known as hypervisor or virtual machine monitor (VMM) .The VMs are shown in the upper boxes, where applications run with their own guest OS over the virtualized CPU, memory, and I/O resources. The main function of the software layer for virtualization is to virtualize the physical hardware of a host machine into virtual resources tobe used by the VMs, exclusively. 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 instruction set architecture (ISA) level, hardware level, operating system level, library support level, and application level.

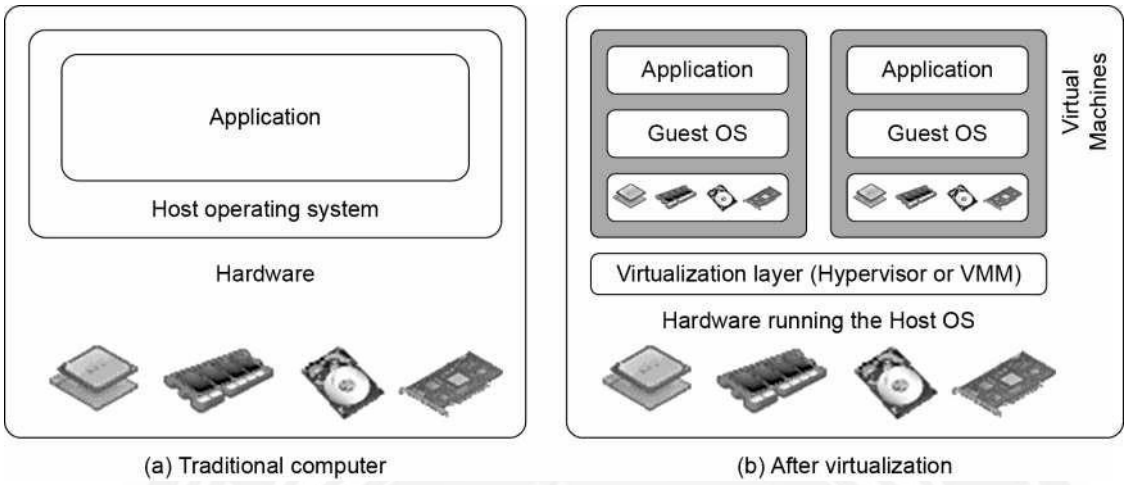


Figure 2.11 The architecture of a computer system before and after Virtualization

