Module 2: VIRTUALIZATION

Virtualization is meant to provide an abstract environment—whether virtual hardware or an operating system—to run applications. The term *virtualization* is often synonymous with *hardware virtualization*, which plays a fundamental role in efficiently delivering *Infrastructure-as-a-Service* (IaaS) solutions for cloud computing. In fact, virtualization technologies are available in many flavors by providing virtual environments at the operating system level, the programming language level, and the application level. Moreover, virtualization technologies provide a virtual environment for not only executing applications but also for storage, memory, and networking.

Virtualization technologies have gained renewed interested recently due to the confluence of several phenomena:

- ***Increased performance and computing capacity***. Nowadays, the average end-user desktop PC is powerful enough to meet almost all the needs of everyday computing, with extra capacity that is rarely used. Almost all these PCs have resources enough to host a virtual machine manager and execute a virtual machine with by far acceptable performance. The same consideration applies to the high-end side of the PC market, where supercomputers can provide immense compute power that can accommodate the execution of hundreds or thousands of virtual machines.
- ***Underutilized hardware and software resources***. Hardware and software underutilization is occurring due to

  (1) increased performance and computing capacity, and

  (2) the effect of limited or sporadic use of resources.

  Computers today are so powerful that in most cases only a fraction of their capacity is used by an application or the system. Moreover, if we consider the IT infrastructure of an enterprise, many computers are only partially utilized whereas they could be used without interruption on a 24*7*365 basis. For example, desktop PCs mostly devoted to office automation tasks and used by administrative staff are only used during work hours, remaining completely unused overnight. Using these resources for other purposes after hours could improve the efficiency of the IT infrastructure. To transparently provide such a service, it would be necessary to deploy a completely separate environment, which can be achieved through virtualization.
- ***Lack of space.*** The continuous need for additional capacity, whether storage or compute power, makes data centers grow quickly. Companies such as Google and Microsoft expand their infrastructures by building data centers as large as football fields that are able to host thousands of nodes. Although this is viable for IT giants, in most cases enterprises cannot afford to build another data center to accommodate additional resource capacity. This condition, along with hardware underutilization, has led to the diffusion of a technique called *server consolidation*, for which virtualization technologies are fundamental.

- ***Greening initiatives.*** Recently, companies are increasingly looking for ways to reduce the amount of energy they consume and to reduce their carbon footprint. Data centers are one of the major power consumers; Maintaining a data center operation not only involves keeping servers on, but a great deal of energy is also consumed in keeping them cool. Infrastructures for cooling have a significant impact on the carbon footprint of a data center. Hence, reducing the number of servers through server consolidation will definitely reduce the impact of cooling and power consumption of a data center. Virtualization technologies can provide an efficient way of consolidating servers.
- ***Rise of administrative costs.*** *The* increased demand for additional capacity, which translates into more servers in a data center, is also responsible for a significant increment in administrative costs. Computers—in particular, servers—do not operate all on their own, but they require care and feeding from system administrators. Common system administration tasks include hardware monitoring, defective hardware replacement, server setup and updates, server resources monitoring, and backups. These are labor-intensive operations, and the higher the number of servers that have to be managed, the higher the administrative costs. Virtualization can help reduce the number of required servers for a given workload, thus reducing the cost of the administrative personnel.

Characteristics of virtualized environments

Virtualization is a broad concept that refers to the creation of a virtual version of something, whether hardware, a software environment, storage, or a network. In a virtualized environment there are three major components: *guest*, *host*, and *virtualization layer*. The *guest* represents the system component that interacts with the virtualization layer rather than with the host, as would normally happen. The *host* represents the original environment where the guest is supposed to be managed. The *virtualization layer* is responsible for recreating the same or a different environment where the guest will operate (see Figure 2.1).

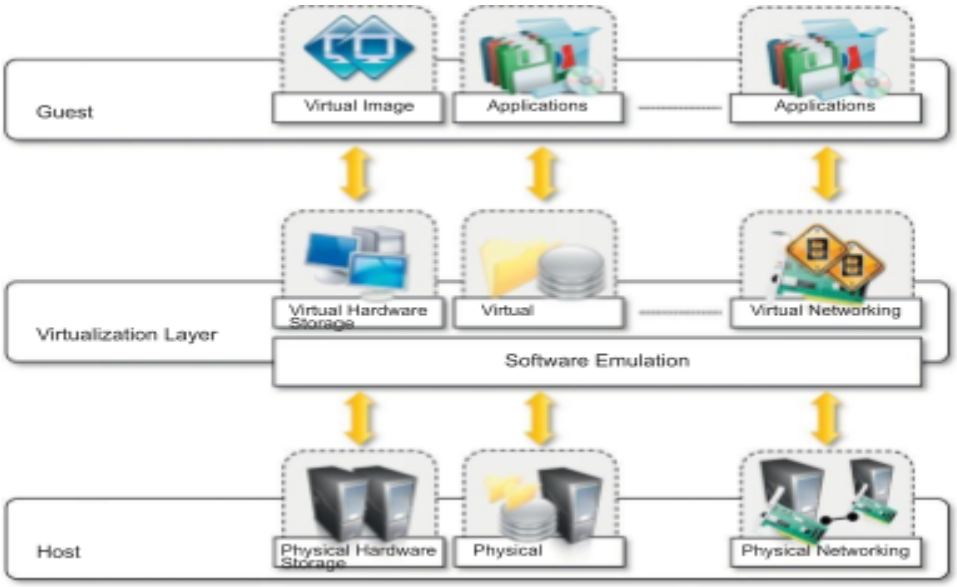

Figure 2.1: The virtualization reference model.

