CHAPTER

Virtualization

3

Virtualization technology is one of the fundamental components of cloud computing, especially in regard to infrastructure-based services. Virtualization allows the creation of a secure, customizable, and isolated execution environment for running applications, even if they are untrusted, without affecting other users' applications. The basis of this technology is the ability of a computer program—or a combination of software and hardware—to emulate an executing environment separate from the one that hosts such programs. For example, we can run Windows OS on top of a virtual machine, which itself is running on Linux OS. Virtualization provides a great opportunity to build elastically scalable systems that can provision additional capability with minimum costs. Therefore, virtualization is widely used to deliver customizable computing environments on demand.

This chapter discusses the fundamental concepts of virtualization, its evolution, and various models and technologies used in cloud computing environments.

3.1 Introduction

Virtualization is a large umbrella of technologies and concepts that are 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 have a long trail in the history of computer science and have been 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.

Since its inception, virtualization has been sporadically explored and adopted, but in the last few years there has been a consistent and growing trend to leverage this technology. 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*,[1] 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; they contribute consistently to the impact that a company has on the environment. 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.* Power consumption and cooling costs have now become higher than the cost of IT equipment. Moreover, 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.

These can be considered the major causes for the diffusion of hardware virtualization solutions as well as the other kinds of virtualization. The first step toward consistent adoption of virtualization technologies was made with the wide spread of virtual machine-based programming languages: In 1995 Sun released Java, which soon became popular among developers. The ability to integrate small Java applications, called *applets*, made Java a very successful platform, and with the

[1]Server consolidation is a technique for aggregating multiple services and applications originally deployed on different servers on one physical server. Server consolidation allows us to reduce the power consumption of a data center and resolve hardware underutilization.

beginning of the new millennium Java played a significant role in the application server market segment, thus demonstrating that the existing technology was ready to support the execution of managed code for enterprise-class applications. In 2002 Microsoft released the first version of .NET Framework, which was Microsoft's alternative to the Java technology. Based on the same principles as Java, able to support multiple programming languages, and featuring complete integration with other Microsoft technologies, .NET Framework soon became the principal development platform for the Microsoft world and quickly became popular among developers. In 2006, two of the three "official languages" used for development at Google, Java and Python, were based on the virtual machine model. This trend of shifting toward virtualization from a programming language perspective demonstrated an important fact: The technology was ready to support virtualized solutions without a significant performance overhead. This paved the way to another and more radical form of virtualization that now has become a fundamental requisite for any data center management infrastructure.

3.2 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 3.1).

Such a general abstraction finds different applications and then implementations of the virtualization technology. The most intuitive and popular is represented by *hardware virtualization*, which also constitutes the original realization of the virtualization concept.[2] In the case of hardware virtualization, the guest is represented by a system image comprising an operating system and installed applications. These are installed on top of virtual hardware that is controlled and managed by the virtualization layer, also called the *virtual machine manager*. The host is instead represented by the physical hardware, and in some cases the operating system, that defines the environment where the virtual machine manager is running. In the case of virtual storage, the guest might be client applications or users that interact with the virtual storage management software deployed on top of the real storage system. The case of virtual networking is also similar: The guest—applications and users—interacts with a virtual network, such as a *virtual private network (VPN)*, which is managed by specific software (VPN client) using the physical network available on the node. VPNs are useful for creating the illusion of being within a different physical network and thus accessing the resources in it, which would otherwise not be available.

[2]Virtualization is a technology that was initially developed during the mainframe era. The IBM CP/CMS mainframes were the first systems to introduce the concept of hardware virtualization and hypervisors. These systems, able to run multiple operating systems at the same time, provided a backward-compatible environment that allowed customers to run previous versions of their applications.