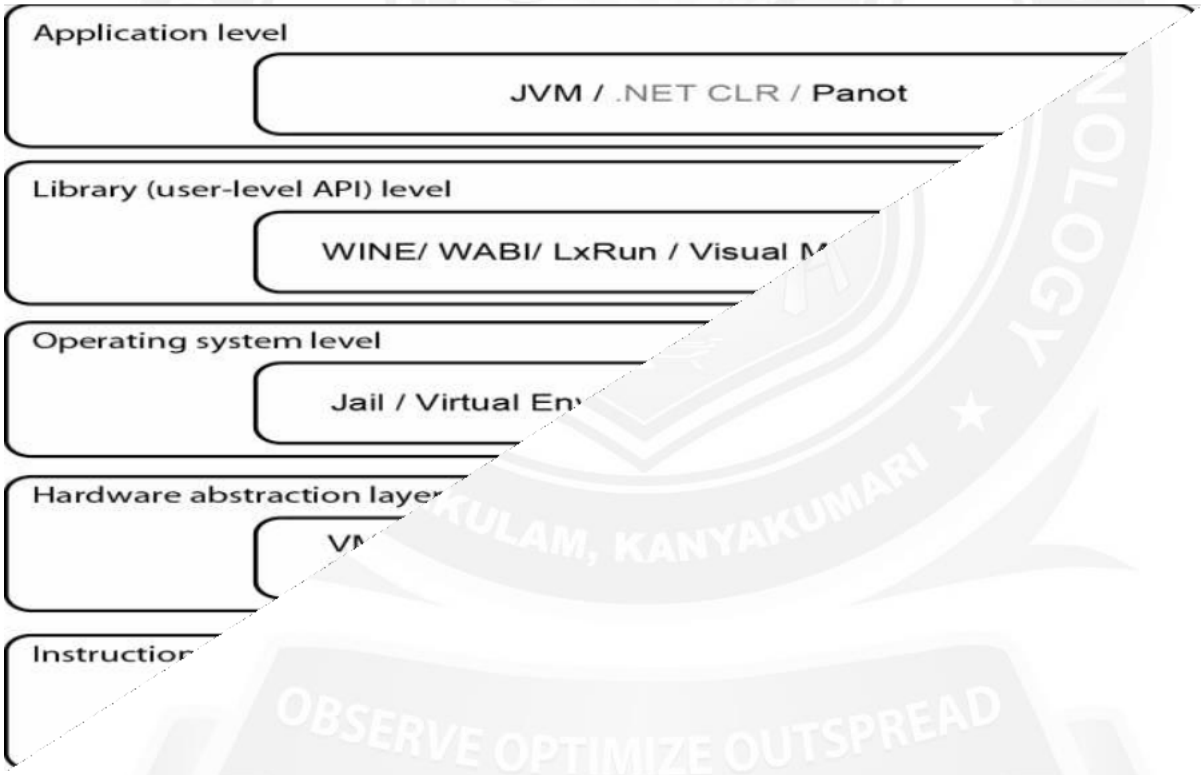


Figure 2.12 Virtualization ranging from hardware to applications in five abstraction levels.

Instruction Set Architecture Level:

At the ISA level, virtualization is performed by emulating a given ISA by the ISA of the host machine. For example, MIPS binary code can run on an x86-based host machine with the help of ISA emulation. 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. OneSource 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 targetinstructions. The basic blocks can also be extended to program traces or super blocks to increase translation efficiency. Instruction set emulation requires binary translation and optimization. A virtual instruction set architecture (V-ISA) thus requires adding a processor-specific software translation layer to the compiler.

Hardware Abstraction Level:

Hardware-level virtualization is performed right on top of the bare hardware. The idea is to virtualize a computer's resources, such as its processors, memory, and I/O devices. The intention is to upgrade the hardware utilization rate by multiple users concurrently.

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 datacenters.

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. It is also used, to a lesser extent, in consolidating server hardware by moving services on separate hosts into containers or VMs on one server.

Library Support Level:

Most applications use APIs exported by user level libraries rather than using lengthy system calls by the OS. Since most systems provide well documented APIs, such an interface becomes another candidate for virtualization.

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. Another

example is the vCUDA which allows applications executing within VMs to leverage GPU hardware acceleration.

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. The most popular approach is to deploy high level language (HLL)VMs.

2.6.2 VMM Design Requirements and Providers

Hardware-level virtualization inserts a layer between real hardware and traditional operating systems. This layer is commonly called the Virtual Machine Monitor (VMM) and it manages the hardware resources of a computing system. Each time programs access the hardware the VMM captures the process. VMM acts as a traditional OS.

One hardware component, such as the CPU, can be virtualized as several virtual copies. Therefore, several traditional operating systems which are the same or different can sit on the same set of hardware simultaneously.

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

Table 3.2 Comparison of Four VMM and Hypervisor Software Packages

Provider and References	Host CPU	Host OS	Guest OS	Architecture
VMware Workstation [71]	x86, x86-64	Windows, Linux	Windows, Linux, Solaris, FreeBSD, Netware, OS/2, SCO, BeOS, Darwin	Full Virtualization
VMware ESX Server [71]	x86, x86-64	No host OS	The same as VMware Workstation	Para-Virtualization
Xen [7,13,42]	x86, x86-64, IA-64	NetBSD, Linux, Solaris	FreeBSD, NetBSD, Linux, Solaris, Windows XP and 2003 Server	Hypervisor
KVM [31]	x86, x86-64, IA-64, S390, PowerPC	Linux	Linux, Windows, FreeBSD, Solaris	Para-Virtualization

2.6.3 Virtualization Support at the OS Level

With the help of VM technology, a new computing mode known as cloud computing is emerging. 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. However, 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. Therefore, to better support cloud computing, a large amount of research and development should be done.

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. From the user's point of view, VEs look like real servers. This means a VE has its own set of processes, file system, user accounts, network interfaces with IP addresses, routing tables, firewall rules, and other personal settings. Although VEs can be customized for different people, they share the same operating system kernel.

Advantages of OS Extensions

(1) VMs at the operating system level have minimal startup/shutdown costs, low resource requirements, and high scalability.

(2) 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:

(1) All OS-level VMs on the same physical machine share a single operating system kernel

(2) 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.