The Characteristics of Virtualization is as follows

1. **Increased security: The** ability to control the execution of a guest in a completely transparent manner opens new possibilities for delivering a secure, controlled execution environment. The virtual machine represents an emulated environment in which the guest is executed. All the operations of the guest are generally performed against the virtual machine, which then translates andapplies them to the host. Resources exposed by the host can then be hidden or simply protected from the guest. Sensitive information that is contained in the host can be naturally hidden without the need to install complex security policies.
2. **Managed execution:** Virtualization of the execution environment not only allows increased security, but a wider range of features also can be implemented. In particular, *sharing*, *aggregation*, *emulation*, and *isolation* are the most relevant features

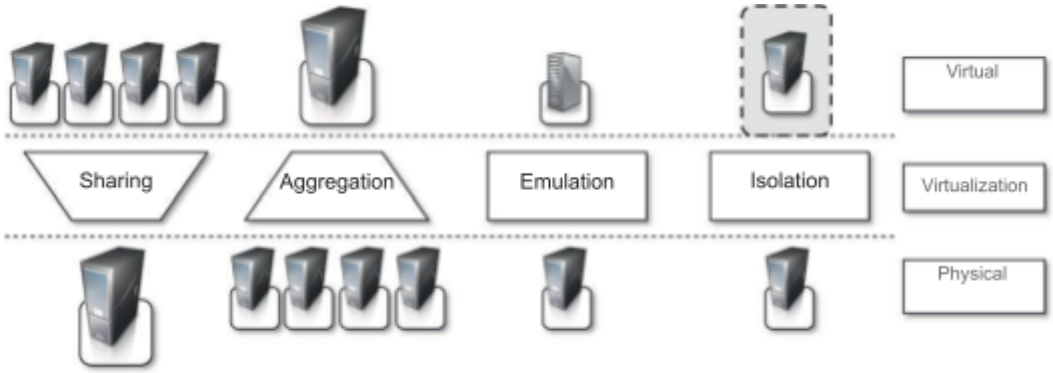

Figure 2.2 : Functions enabled by managed execution

- *Aggregation.* Not only is it possible to share physical resource among several guests, but virtualization also allows aggregation, which is the opposite process. A group of separate hosts can be tied together and represented to guests as a single virtual host. This function is naturally implemented in middleware for distributed computing, with a classical example represented by cluster management software, which harnesses the physical resources of a homogeneous group of machines and represents them as a single resource.

- *Emulation.* Guest programs are executed within an environment that is controlled by the virtualization layer, which ultimately is a program. This allows for controlling and tuning the environment that is exposed to guests. For instance, a completely different environment with respect to the host can be emulated, thus allowing the execution of guest programs requiring specific characteristics that are not present in the physical host. Hardware virtualization solutions are able to provide virtual hardware and emulate a particular kind of device such as *Small Computer System Interface (SCSI)* devices for file I/O, without the hosting

machine having such hardware installed.

- *Isolation.* Virtualization allows providing guests—whether they are operating systems, applications, or other entities—with a completely separate environment, in which they are executed. The guest program performs its activity by interacting with an abstraction layer, which provides access to the underlying resources. The virtual machine can filter the activity of the guest and prevent harmful operations against the host.

3. **Portability:** The concept of *portability* applies in different ways according to the specific type of virtualization considered. In the case of a hardware virtualization solution, the guest is packaged into a virtual image that, in most cases, can be safely moved and executed on top of different virtual machines. Except for the file size, this happens with the same simplicity with which we can display a picture image in different computers. Virtual images are generally proprietary formats that require a specific virtual machine manager to be executed.

Taxonomy of virtualization techniques

Virtualization covers a wide range of emulation techniques that are applied to different areas of computing. A classification of these techniques helps us better understand their characteristics and use (see Figure 2.3).

The first classification discriminates against the service or entity that is being emulated. Virtualization is mainly used to emulate *execution environments*, *storage*, and *networks*. Among these categories, *execution virtualization* constitutes the oldest, most popular, and most developed area. In particular we can divide these execution virtualization techniques into two major categories by considering the type of host they require.

Process-level techniques are implemented on top of an existing operating system, which has full control of the hardware. *System-level* techniques are implemented directly on hardware and do not require—or require a minimum of support from—an existing operating system. Within these two categories we can list various techniques that offer the guest a different type of virtual computation environment: bare hardware, operating system resources, low-level programming language, and application libraries.