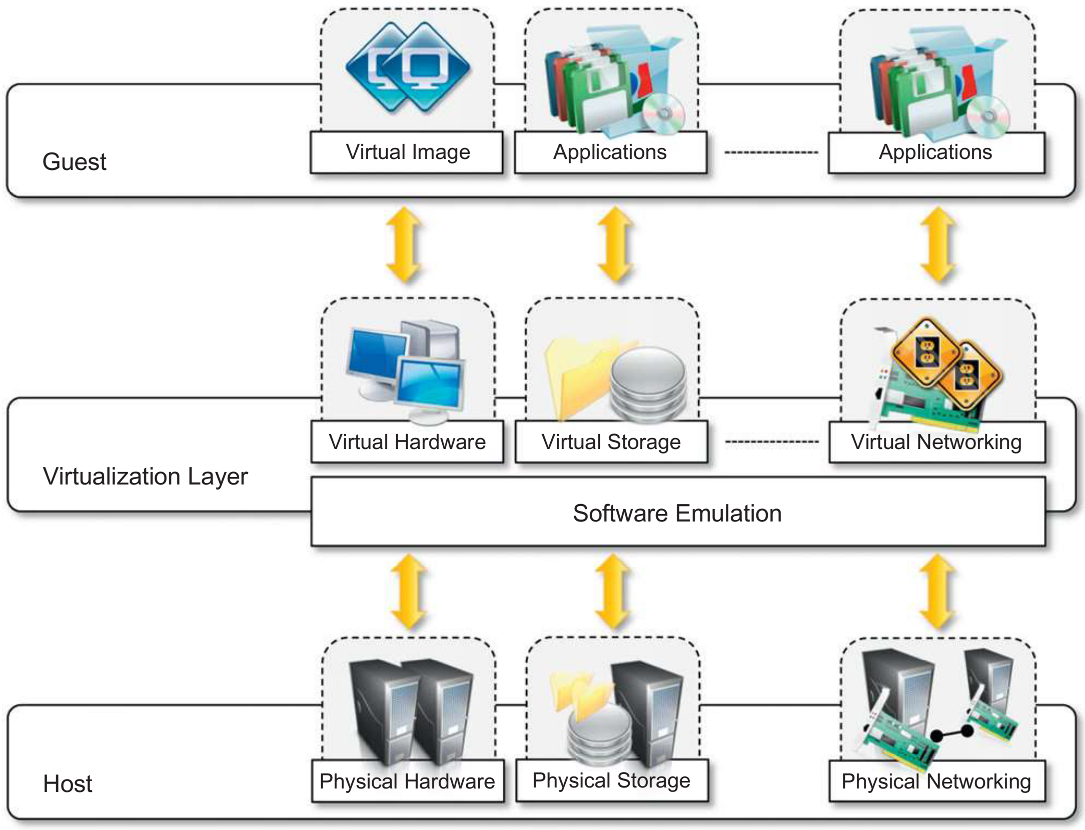

FIGURE 3.1

The virtualization reference model.

The main common characteristic of all these different implementations is the fact that the virtual environment is created by means of a *software program*. The ability to use software to emulate such a wide variety of environments creates a lot of opportunities, previously less attractive because of excessive overhead introduced by the virtualization layer. The technologies of today allow profitable use of virtualization and make it possible to fully exploit the advantages that come with it. Such advantages have always been characteristics of virtualized solutions.

3.2.1 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 and applies them to the host. This level of indirection allows the virtual machine manager to *control* and *filter* the activity of the guest, thus preventing some harmful operations from being performed. Resources exposed by the host can then be hidden or simply protected from the guest. Moreover, sensitive

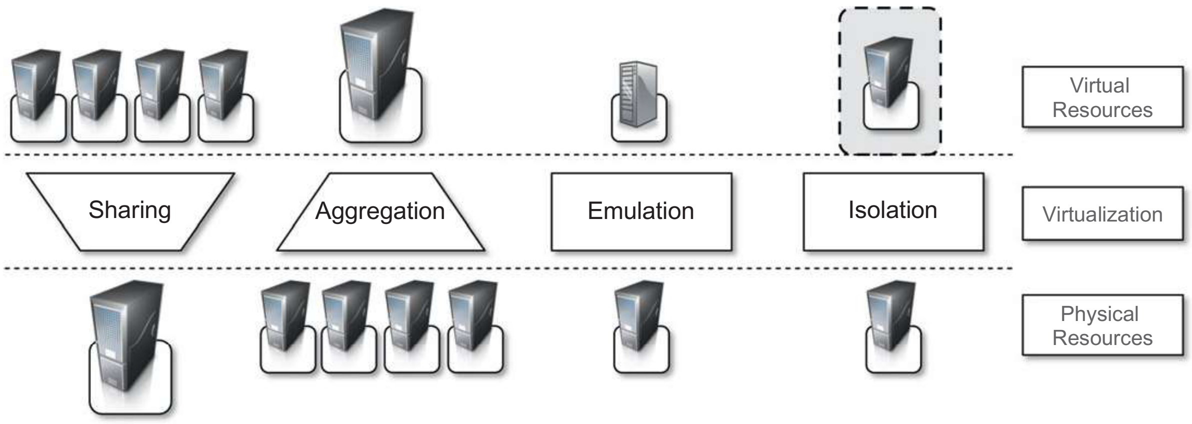

FIGURE 3.2

Functions enabled by managed execution.

information that is contained in the host can be naturally hidden without the need to install complex security policies. Increased security is a requirement when dealing with untrusted code. For example, applets downloaded from the Internet run in a sandboxed[3] version of the *Java Virtual Machine (JVM)*, which provides them with limited access to the hosting operating system resources. Both the JVM and the .NET runtime provide extensive security policies for customizing the execution environment of applications. Hardware virtualization solutions such as VMware Desktop, VirtualBox, and Parallels provide the ability to create a virtual computer with customized virtual hardware on top of which a new operating system can be installed. By default, the file system exposed by the virtual computer is completely separated from the one of the host machine. This becomes the perfect environment for running applications without affecting other users in the environment.

3.2.2 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 (see Figure 3.2).

- *Sharing.* Virtualization allows the creation of a separate computing environments within the same host. In this way it is possible to fully exploit the capabilities of a powerful guest, which would otherwise be underutilized. As we will see in later chapters, sharing is a particularly important feature in virtualized data centers, where this basic feature is used to reduce the number of active servers and limit power consumption.

[3]The term sandbox identifies an isolated execution environment where instructions can be filtered and blocked before being translated and executed in the real execution environment. The expression sandboxed version of the Java Virtual Machine (JVM) refers to a particular configuration of the JVM where, by means of security policy, instructions that are considered potential harmful can be blocked.

- *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. This feature becomes very useful for testing purposes, where a specific guest has to be validated against different platforms or architectures and the wide range of options is not easily accessible during development. Again, 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. Old and legacy software that does not meet the requirements of current systems can be run on emulated hardware without any need to change the code. This is possible either by emulating the required hardware architecture or within a specific operating system sandbox, such as the MS-DOS mode in Windows 95/98. Another example of emulation is an arcade-game emulator that allows us to play arcade games on a normal personal computer.
- *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. Isolation brings several benefits; for example, it allows multiple guests to run on the same host without interfering with each other. Second, it provides a separation between the host and the guest. The virtual machine can filter the activity of the guest and prevent harmful operations against the host.