Virtualization on Linux or Windows Platforms

Virtualization support on the Windows-based platform is still in the research stage. The Linux kernel offers an abstraction layer to allow software processes to work with and operate on resources without knowing the hardware details. New hardware may need a new Linux kernel to support. Therefore, different Linux platforms use patched kernels to provide special support for extended functionality.

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

2.6.4 Middleware Support for Virtualization

Library-level virtualization is also known as user-level Application Binary Interface(ABI) or API emulation. This type of virtualization can create execution environments forrunning 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. This provides an overview of several library-level virtualization systems: namely the Windows Application Binary Interface (WABI), lxrun, WINE, Visual MainWin, and Vcuda.

Table 3.4 Middleware and Library Support for Virtualization

Middleware or Runtime Library and References or Web Link	Brief Introduction and Applic
WABI (http://docs.sun.com/app/docs/doc/802-6306)	Middleware that conver running on x86 P running on S
Lxrun (Linux Run) (http://www.ugcs.caltech.edu/~steven/lxrun/)	A syst
WINE (http://www.winehq.org/)	
Visual MainWin (http://www.	
vCUDA (Exam	

2.3 Virtualization Structures/Tools and Mechanisms

There are three typical classes of VM architecture. Before virtualization, the operating system manages the hardware. After virtualization, a virtualization layer is inserted between the hardware and the operating system. In such a case, the virtualization layer is responsible for converting portions of the real hardware into virtual hardware. Therefore, different operating systems such as Linux and Windows can run on the same physical machine, simultaneously.

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). They both perform the same virtualization operations.

2.7.1 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 monolithic hypervisor architecture like the VMware ESX for server virtualization.

A micro-kernel hypervisor includes only the basic and unchanging functions (such as physical memory management and processor scheduling). The device drivers and other changeable components are outside the hypervisor. A monolithic hypervisor implements all the aforementioned functions, including those of the device drivers.

Therefore, the size of the hypervisor code of a micro-kernel hypervisor is smaller than that of a monolithic hypervisor. Essentially, a hypervisor must be able to convert physical devices into virtual resources dedicated for the deployed VM to use.

The Xen Architecture:

The core components of a Xen system are the hypervisor, kernel, and applications. The organization of the three components is important. Like other virtualization systems, many guest OSes can run on top of the hypervisor. However, not all guest OSes are created equal, and one in particular controls the others.

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 (the Domain U domains).

2.7.2 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, non virtualizable instructions. The guest OSes and their applications consist of noncritical and critical 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.

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.

Binary Translation of Guest OS Requests Using a VMM :

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.

Figure 2.13 Indirect execution of complex instructions via binary translation of guest OS requests using the VMM plus direct execution of simpleinstructions on the same host.

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. Binary translation employs a code cache to store translated hot instructions toimprove performance, but it increases the cost of memory usage.

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.

2.7.3 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.

The virtualization layer can be inserted at different positions in a machine software stack. However, para-virtualization attempts to reduce the virtualization overhead, and thus improve performance by modifying only the guest OS kernel. The guest operating systems are para-virtualized. 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.

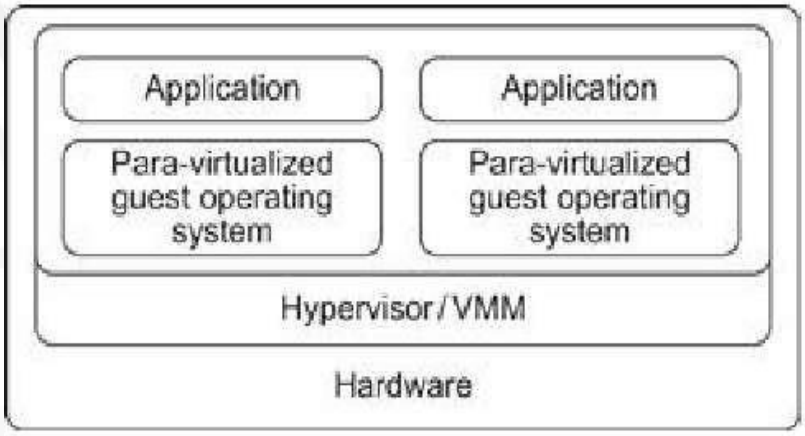


Figure 2.14 Para-virtualized VM architecture

Figure 2.15 The use of a para-virtualized guest OS assisted by an intelligent compiler to replace non virtualizable OS instructions by hyper calls.