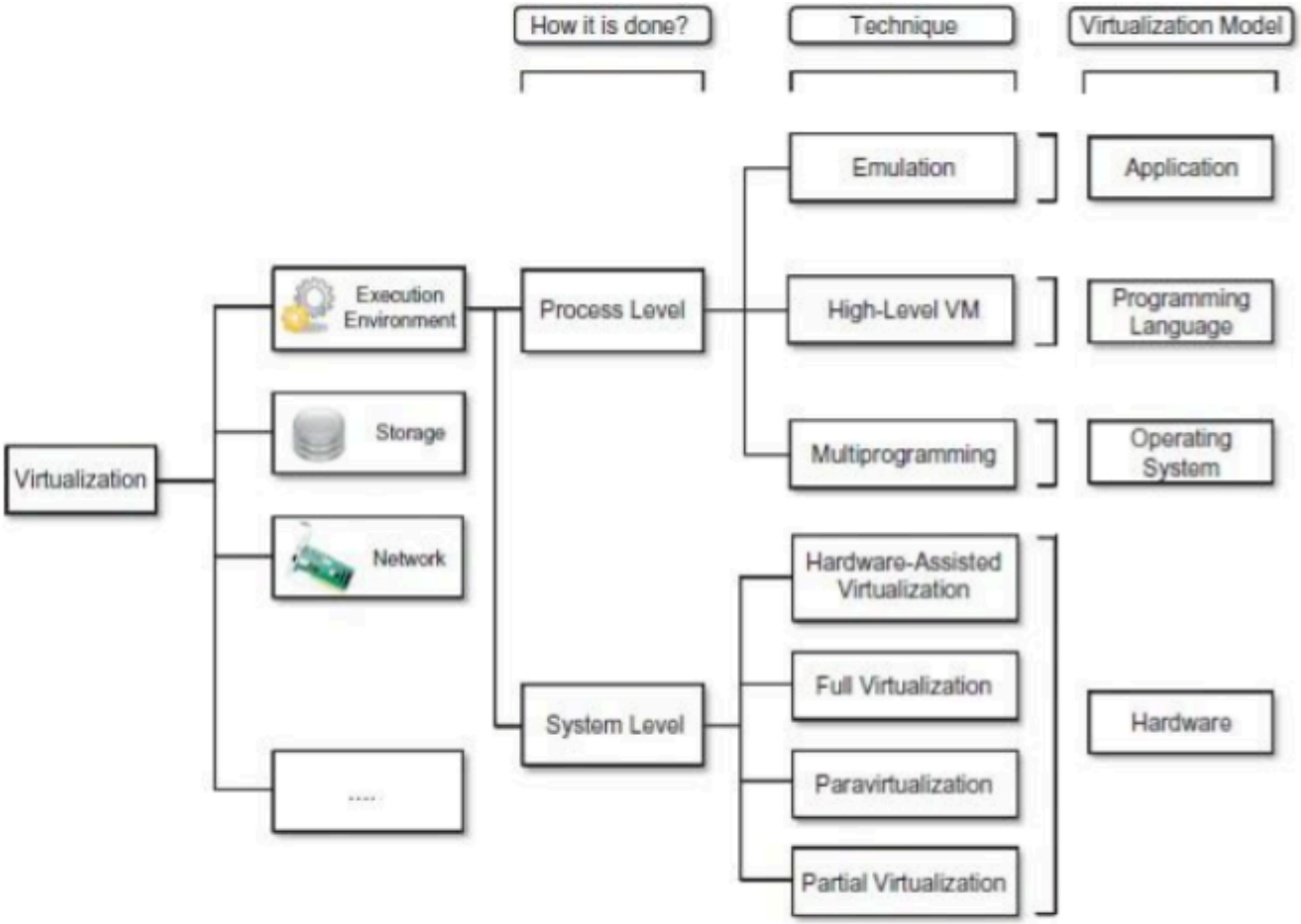

Figure 2.3 A taxonomy of virtualization techniques

Execution virtualization

Execution virtualization includes all techniques that aim to emulate an execution environment that is separate from the one hosting the virtualization layer. All these techniques concentrate their interest on providing support for the execution of programs, whether these are the operating system, a binary specification of a program compiled against an abstract machine model, or an application. Therefore, execution virtualization can be implemented directly on top of the hardware by the operating system, an application, or libraries dynamically or statically linked to an application image.

Machine reference model

Virtualzing an execution environment at different levels of the computing stack requires a reference model that defines the interfaces between the levels of abstractions, which hide implementation details. From this perspective, virtualization techniques actually replace one of the layers and intercept the calls that are directed toward it.

Modern computing systems can be expressed in terms of the reference model described in Figure

2.4. At the bottom layer, the model for the hardware is expressed in terms of the *Instruction Set Architecture (ISA)*, which defines the instruction set for the processor, registers, memory, and interrupts management. ISA is the interface between hardware and software, and it is important to the operating system (OS) developer (*System ISA*) and developers of applications that directly manage the underlying hardware (*User ISA*).

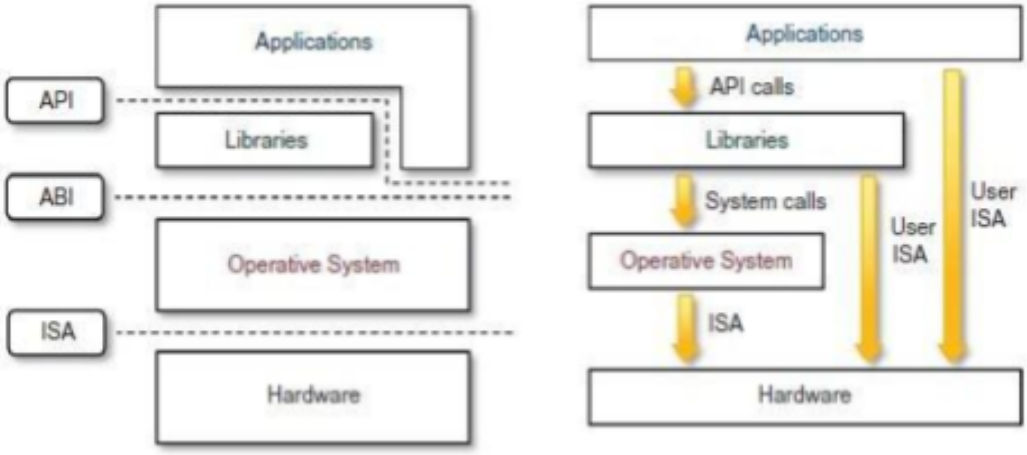

Figure 2.4 A Machine Reference Model

The *application binary interface (ABI)* separates the operating system layer from the applications and libraries, which are managed by the OS. ABI covers details such as low-level data types, alignment, and call conventions and defines a format for executable programs. System calls are defined at this level. This interface allows portability of applications and libraries across operating systems that implement the same ABI. The highest level of abstraction is represented by the *application programming interface (API)*, which interfaces applications to libraries and/or the underlying operating system.

The instruction set exposed by the hardware has been divided into different security classes that define who can operate with them. The first distinction can be made between *privileged* and *nonprivileged* instructions. Nonprivileged instructions are those instructions that can be used without interfering with other tasks because they do not access shared resources. This category contains, for example, all the floating, fixed-point, and arithmetic instructions. Privileged instructions are those that are executed under specific restrictions and are mostly used for sensitive operations, which expose (*behavior-sensitive*) or modify (*control-sensitive*) the privileged state. For instance, behavior-sensitive instructions are those that operate on the I/O, whereas control-sensitive instructions alter the state of the CPU registers.

A possible implementation features a hierarchy of privileges (see Figure 2.5) in the form of ring-based security: *Ring 0*, *Ring 1*, *Ring 2*, and *Ring 3*; Ring 0 is in the most privileged level and Ring 3 in the least privileged level. Ring 0 is used by the kernel of the OS, rings 1 and 2 are used by the OS-level services, and Ring 3 is used by the user. Recent systems support only two levels, with Ring 0 for supervisor mode and Ring 3 for user mode.