Besides these characteristics, another important capability enabled by virtualization is *performance tuning*. This feature is a reality at present, given the considerable advances in hardware and software supporting virtualization. It becomes easier to control the performance of the guest by finely tuning the properties of the resources exposed through the virtual environment. This capability provides a means to effectively implement a quality-of-service (QoS) infrastructure that more easily fulfills the service-level agreement (SLA) established for the guest. For instance, software-implementing hardware virtualization solutions can expose to a guest operating system only a fraction of the memory of the host machine or set the maximum frequency of the processor of the virtual machine. Another advantage of managed execution is that sometimes it allows easy capturing of the state of the guest program, persisting it, and resuming its execution. This, for example, allows virtual machine managers such as Xen Hypervisor to stop the execution of a guest operating system, move its virtual image into another machine, and resume its execution in a completely transparent manner. This technique is called *virtual machine migration* and constitutes an important feature in virtualized data centers for optimizing their efficiency in serving application demands.

3.2.3 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. In the case of programming-level virtualization, as implemented by the JVM or the .NET runtime, the binary code representing application components (jars or assemblies) can be run without any recompilation on any implementation of the corresponding virtual machine. This makes the application development cycle more flexible and application deployment very straightforward: One version of the application, in most cases, is able to run on different platforms with no changes. Finally, portability allows having your own system always with you and ready to use as long as the required virtual machine manager is available. This requirement is, in general, less stringent than having all the applications and services you need available to you anywhere you go.

3.3 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 3.3).

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. Therefore, it deserves major investigation and a further categorization. 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.

3.3.1 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.

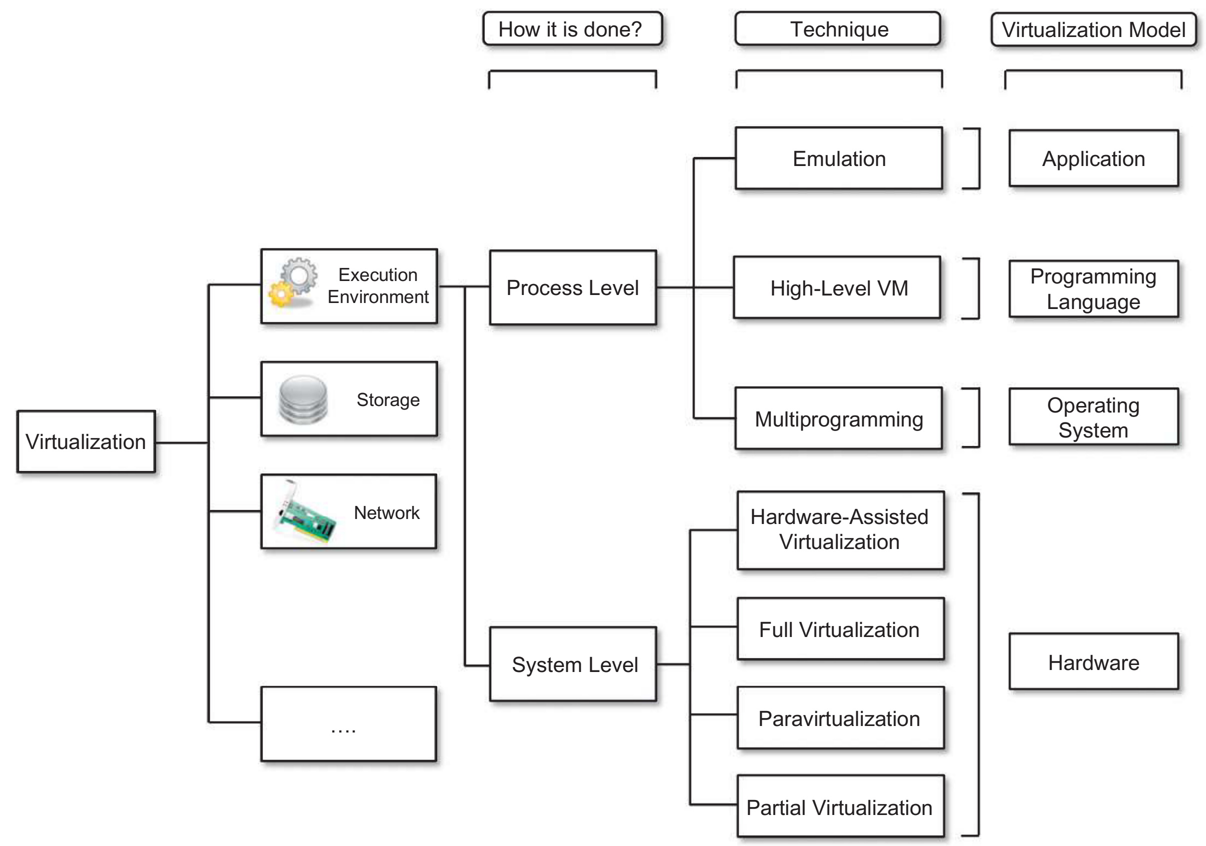

FIGURE 3.3

A taxonomy of virtualization techniques.

3.3.1.1 Machine reference model

Virtualizing 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. Therefore, a clear separation between layers simplifies their implementation, which only requires the emulation of the interfaces and a proper interaction with the underlying layer.