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 definitions, the virtualization layer should also be installed at Ring 0. The para-virtualization replaces non virtualizable instructions with hyper calls that communicate directly with the hypervisor or VMM. However, 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. Finally, the performance advantage of para virtualization varies greatly due to workload variations.

KVM (Kernel-Based VM):

This is a Linux para-virtualization system—a part of the Linux version 2.6.20 kernel. 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. Unlike the

full virtualization architecture which intercepts and emulates privileged and sensitive instructions at runtime, para-virtualization handles these instructions at compile time.

The guest OS kernel is modified to replace the privileged and sensitive instructions with hyper calls to the hypervisor or VMM. Xen assumes such a para virtualization architecture. The guest OS running in a guest domain may run at Ring 1instead of at Ring 0. This implies that the guest OS may not be able to execute some privileged and sensitive instructions. The privileged instructions are implemented by hypercalls to the hypervisor. After replacing the instructions with hyper calls, the modified guest OS emulates the behavior of the original guest OS.

2.4 VIRTUALIZATION OF CPU, MEMORY, AND I/O DEVICES

To support virtualization, processors such as the x86 employ a special running mode and instructions, known as hardware-assisted virtualization. In this way, the VMM and guest OS run in different modes and all sensitive instructions of the guest OS and its applications are trapped in the VMM. To save processor states, modes witching are completed by hardware. For the x86architecture, Intel and AMD have proprietary technologies for hardware-assisted virtualization.

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.

CPU Virtualization:

A VM is a duplicate of an existing computer system in which a majority of the VM instructions are executed on the host processor in native mode. Thus, unprivileged instructions of VMs run directly on the host machine for higher efficiency. Other critical instructions should be handled carefully for correctness and stability. The critical instructions are divided into three categories:

Privileged instructions - Privileged instructions execute in a privileged mode and will be trapped if executed outside this mode.

Control sensitive instructions - Control-sensitive instructions attempt to change the configuration of resources used.

Behavior-sensitive instructions - Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.

A CPU architecture is virtualizable if it supports the ability to run the VM's privileged and privileged 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. RISC CPU architectures can be naturally virtualized because all control- and behavior- sensitive instructions are privileged instructions.

Hardware-Assisted CPU Virtualization:

This technique attempts to simplify virtualization because full or para virtualization 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.

Memory Virtualization:

Virtual memory virtualization is similar to the virtual memory support provided bymodern 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. Figure 2.16 shows the two-level memory mapping procedure.

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
- Direct I/O

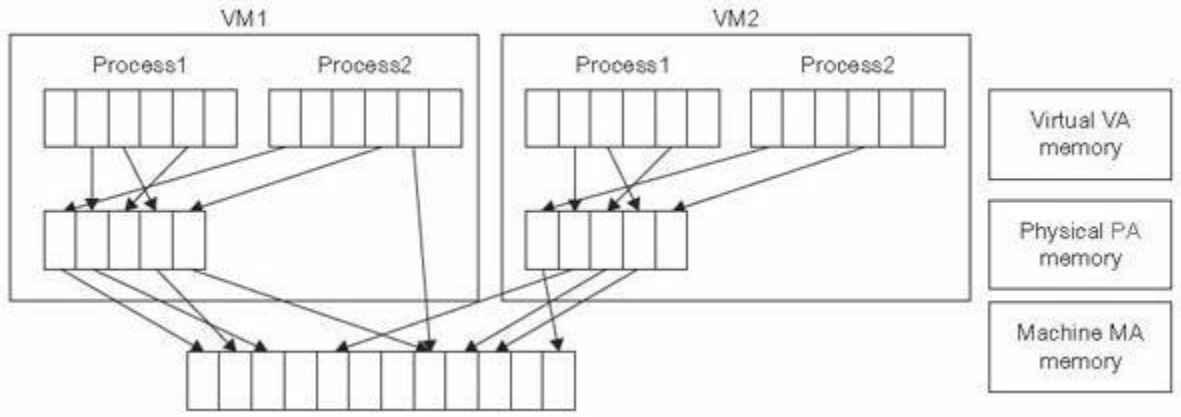


Figure 2.16 Two-level memory mapping procedure.