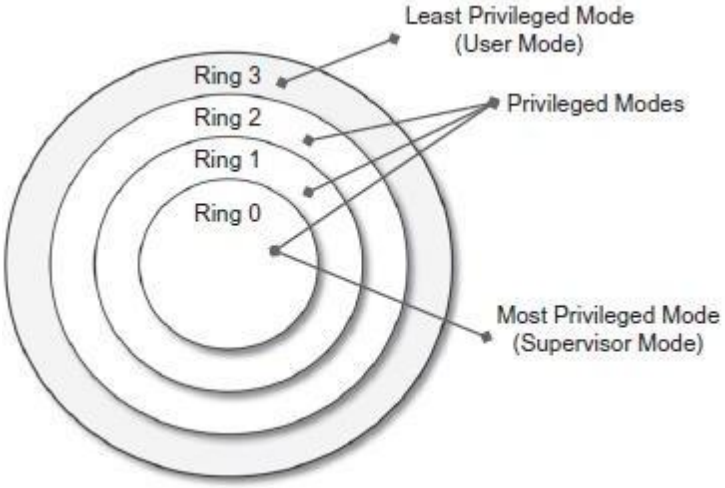

Figure 2.5 Security Rings and Privileged mode

All the current systems support at least two different execution modes: *supervisor mode* and *user mode*. The first mode denotes an execution mode in which all the instructions (privileged and nonprivileged) can be executed without any restriction. This mode, also called *master mode* or *kernel mode*, is generally used by the operating system (or the hypervisor) to perform sensitive operations on hardware level resources. In user mode, there are restrictions to control the machine-level resources. If code running in user mode invokes the privileged instructions, hardware interrupts occur and trap the potentially harmful execution of the instruction. Conceptually, the hypervisor runs above the supervisor mode.

Hardware-level virtualization

Hardware-level virtualization is a virtualization technique that provides an abstract execution environment in terms of computer hardware on top of which a guest operating system can be run. In this model, the guest is represented by the operating system, the host by the physical computer hardware, the virtual machine by its emulation, and the virtual machine manager by the hypervisor (see Figure 2.6). The hypervisor is generally a program or a combination of software and hardware that allows the abstraction of the underlying physical hardware.

Hardware-level virtualization is also called *system virtualization*, since it provides ISA to virtual machines, which is the representation of the hardware interface of a system.

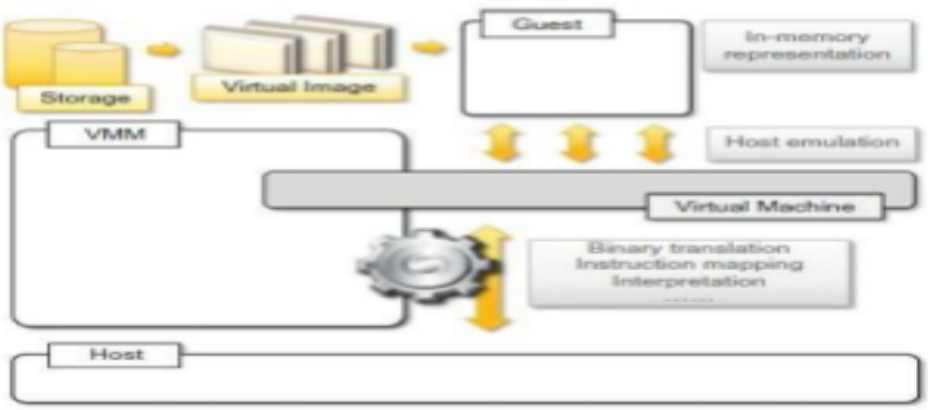

Figure 2.6 A hardware virtualization reference model.

Hypervisors: A fundamental element of hardware virtualization is the hypervisor, or virtual machine manager (VMM).
It recreates a hardware environment in which guest operating systems are installed. There are two major types of hypervisor: *Type I* and *Type II* (see Figure 2.7).

Type I hypervisors run directly on top of the hardware. Therefore, they take the place of the operating systems and interact directly with the ISA interface exposed by the underlying hardware, and they emulate this interface in order to allow the management of guest operating systems. This type of hypervisor is also called a *native virtual machine* since it runs natively on hardware.

- *Type II* hypervisors require the support of an operating system to provide virtualization services. This means that they are programs managed by the operating system, which interact with it through the ABI and emulate the ISA of virtual hardware for guest operating systems. This type of hypervisor is also called a *hosted virtual machine* since it is hosted within an operating system.

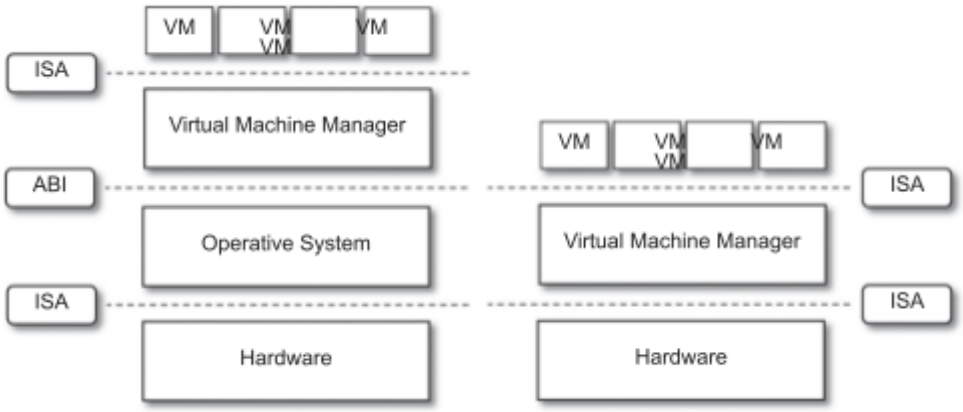

Figure 2.7 Hosted (left) and native (right) virtual machines.

A virtual machine manager is internally organized as described in Figure 2.8. Three main modules, *dispatcher*, *allocator*, and *interpreter*, coordinate their activity in order to emulate the underlying hardware. The dispatcher constitutes the entry point of the monitor and reroutes the instructions issued by the virtual machine instance to one of the two other modules. The allocator is responsible for deciding the system resources to be provided to the VM: whenever a virtual machine tries to execute an instruction that results in changing the machine resources associated with that VM, the allocator is invoked by the dispatcher. The interpreter module consists of interpreter routines. These are executed whenever a virtual machine executes a privileged instruction: a trap is triggered and the corresponding routine is executed.