Modern computing systems can be expressed in terms of the reference model described in Figure 3.4. 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 interrupt 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*). 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

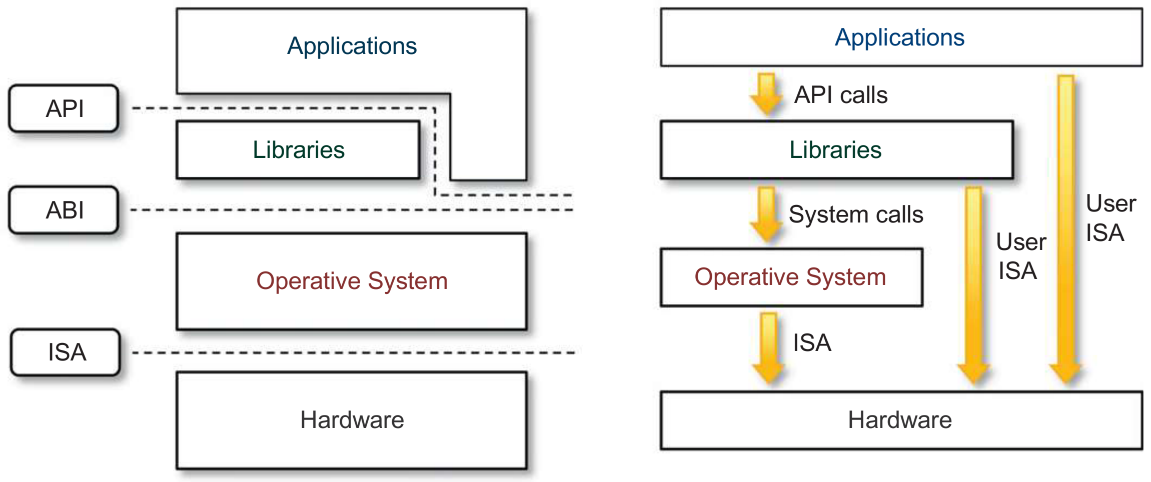

FIGURE 3.4

A machine reference model.

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.

For any operation to be performed in the application level API, ABI and ISA are responsible for making it happen. The high-level abstraction is converted into machine-level instructions to perform the actual operations supported by the processor. The machine-level resources, such as processor registers and main memory capacities, are used to perform the operation at the hardware level of the central processing unit (CPU). This layered approach simplifies the development and implementation of computing systems and simplifies the implementation of multitasking and the coexistence of multiple executing environments. In fact, such a model not only requires limited knowledge of the entire computing stack, but it also provides ways to implement a minimal security model for managing and accessing shared resources.

For this purpose, 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. Some types of architecture feature more than one class of privileged instructions and implement a finer control of how these instructions can be accessed. For instance, a possible implementation features a hierarchy of privileges (see Figure 3.5) 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.

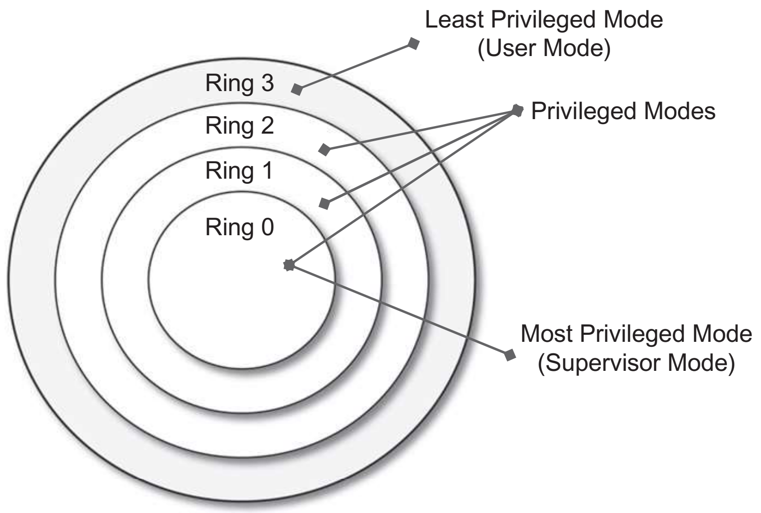

FIGURE 3.5

Security rings and privilege modes.

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. Despite this, there might be some instructions that can be invoked as privileged instructions under some conditions and as nonprivileged instructions under other conditions.

The distinction between *user* and *supervisor* mode allows us to understand the role of the *hypervisor* and why it is called that. Conceptually, the hypervisor runs above the supervisor mode, and from here the prefix *hyper-* is used. In reality, hypervisors are run in supervisor mode, and the division between privileged and nonprivileged instructions has posed challenges in designing virtual machine managers. It is expected that all the sensitive instructions will be executed in privileged mode, which requires supervisor mode in order to avoid traps. Without this assumption it is impossible to fully emulate and manage the status of the CPU for guest operating systems. Unfortunately, this is not true for the original ISA, which allows 17 sensitive instructions to be called in user mode. This prevents multiple operating systems managed by a single hypervisor to be isolated from each other, since they are able to access the privileged state of the processor and change it.[4] More recent implementations of ISA (Intel VT and AMD Pacifica) have solved this problem by redesigning such instructions as privileged ones.

By keeping in mind this reference model, it is possible to explore and better understand the various techniques utilized to virtualize execution environments and their relationships to the other components of the system.

[4]It is expected that in a hypervisor-managed environment, all the guest operating system code will be run in user mode in order to prevent it from directly accessing the status of the CPU. If there are sensitive instructions that can be called in user mode (that is, implemented as nonprivileged instructions), it is no longer possible to completely isolate the guest OS.