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.

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

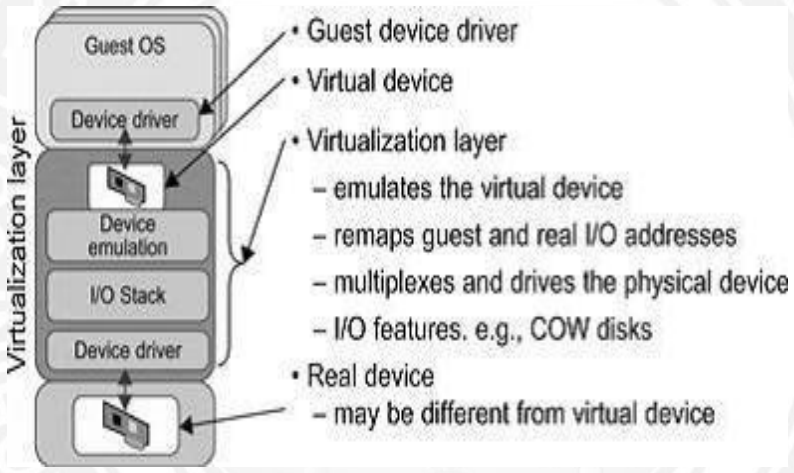


Figure 2.17 Device emulation for I/O virtualization implemented inside the middle layer that maps real I/O devices into the virtual devices for the guest device driver to use.

Virtualization in Multi-Core Processors:

Virtualizing a multi-core processor is relatively more complicated than virtualizing a unicore processor. Though multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip, muti-core virtualization 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.

2.5 Virtualization Support and Disaster Recover:

One very distinguishing feature of cloud computing infrastructure is the use of system virtualization and the modification to provisioning tools. Virtualizations of servers on a shared cluster can consolidate web services. As the VMs are the containers of cloud services, the provisioning tools will first find the corresponding physical machines and deploy the VMs to those nodes before scheduling the service to run on the virtual nodes.

In addition, in cloud computing, virtualization also means the resources and fundamental infrastructure are virtualized. The user will not care about the computing resources that are used for pro-viding the services. Cloud users do not need to know and have no way to discover physical resources that are involved while processing a service request.

Also, application developers do not care about some infrastructure issues such as scalability and fault tolerance (i.e., they are virtualized). Application developers focus on service logic. Figure 2.18 shows the infrastructure needed to virtualize the servers in a data center for implementing specific cloud applications.

2.9.1 Hardware Virtualization

In many cloud computing systems, virtualization software is used to virtualize the hardware. System virtualization software is a special kind of software which simulates the execution of hardware and runs even unmodified operating systems. Cloud computing systems use virtualization software as the running environment for legacy software such as old operating systems and unusual applications. Virtualization software is also used as the platform for developing new cloud applications that enable developers to use any operating systems and programming environments they like.

The development environment and deployment environment can now be the same, which eliminates some runtime problems. Some cloud computing providers have used virtualization technology to provide this service for developers. As mentioned before, system virtualization software is considered the hardware analog mechanism to run an unmodified operating system, usually on bare hardware directly, on top of software. Table 4.4 lists some of the system virtualization software in wide use at the time of this writing. Currently, the VMs installed on a cloud computing platform are mainly used for hosting third-party programs. VMs provide flexible runtime services to free users from worrying about the system environment