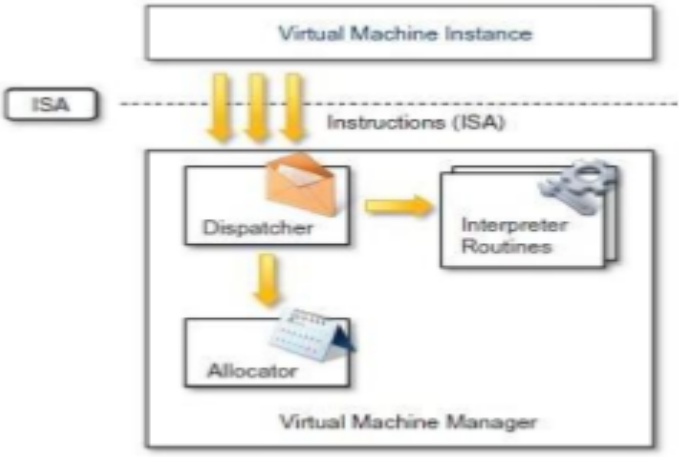

Figure 2.8 A hypervisor reference architecture. Three properties of Virtual Machine Manager that have to be satisfied:

- *Equivalence.* A guest running under the control of a virtual machine manager should exhibit the same behavior as when it is executed directly on the physical host.
- *Resource control.* The virtual machine manager should be in complete control of virtualized resources.
- *Efficiency.* A statistically dominant fraction of the machine instructions should be executed without intervention from the virtual machine manager.

Popek and Goldberg provided a classification of the instruction set and proposed three theorems that define the properties that hardware instructions need to satisfy in order to efficiently support virtualization

THEOREM 1

For any conventional third-generation computer, a VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.

This theorem establishes that all the instructions that change the configuration of the system resources should generate a trap in user mode and be executed under the control of the virtual machine manager. This allows hypervisors to efficiently control only those instructions that would reveal the presence of an abstraction layer while executing all the rest of the instructions without considerable performance loss.

THEOREM 2

A conventional third-generation computer is recursively virtualizable if:

- **It is virtualizable and**

- **A VMM without any timing dependencies can be constructed for it.**

Recursive virtualization is the ability to run a virtual machine manager on top of another virtual machine manager. This allows nesting hypervisors as long as the capacity of the underlying resources can accommodate that. Virtualizable hardware is a prerequisite to recursive virtualization.

THEOREM 3

A hybrid VMM may be constructed for any conventional third-generation machine in which the set of user-sensitive instructions is a subset of the set of privileged instructions.

There is another term, *hybrid virtual machine (HVM)*, which is less efficient than the virtual machine system. In the case of an HVM, more instructions are interpreted rather than being executed directly. All instructions in virtual supervisor mode are interpreted. Whenever there is an attempt to execute a behavior-sensitive or control-sensitive instruction, HVM controls the execution directly or gains the control via a trap.

Hardware virtualization techniques

Hardware-assisted virtualization. This term refers to a scenario in which the hardware provides architectural support for building a virtual machine manager able to run a guest operating system in complete isolation. This technique was originally introduced in the IBM System/370. At present, examples of hardware-assisted virtualization are the extensions to the x86-64 bit architecture introduced with *Intel VT* (formerly known as *Vanderpool*) and *AMD V* (formerly known as *Pacifica*).. Products such as VMware Virtual Platform, introduced in 1999 by VMware, which pioneered the field of x86 virtualization, were based on this technique. After 2006, Intel and AMD introduced processor extensions, and a wide range of virtualization solutions took advantage of them*:* Kernel-based Virtual Machine (KVM), VirtualBox, Xen, VMware, Hyper-V, Sun xVM, Parallels, and others.

Full virtualization. *Full virtualization* refers to the ability to run a program, most likely an operating system, directly on top of a virtual machine and without any modification, as though it were run on the raw hardware. To make this possible, virtual machine managers are required to provide a complete emulation of the entire underlying hardware. The principal advantage of full virtualization is complete isolation, which leads to enhanced security, ease of emulation of different architectures, and coexistence of different systems on the same platform. A simple solution to achieve full virtualization is to provide a virtual environment for all the instructions, thus posing some limits on performance.

Paravirtualization. This is a not-transparent virtualization solution that allows implementing thin virtual machine managers. Paravirtualization techniques expose a software interface to the virtual machine that is slightly modified from the host and, as a consequence, guests need to be modified. The aim of paravirtualization is to provide the capability to demand the execution of performance-critical operations directly on the host, thus preventing performance losses that would otherwise be experienced in managed execution. This technique has been successfully used by Xen for providing virtualization solutions for Linux-based operating systems specifically

ported to run on Xen hypervisors.

Partial virtualization. Partial virtualization provides a partial emulation of the underlying hardware, thus not allowing the complete execution of the guest operating system in complete isolation. Partial virtualization allows many applications to run transparently, but not all the features of the operating system can be supported, as happens with full virtualization. Partial virtualization was implemented on the experimental IBM M44/44X. Address space virtualization is a common feature of contemporary operating systems.

Operating system-level virtualization

Operating system-level virtualization offers the opportunity to create different and separated execution environments for applications that are managed concurrently. Differently from hardware virtualization, there is no virtual machine manager or hypervisor, and the virtualization is done within a single operating system, where the OS kernel allows for multiple isolated user space instances. The kernel is also responsible for sharing the system resources among instances and for limiting the impact of instances on each other. A user space instance in general contains a proper view of the file system, which is completely isolated, and separate IP addresses, software configurations, and access to devices. Operating system-level virtualization aims to provide separated and multiple execution containers for running applications. Compared to hardware virtualization, this strategy imposes little or no overhead because applications directly use OS system calls and there is no need for emulation.

Examples of operating system-level virtualizations are FreeBSD Jails, IBM Logical Partition (LPAR), SolarisZones and Containers, Parallels Virtuozzo Containers, OpenVZ, iCore Virtual Accounts, Free Virtual Private Server (FreeVPS).

Programming language-level virtualization

Programming language-level virtualization is mostly used to achieve ease of deployment of applications, managed execution, and portability across different platforms and operating systems. It consists of a virtual machine executing the byte code of a program, which is the result of the compilation process. Compilers implemented and used this technology to produce a binary format representing the machine code for an abstract architecture. The characteristics of this architecture vary from implementation to implementation.

Programming language-level virtualization has a long trail in computer science history and originally was used in 1966 for the implementation of Basic Combined Programming Language (BCPL), a language for writing compilers and one of the ancestors of the C programming language. Other important examples of the use of this technology have been the UCSD Pascal and Smalltalk. Virtual machine programming languages become popular again with Sun's introduction of the Java platform in 1996.