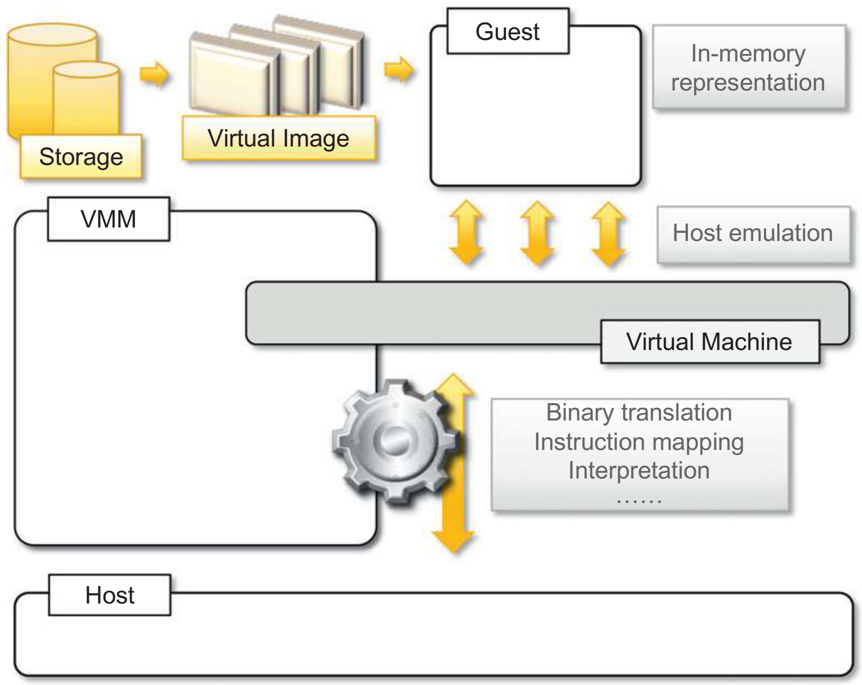

FIGURE 3.6

A hardware virtualization reference model.

3.3.1.2 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 3.6). 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. This is to differentiate it from *process virtual machines*, which expose ABI to virtual machines.

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 3.7).

- *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.

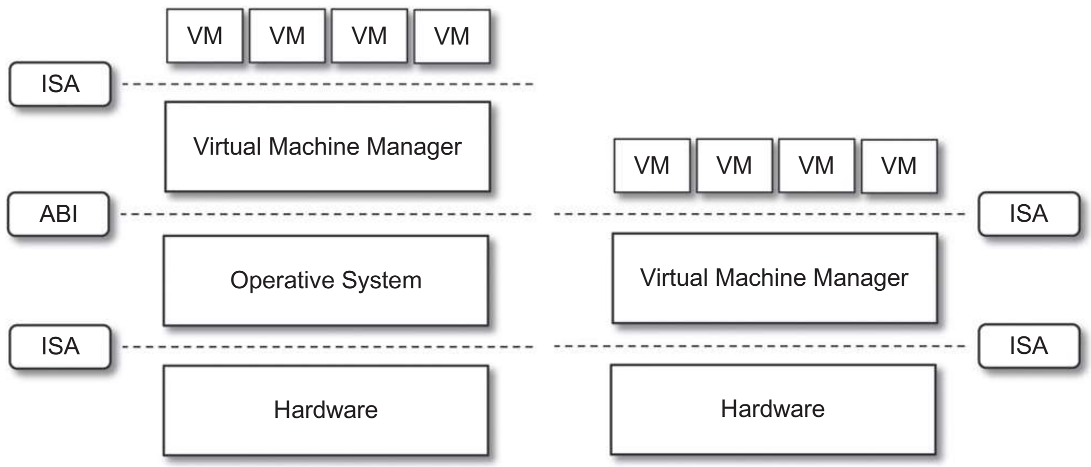

FIGURE 3.7

Hosted (left) and native (right) virtual machines. This figure provides a graphical representation of the two types of hypervisors.

Conceptually, a virtual machine manager is internally organized as described in Figure 3.8. 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.

The design and architecture of a virtual machine manager, together with the underlying hardware design of the host machine, determine the full realization of hardware virtualization, where a guest operating system can be transparently executed on top of a VMM as though it were run on the underlying hardware. The criteria that need to be met by a virtual machine manager to efficiently support virtualization were established by Goldberg and Popek in 1974 [23]. Three properties 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.

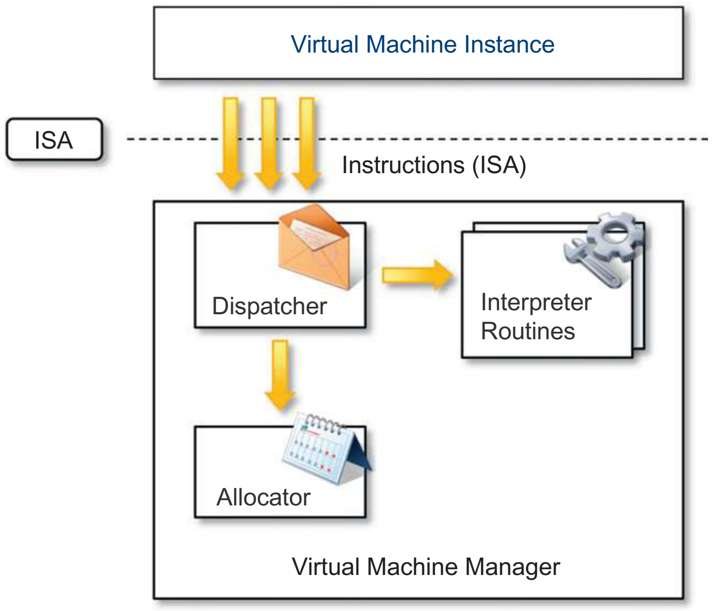

FIGURE 3.8

A hypervisor reference architecture.