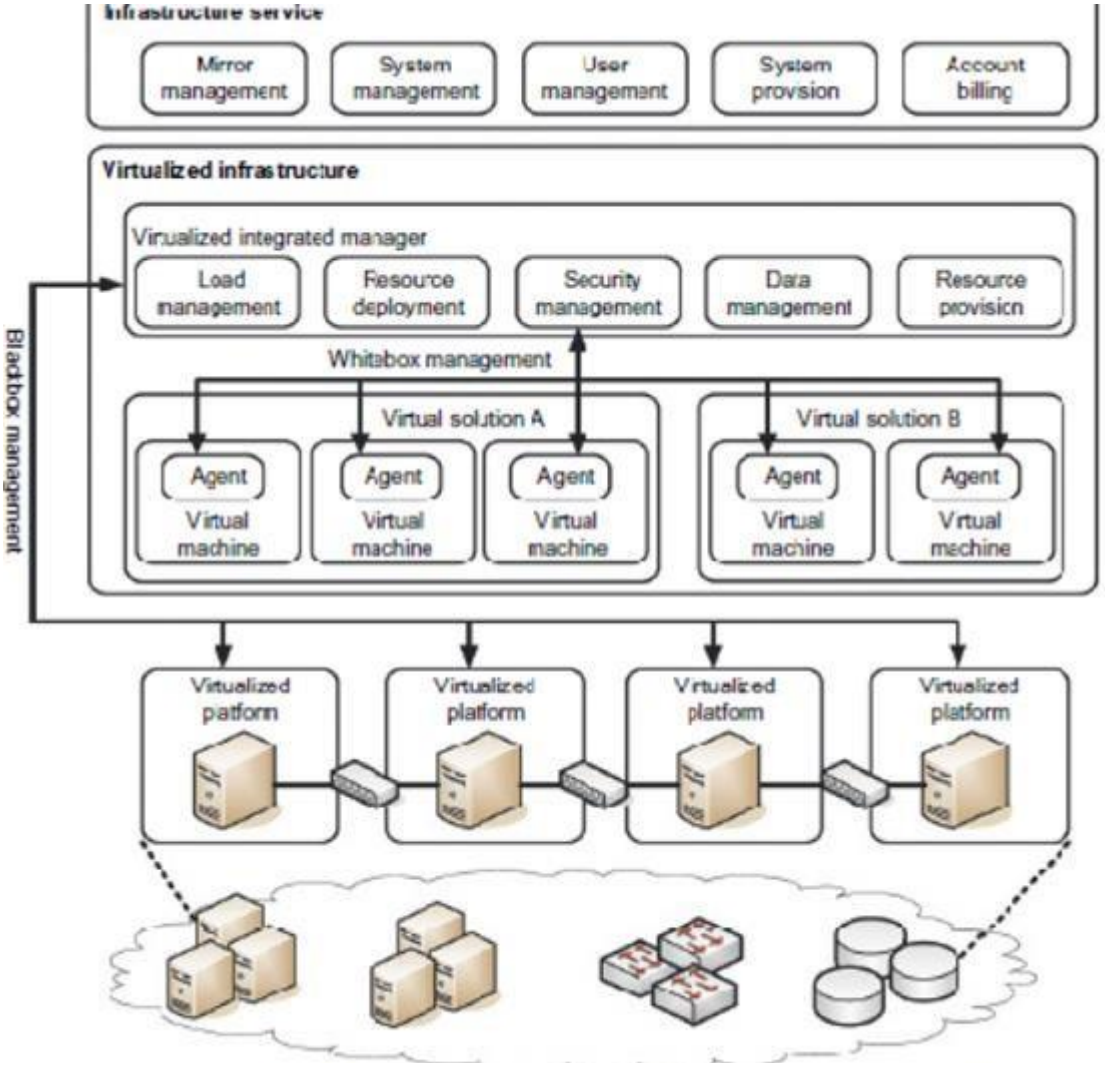
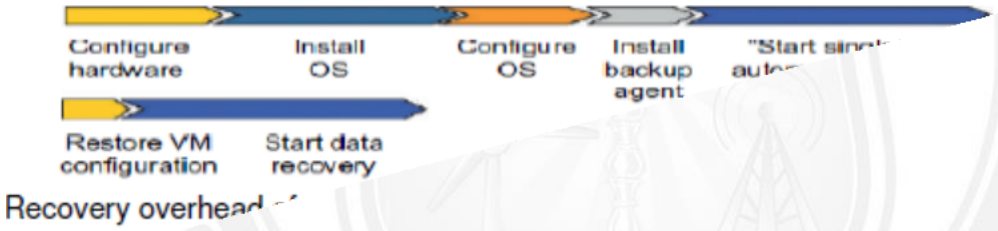


Figure 2.18 Virtualized Storage server and Networks

Using VMs in a cloud computing platform ensures extreme flexibility for users. As the computing resources are shared by many users, a method is required to maximize the users' privileges and still keep them separated safely. Traditional sharing of cluster resources depends on the user and group mechanism on a system. Such sharing is not flexible. Users cannot customize the system for their special purposes. Operating systems cannot be changed. The separation is not complete.