Currently, the Java platform and .NET Framework represent the most popular technologies for enterprise application development. The main advantage of programming-level virtual machines, also called *process virtual machines*, is the ability to provide a uniform execution environment

across different platforms. Programs compiled into byte code can be executed on any operating system and platform for which a virtual machine able to execute that code has been provided.

Application-level virtualization

Application-level virtualization is a technique allowing applications to be run in runtime environments that do not natively support all the features required by such applications. In this scenario, applications are not installed in the expected runtime environment but are run as though they were. In general, these techniques are mostly concerned with partial file systems, libraries, and operating system component emulation. Such emulation is performed by a thin layer—a program or an operating system component—that is in charge of executing the application. Emulation can also be used to execute program binaries compiled for different hardware architectures. In this case, one of the following strategies can be implemented:

- **Interpretation.** In this technique every source instruction is interpreted by an emulator for executing native ISA instructions, leading to poor performance. Interpretation has a minimal startup cost but a huge overhead, since each instruction is emulated.
- **Binary translation.** In this technique every source instruction is converted to native instructions with equivalent functions. After a block of instructions is translated, it is cached and reused. Binary translation has a large initial overhead cost, but over time it is subject to better performance, since previously translated instruction blocks are directly executed.

Application virtualization is a good solution in the case of missing libraries in the host operating system. One of the most popular solutions implementing application virtualization is Wine, which is a software application allowing Unix-like operating systems to execute programs written for the Microsoft Windows platform

Other types of virtualization

1. Storage virtualization

Storage virtualization is a system administration practice that allows decoupling the physical organization of the hardware from its logical representation. Using this technique, users do not have to be worried about the specific location of their data, which can be identified using a logical path. There are different techniques for storage virtualization, one of the most popular being network-based virtualization by means of *storage area networks (SANs)*.

2. Network virtualization

Network virtualization combines hardware appliances and specific software for the creation and management of a virtual network. Network virtualization can aggregate different physical networks into a single logical network (*external* network virtualization) or provide network-like functionality to an operating system partition (*internal* network virtualization). The result of external network virtualization is generally a *virtual LAN (VLAN)*. A VLAN is an aggregation of hosts that communicate with each other as though they were located under the same

broadcasting domain. There are several options for implementing internal network virtualization: The guest can share the same net- work interface of the host and use Network Address Translation (NAT) to access the network; the virtual machine manager can emulate, and install on the host, an additional network device, together with the driver; or the guest can have a private network only with the guest.

3. Desktop virtualization

Desktop virtualization abstracts the desktop environment available on a personal computer in order to provide access to it using a client/server approach. Desktop virtualization provides the same out- come of hardware virtualization but serves a different purpose. Similarly to hardware virtualization, desktop virtualization makes accessible a different system as though it were natively installed on the host, but this system is remotely stored on a different host and accessed through a network connection. Moreover, desktop virtualization addresses the problem of making the same desktop environment accessible from everywhere. The advantages of desktop virtualization are high availability, persistence, accessibility, and ease of management. Infrastructures for desktop virtua- lization based on cloud computing solutions include Sun Virtual Desktop Infrastructure (VDI), Parallels Virtual Desktop Infrastructure (VDI), Citrix XenDesktop, and others.

4. **Application server virtualization**

Application server virtualization abstracts a collection of application servers that provide the same services as a single virtual application server by using load-balancing strategies and providing a high-availability infrastructure for the services hosted in the application server. This is a particular form of virtualization and serves the same purpose of storage virtualization: providing a better quality of service rather than emulating a different environment.

Virtualization and cloud computing

Virtualization plays an important role in cloud computing since it allows for the appropriate degree of customization, security, isolation, and manageability that are fundamental for delivering IT services on demand. Virtualization technologies are primarily used to offer configurable computing environments and storage.

Particularly important is the role of virtual computing environment and execution virtualization techniques. Among these, hardware and programming language virtualization are the techniques adopted in cloud computing systems. Hardware virtualization is an enabling factor for solutions in the Infrastructure-as-a-Service (IaaS) market segment, while programming language virtualization is a technology leveraged in Platform-as-a-Service (PaaS) offerings. In both cases, the capability of offering a customizable and sandboxed environment constituted an attractive business opportunity for companies featuring a large computing infrastructure that was able to sustain and process huge workloads. Moreover, virtualization also allows isolation and a finer control, thus simplifying the leasing of services and their accountability on the vendor side.

Virtualization allows us to create isolated and controllable environments, it is possible to serve

these environments with the same resource without them interfering with each other. If the underlying resources are capable enough, there will be no evidence of such sharing.It allows reducing the number of active resources by aggregating virtual machines over a smaller number of resources that become fully utilized. This practice is also known as server consolidation, while the movement of virtual machine instances is called virtual machine migration (see Figure 3.10). Because virtual machine instances are controllable environments, consolidation can be applied with a minimum impact, either by temporarily stopping its execution and moving its data to the new resources or by performing a finer control and moving the instance while it is running. This second techniques is known as live migration and in general is more complex to implement but more efficient since there is no disruption of the activity of the virtual machine instance

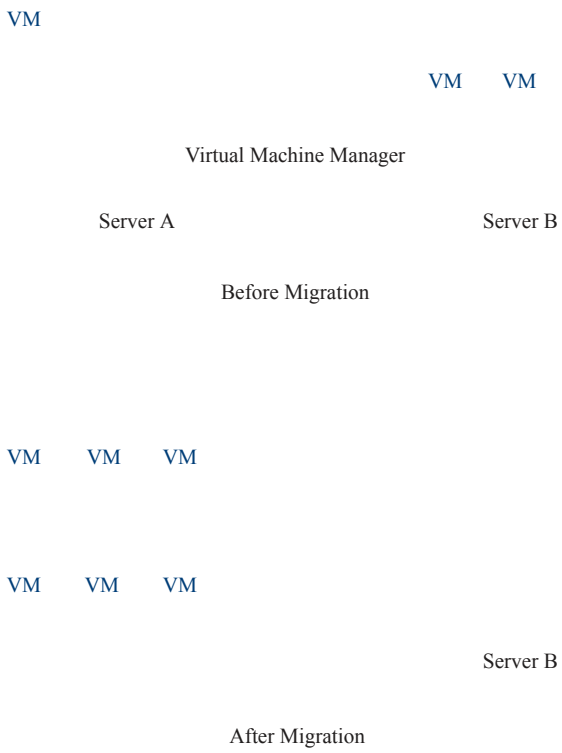

Figure 2.10 Live migration and server consolidation.