- *Efficiency*. A statistically dominant fraction of the machine instructions should be executed without intervention from the virtual machine manager.

The major factor that determines whether these properties are satisfied is represented by the layout of the ISA of the host running a 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 3.1

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. The theorem always guarantees the resource control property when the hypervisor is in the most privileged mode (*Ring 0*). The nonprivileged instructions must be executed without the intervention of the hypervisor. The equivalence property also holds good since the output of the code is the same in both cases because the code is not changed.

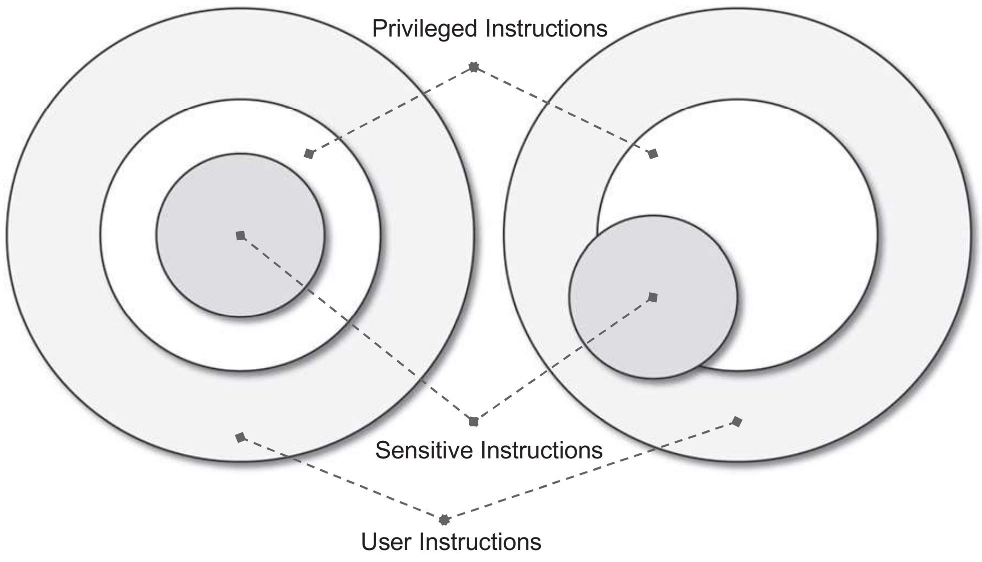

FIGURE 3.9

A virtualizable computer (left) and a nonvirtualizable computer (right).

THEOREM 3.2

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 3.3

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. Here all sensitive instructions are caught by HVM that are simulated.

This reference model represents what we generally consider classic virtualization—that is, the ability to execute a guest operating system in complete isolation. To a greater extent, hardware-level virtualization includes several strategies that differentiate from each other in terms of which kind of support is expected from the underlying hardware, what is actually abstracted from the host, and whether the guest should be modified or not.

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*). These extensions, which differ between the two vendors, are meant to reduce the performance penalties experienced by emulating x86 hardware with hypervisors. Before the introduction of hardware-assisted virtualization, software emulation of x86 hardware was significantly costly from the performance point of view. The reason for this is that by design the x86 architecture did not meet the formal requirements introduced by Popek and Goldberg, and early products were using binary translation to trap some sensitive instructions and provide an emulated version. 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. Whereas it is a desired goal for many virtualization solutions, full virtualization poses important concerns related to performance and technical implementation. A key challenge is the interception of privileged instructions such as I/O instructions: Since they change the state of the resources exposed by the host, they have to be contained within the virtual machine manager. A simple solution to achieve full virtualization is to provide a virtual environment for all the instructions, thus posing some limits on performance. A successful and efficient implementation of full virtualization is obtained with a combination of hardware and software, not allowing potentially harmful instructions to be executed directly on the host. This is what is accomplished through hardware-assisted virtualization.

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 allows a simpler implementation of virtual machine managers that have to simply transfer the execution of these operations, which were hard to virtualize, directly to the host. To take advantage of such an opportunity, guest operating systems need to be modified and explicitly ported by remapping the performance-critical operations through the virtual machine software interface. This is possible when the source code of the operating system is available, and this is the reason that paravirtualization was mostly explored in the open-source and academic environment. Whereas this technique was initially applied in the IBM VM operating system families, the term *paravirtualization* was introduced in literature in the Denali project [24] at the University of Washington. This technique has been successfully used by Xen for providing virtualization solutions for Linux-based operating systems specifically ported to run on Xen hypervisors. Operating systems that cannot be ported can still take advantage of paravirtualization by using ad hoc device drivers that remap the execution of critical instructions to the paravirtualization APIs exposed by the hypervisor. Xen provides this solution for running Windows-based operating systems on x86 architectures. Other solutions using paravirtualization include VMWare, Parallels, and some solutions for embedded and real-time environments such as TRANGO, Wind River, and XtratuM.

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. An example of partial virtualization is address space virtualization used in time-sharing systems; this allows multiple applications and users to run concurrently in a separate memory space, but they still share the same hardware resources (disk, processor, and network). Historically, partial virtualization has been an important milestone for achieving full virtualization, and it 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 systems supporting this type of virtualization are general-purpose, time-shared operating systems with the capability to provide stronger namespace and resource isolation.