Table 4.4 Virtualized Resources in Compute, Storage, and Network Clouds [4]

Provider	AWS	Microsoft Azure	GAE
Compute cloud with virtual cluster of servers	x86 instruction set, Xen VMs, resource elasticity allows scalability through virtual cluster, or a third party such as RightScale must provide the cluster	Common language runtime VMs provisioned by declarative descript	Predefi fr
Storage cloud with virtual storage	Models for block store (EBS) and augmented key/blob store (SimpleDB), automati scaling varies from EB fully automatic (S	SQ	
Network cloud services	Declarative plac		



An environment that meets one user's requirements often cannot satisfy another user. Virtualization allows users to have full privileges while keeping them separate. Users have full access to their own VMs, which are completely separate from other users' VMs. Multiple VMs can be mounted on the same physical server. Different VMs may run with different OSes. We also need to establish the virtual disk storage and virtual networks needed by the VMs. The virtualized resources form a resource pool.

The virtualization is carried out by special servers dedicated to generating the virtualized resource pool. The virtualized infrastructure (black box in the middle) is built with many virtualizing integration managers. These managers handle loads, resources, security, data, and provisioning functions.

2.9.2 Virtualization Support in Public Clouds

AWS provides extreme flexibility (VMs) for users to execute their own applications. GAE provides limited application-level virtualization for users to build applications only based on the services that are created by Google. Microsoft provides programming-level virtualization (.NET virtualization) for users to build their applications. The VMware tools apply to workstations, servers, and virtual infrastructure. The Microsoft tools are used on PCs and some special servers. The Xen Enterprise tool applies only to Xen-based servers.

Everyone is interested in the cloud; the entire IT industry is moving toward the vision of the cloud. Virtualization leads to HA, disaster recovery, dynamic load leveling, and rich provisioning support. Both cloud computing and utility computing leverage the benefits of virtualization to provide a scalable and autonomous computing environment.

2.9.3 Storage Virtualization for Green Data Centers

IT power consumption in the United States has more than doubled to 3 percent of the total energy consumed in the country. The large number of data centers in the country has contributed to this energy crisis to a great extent. More than half of the companies in the Fortune 500 are actively implementing new corporate energy policies. Recent surveys from both IDC andGartner confirm the fact that virtualization had a great impact on cost reduction from reduced power consumption in physical computing systems. This alarming situation has made the IT industry become more energy aware.

With little evolution of alternate energy resources, there is an imminent need to con-serve power in all computers. Virtualization and server consolidation have already proven handy in this aspect. Green data centers and benefits of storage virtualization are considered to further strengthen the synergy of green computing.

2.9.4 Virtualization for IaaS

VM technology has increased in ubiquity. This has enabled users to create customized environments atop physical infrastructure for cloud computing.

Use of VMs in clouds has the following distinct benefits:

(1) System administrators consolidate workloads of underutilized servers in fewer servers;

(2) VMs have the ability to run legacy code without interfering with other APIs;

(3) VMs can be used to improve security through creation of sandboxes for running applications with questionable reliability;

(4) Virtualized cloud platforms can apply performance isolation, letting providers offer some guarantees and better QoS to customer applications.

2.9.5.5 VM Cloning for Disaster Recovery

VM technology requires an advanced disaster recovery scheme. One scheme is torecover one physical machine by another physical machine. The second scheme is to recover oneVM by another VM. Traditional disaster recovery from one physical machine to another is ratherslow, complex, and expensive. Total recovery time is attributed to the hardware configuration, installing and configuring the OS, installing the backup agents, and the long time to restart the physical machine. To recover a VM platform, the installation and configuration times for the OS and backup agents are eliminated.

Therefore, we end up with a much shorter disaster recovery time, about 40 percent of that to recover the physical machines. Virtualization aids in fast disaster recovery by VM encapsulation. The cloning of VMs offers an effective solution. The idea is to make a clone VM on a remote server for every running VM on a local server. Among all the clone VMs, only one needs to be active. The remote VM should be in a suspended mode.

A cloud control center should be able to activate this clone VM in case of failure of the original VM, taking a snapshot of the VM to enable live migration in a minimal amount of time. The migrated VM can run on a shared Internet connection.

Only updated data and modified states are sent to the suspended VM to update its state. The Recovery Property Objective (RPO) and Recovery Time Objective (RTO) are affected by the number of snapshots taken. Security of the VMs should be enforced during live migration of VMs.