Pros and cons of virtualization

Advantages of virtualization

Managed execution and isolation are perhaps the most important advantages of virtualization. In the case of techniques supporting the creation of virtualized execution environments, these two characteristics allow building secure and controllable computing environments. A virtual execution environment can be configured as a sandbox, thus preventing any harmful operation to cross the borders of the virtual host. Moreover, allocation of resources and their partitioning

among different guests is simplified, being the virtual host controlled by a program. This enables fine-tuning of resources, which is very important in a server consolidation scenario and is also a requirement for effective quality of service

Portability and self-containment also contribute to reducing the costs of maintenance, since the number of hosts is expected to be lower than the number of virtual machine instances. By means of virtualization it is possible to achieve a more efficient use of resources. Multiple systems can securely coexist and share the resources of the underlying host, without interfering with each other.

Performance degradation

Performance is definitely one of the major concerns in using virtualization technology. Since virtualization interposes an abstraction layer between the guest and the host, the guest can experience increased latencies.

For instance, in the case of hardware virtualization, where the intermediate emulates a bare machine on top of which an entire system can be installed, the causes of performance degradation can be traced back to the overhead introduced by the following activities:

- Maintaining the status of virtual processors
- Support of privileged instructions (trap and simulate privileged instructions)
- Support of paging within VM
- Console functions

Furthermore, when hardware virtualization is realized through a program that is installed or executed on top of the host operating systems, a major source of performance degradation is represented by the fact that the virtual machine manager is executed and scheduled together with other applications, thus sharing with them the resources of the host.

Similar considerations can be made in the case of virtualization technologies at higher levels, such as in the case of programming language virtual machines (Java, .NET, and others). Binary translation and interpretation can slow down the execution of managed applications. Moreover, because their execution is filtered by the runtime environment, access to memory and other physi- cal resources can represent sources of performance degradation.

These concerns are becoming less and less important thanks to technology advancements and the ever-increasing computational power available today. For example, specific techniques for hard- ware virtualization such as paravirtualization can increase the performance of the guest program by offloading most of its execution to the host without any change. In programming- level virtual

machines such as the JVM or .NET, compilation to native code is offered as an option when perfor- mance is a serious concern.

Disadvantages:

Inefficiency and degraded user experience

Virtualization can sometimes lead to an inefficient use of the host. In particular, some of the specific features of the host cannot be exposed by the abstraction layer and then become inaccessible. In the case of hardware virtualization, this could happen for device drivers: The virtual machine can sometime simply provide a default graphic card that maps only a subset of the features available in the host. In the case of programming-level virtual machines, some of the features of the underlying operating systems may become inaccessible unless specific libraries are used.

Security holes and new threats

Virtualization opens the door to a new and unexpected form of *phishing.*The capability of emu- lating a host in a completely transparent manner led the way to malicious programs that are designed to extract sensitive information from the guest.

In the case of hardware virtualization, malicious programs can preload themselves before the operating system and act as a thin virtual machine manager toward it. The operating system is then controlled and can be manipulated to extract sensitive information of interest to third parties.

Technology examples

Xen: paravirtualization

Xen is an open-source initiative implementing a virtualization platform based on paravirtualization. Initially developed by a group of researchers at the University of Cambridge in the United Kingdom, Xen now has a large open-source community backing it. . Xen-based technology is used for either desktop virtualization or server virtualization, and recently it has also been used to provide cloud computing solutions by means of Xen Cloud Platform (XCP).

Figure 2.11 describes the architecture of Xen and its mapping onto a classic x86 privilege model. A Xen-based system is managed by the Xen hypervisor, which runs in the highest

privileged mode and controls the access of guest operating system to the underlying hardware.

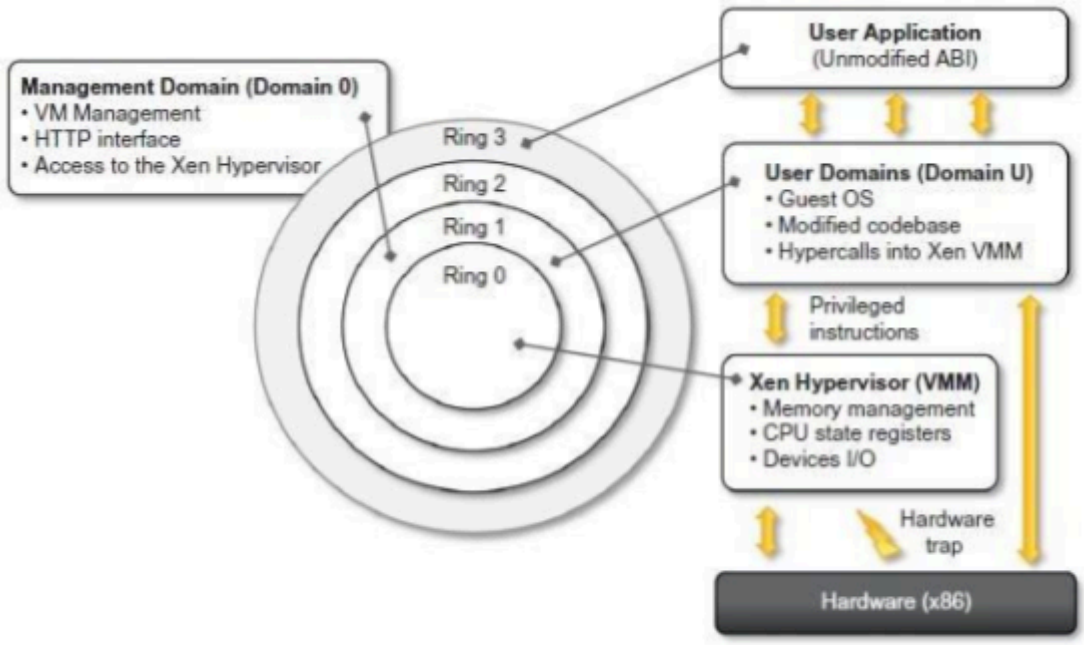

Figure 2.11 Xen architecture and guest OS management

Guest operating systems are executed within domains, which represent virtual machine instances. Moreover, specific control software, which has privileged access to the host and controls all the other guest operating systems, is executed in a special domain called Domain 0. This is the first one that is loaded once the virtual machine manager has completely booted, and it hosts a HyperText Transfer Protocol (HTTP) server that serves requests for virtual machine creation, con- figuration, and termination. This component constitutes the embryonic version of a distributed virtual machine manager, which is an essential component of cloud computing systems providing Infrastructure-as-a-Service (IaaS) solutions.