This virtualization technique can be considered an evolution of the *chroot* mechanism in Unix systems. The *chroot* operation changes the file system root directory for a process and its children to a specific directory. As a result, the process and its children cannot have access to other portions of the file system than those accessible under the new root directory. Because Unix systems also expose devices as parts of the file system, by using this method it is possible to completely isolate a set of processes. Following the same principle, 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. There is no need to modify applications to run them, nor to modify any specific hardware, as in the case of hardware-assisted virtualization. On the other hand, operating system-level virtualization does not expose the same flexibility of hardware virtualization, since all the user space instances must share the same operating system.

This technique is an efficient solution for server consolidation scenarios in which multiple application servers share the same technology: operating system, application server framework, and other components. When different servers are aggregated into one physical server, each server is run in a different user space, completely isolated from the others.

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), and others. The services offered by these technologies differ, and most of them are available on Unix-based systems. Some of them, such as Solaris and OpenVZ, allow for different versions of the same operating system to operate concurrently.

3.3.1.3 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. Generally these virtual machines constitute a simplification of the underlying hardware instruction set and provide some high-level instructions that map some of the features of the languages compiled for them. At runtime, the byte code can be either interpreted or compiled on the fly—or *jitted*[5]—against the underlying hardware instruction set.

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. Originally created as a platform for developing Internet applications, Java became one of the technologies of choice for enterprise applications, and a large community of developers formed around it. The Java virtual machine was originally designed for the execution of programs written in the Java language, but other languages such as Python, Pascal, Groovy, and Ruby were made available. The ability to support multiple programming languages has been one of the key elements of the *Common Language Infrastructure (CLI)*, which is the specification behind

[5]The term *jitted* is an improper use of the *just-in-time (JIT)* acronym as a verb, which has now become common. It refers to a specific execution strategy in which the byte code of a method is compiled against the underlying machine code upon method call—that is, *just in time*. Initial implementations of programming-level virtualization were based on interpretation, which led to considerable slowdowns during execution. The advantage of just-in-time compilation is that the machine code that has been compiled can be reused for executing future calls to the same methods. Virtual machines that implement JIT compilation generally have a method cache that stores the code generated for each method and simply look up this cache before triggering the compilation upon each method call.

.NET Framework. Currently, the Java platform and .NET Framework represent the most popular technologies for enterprise application development.

Both Java and the CLI are *stack-based* virtual machines: The reference model of the abstract architecture is based on an execution stack that is used to perform operations. The byte code generated by compilers for these architectures contains a set of instructions that load operands on the stack, perform some operations with them, and put the result on the stack. Additionally, specific instructions for invoking methods and managing objects and classes are included. Stack-based virtual machines possess the property of being easily interpreted and executed simply by lexical analysis and hence are easily portable over different architectures. An alternative solution is offered by *register-based* virtual machines, in which the reference model is based on registers. This kind of virtual machine is closer to the underlying architecture we use today. An example of a register-based virtual machine is Parrot, a programming-level virtual machine that was originally designed to support the execution of PERL and then generalized to host the execution of dynamic languages.

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. From a development lifecycle point of view, this simplifies the development and deployment efforts since it is not necessary to provide different versions of the same code. The implementation of the virtual machine for different platforms is still a costly task, but it is done once and not for any application. Moreover, process virtual machines allow for more control over the execution of programs since they do not provide direct access to the memory. Security is another advantage of managed programming languages; by filtering the I/O operations, the process virtual machine can easily support sandboxing of applications. As an example, both Java and .NET provide an infrastructure for pluggable security policies and code access security frameworks. All these advantages come with a price: performance. Virtual machine programming languages generally expose an inferior performance compared to languages compiled against the real architecture. This performance difference is getting smaller, and the high compute power available on average processors makes it even less important.

Implementations of this model are also called *high-level virtual machines*, since high-level programming languages are compiled to a conceptual ISA, which is further interpreted or dynamically translated against the specific instruction of the hosting platform.

3.3.1.4 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.

Emulation, as described, is different from hardware-level virtualization. The former simply allows the execution of a program compiled against a different hardware, whereas the latter emulates a complete hardware environment where an entire operating system can be installed.

Application virtualization is a good solution in the case of missing libraries in the host operating system; in this case a replacement library can be linked with the application, or library calls can be remapped to existing functions available in the host system. Another advantage is that in this case the virtual machine manager is much lighter since it provides a partial emulation of the runtime environment compared to hardware virtualization. Moreover, this technique allows incompatible applications to run together. Compared to programming-level virtualization, which works across all the applications developed for that virtual machine, application-level virtualization works for a specific environment: It supports all the applications that run on top of a specific environment.

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. Wine features a software application acting as a container for the guest application and a set of libraries, called *Winelib*, that developers can use to compile applications to be ported on Unix systems. Wine takes its inspiration from a similar product from Sun, *Windows Application Binary Interface (WABI)*, which implements the Win 16 API specifications on Solaris. A similar solution for the Mac OS X environment is CrossOver, which allows running Windows applications directly on the Mac OS X operating system. VMware ThinApp, another product in this area, allows capturing the setup of an installed application and packaging it into an executable image isolated from the hosting operating system.

3.3.2 Other types of virtualization

Other than execution virtualization, other types of virtualization provide an abstract environment to interact with. These mainly cover storage, networking, and client/server interaction.

3.3.2.1 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.

Storage virtualization allows us to harness a wide range of storage facilities and represent them under a single logical file system. There are different techniques for storage virtualization, one of the most popular being network-based virtualization by means of *storage area networks (SANs)*. SANs use a network-accessible device through a large bandwidth connection to provide storage facilities.

3.3.2.2 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. Internal network virtualization is generally applied together with hardware and operating system-level virtualization, in which the guests obtain a virtual network interface to communicate with. There are several options for implementing internal network virtualization: The guest can share the same network 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.