Many of the x86 implementations support four different security levels, called rings, where Ring 0 represent the level with the highest privileges and Ring 3 the level with the lowest ones.

Because of the structure of the x86 instruction set, some instructions allow code executing in Ring 3 to jump into Ring 0 (kernel mode). Such operation is performed at the hardware level and therefore within a virtualized environment will result in a trap or silent fault, thus preventing

the normal operations of the guest operating system, since this is now running in Ring 1.This condition is generally triggered by a subset of the system calls. To avoid this situation, operating systems need to be changed in their implementation, and the sensitive system calls need to be reimplemented with hypercalls, which are specific calls exposed by the virtual machine interface of Xen. With the use of hypercalls, the Xen hypervisor is able to catch the execution of all the sensitive instructions, manage them, and return the control to the guest operating system by means of a supplied handler.

Paravirtualization needs the operating system codebase to be modified, and hence not all operating systems can be used as guests in a Xen-based environment. Open-source operating systems such as Linux can be easily modified, since their code is publicly available and Xen provides full support for their virtualization, whereas components of the Windows family are generally not supported by Xen unless hardware-assisted virtualization is available.

VMware: full virtualization: VMware's technology is based on the concept of *full virtualization*, where the underlying hardware is replicated and made available to the guest operating system, which runs unaware of such abstrac- tion layers and does not need to be modified. VMware implements full virtualization either in the desktop environment, by means of *Type II* hypervisors, or in the server environment, by means of *Type I* hypervisors. In both cases, full virtualization is made possible by means of *direct execution* (for nonsensitive instructions) and *binary translation* (for sensitive instructions), thus allowing the virtualization of architecture such as x86.

Microsoft Hyper-V

Hyper-V is an infrastructure virtualization solution developed by Microsoft for server virtualization. As the name recalls, it uses a hypervisor-based approach to hardware virtualization, which leverages several techniques to support a variety of guest operating systems.

3.6.3.1 Architecture

Hyper-V supports multiple and concurrent execution of guest operating systems by means of partitions. A partition is a completely isolated environment in which an operating system is installed and run. Figure 3.17r(refer to PPT in site) provides an overview of the architecture of Hyper-V. Despite its straightforward installation as a component of the host operating system, Hyper-V takes control of the hardware, and the host operating system becomes a virtual machine instance with special privileges, called the parent partition.

The parent partition (also called the root partition) is the only one that has direct access to the hardware. It runs the virtualization stack, hosts all the drivers required to configure guest operating systems, and creates child partitions through the hypervisor.

Child partitions are used to host guest operating systems and do not have access to the underlying

hardware,but their interaction with it is controlled by either the parent partition or the hypervisor itself.

Hypervisor

The hypervisor is the component that directly manages the underlying hardware (processors and memory). It is logically defined by the following components:

• Hypercalls interface. This is the entry point for all the partitions for the execution of sensitive instructions. This is an implementation of the paravirtualization approach already discussed withXen. This interface is used by drivers in the partitioned operating system to contact the hypervisor using the standard Windows calling convention. The parent partition also uses this interface to create child partitions.

• Memory service routines (MSRs). These are the set of functionalities that control the memory and its access from partitions. By leveraging hardware-assisted virtualization, the hypervisor uses the Input/Output Memory Management Unit (I/O MMU or IOMMU) to fast-track access to devices from partitions by translating virtual memory addresses.

• Advanced programmable interrupt controller (APIC). This component represents the interrupt controller, which manages the signals coming from the underlying hardware when some event occurs (timer expired, I/O ready, exceptions and traps). Each virtual processor is equipped with a synthetic interrupt controller (SynIC), which constitutes an extension of the local APIC. The hypervisor is responsible of dispatching, when appropriate, the physical interrupts to the synthetic interrupt controllers.

• Scheduler. This component schedules the virtual processors to run on available physical processors. The scheduling is controlled by policies that are set by the parent partition.

• Address manager. This component is used to manage the virtual network addresses that are allocated to each guest operating system.

• Partition manager. This component is in charge of performing partition creation, finalization,destruction, enumeration, and configurations. Its services are available through the hypercalls interface API previously discussed.

The hypervisor runs in Ring -1 and therefore requires corresponding hardware technology that enables such a condition. By executing in this highly privileged mode, the hypervisor can support legacy operating systems that have been designed for x86 hardware. Operating systems of newer generations can take advantage of the new specific architecture of Hyper-V especially for the I/O operations performed by child partitions.

Parent partition

The parent partition executes the host operating system and implements the virtualization stack that complements the activity of the hypervisor in running guest operating systems. This partition always hosts an instance of the Windows Server 2008 R2, which manages the virtualization stack made available to the child partitions. This partition is the only one that directly accesses device drivers and mediates the access to them by child partitions by hosting the VSPs. The parent partition is also the one that manages the creation, execution, and destruction of child partitions. It does so by means of the Virtualization Infrastructure Driver (VID), which controls access to the hypervisor and allows the management of virtual processors and memory. For each child partition created, a Virtual Machine Worker Process (VMWP) is instantiated in the parent partition, which manages the child partitions by interacting with the hypervisor through the VID. Virtual Machine Management services are also accessible remotely through a WMI9 provider that allows remote hosts to access the VID.

Child partitions

Child partitions are used to execute guest operating systems. These are isolated environments that allow secure and controlled execution of guests. Two types of child partition exist, they differ on whether the guest operating system is supported by Hyper-V or not. These are called Enlightened and Unenlightened partitions, respectively. The first ones can benefit from Enlightened I/O; the other ones are executed by leveraging hardware emulation

from the hypervisor.