3.3.2.3 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 outcome 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. Although the term *desktop virtualization* strictly refers to the ability to remotely access a desktop environment, generally the desktop environment is stored in a remote server or a data center that provides a high-availability infrastructure and ensures the accessibility and persistence of the data.

In this scenario, an infrastructure supporting hardware virtualization is fundamental to provide access to multiple desktop environments hosted on the same server; a specific desktop environment is stored in a virtual machine image that is loaded and started on demand when a client connects to the desktop environment. This is a typical cloud computing scenario in which the user leverages the virtual infrastructure for performing the daily tasks on his computer. The advantages of desktop virtualization are high availability, persistence, accessibility, and ease of management. As we will discuss in Section 4.5.4 of the next chapter, security issues can prevent the use of this technology. The basic services for remotely accessing a desktop environment are implemented in software components such as Windows Remote Services, VNC, and X Server. Infrastructures for desktop virtualization based on cloud computing solutions include Sun Virtual Desktop Infrastructure (VDI), Parallels Virtual Desktop Infrastructure (VDI), Citrix XenDesktop, and others.

3.3.2.4 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.

3.4 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. Network virtualization is less popular and, in most cases, is a complementary feature, which is naturally needed in build virtual computing systems.

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.

Besides being an enabler for computation on demand, virtualization also gives the opportunity to design more efficient computing systems by means of consolidation, which is performed transparently to cloud computing service users. Since 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. This opportunity is particularly attractive when resources are underutilized, because 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.[6]

[6]It is important to notice that cloud computing is strongly leveraged for the development of applications that need to scale on demand. In most cases, this is because applications have to process increased workloads or serve more requests, which makes them server applications. In this scenario, it is evident that live migration offers a better solution because it does not create any service interruption during consolidation.

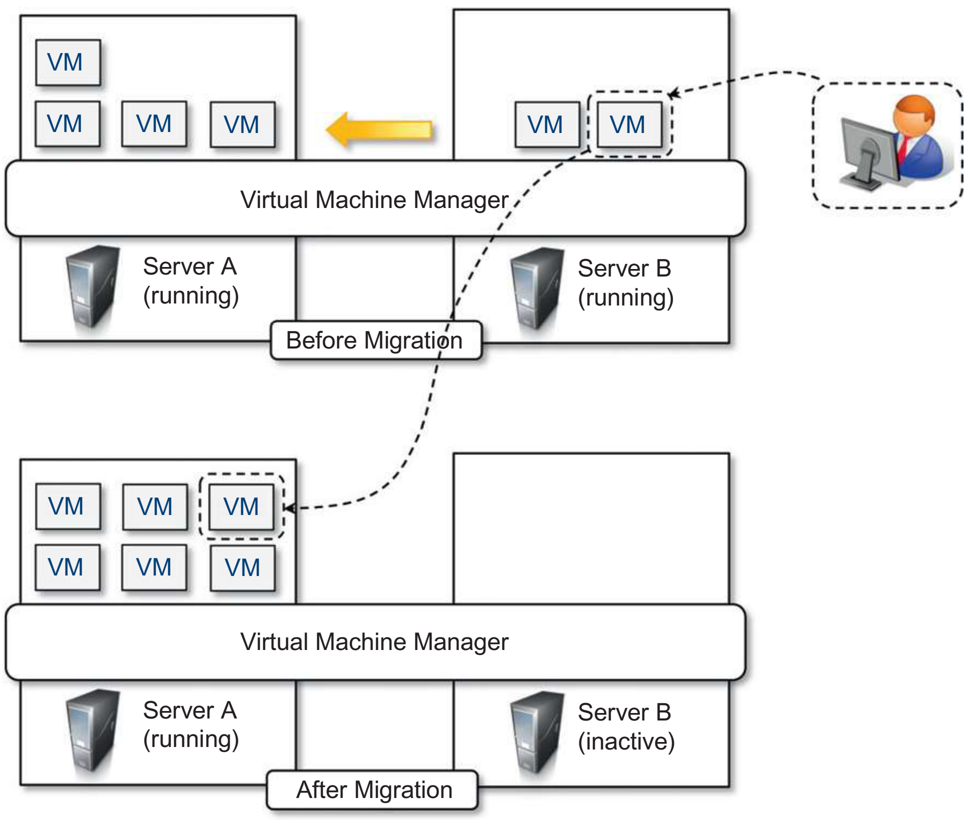

FIGURE 3.10

Live migration and server consolidation.

Server consolidation and virtual machine migration are principally used in the case of hardware virtualization, even though they are also technically possible in the case of programming language virtualization (see Figure 3.9).

Storage virtualization constitutes an interesting opportunity given by virtualization technologies, often complementary to the execution of virtualization. Even in this case, vendors backed by large computing infrastructures featuring huge storage facilities can harness these facilities into a virtual storage service, easily partitionable into slices. These slices can be dynamic and offered as a service. Again, opportunities to secure and protect the hosting infrastructure are available, as are methods for easy accountability of such services.

Finally, cloud computing revamps the concept of desktop virtualization, initially introduced in the mainframe era. The ability to recreate the entire computing stack—from infrastructure to application services—on demand opens the path to having a complete virtual computer hosted on the infrastructure of the provider and accessed by a thin client over a capable Internet connection.

3.5 Pros and cons of virtualization

Virtualization has now become extremely popular and widely used, especially in cloud computing. The primary reason for its wide success is the elimination of technology barriers that prevented virtualization from being an effective and viable solution in the past. The most relevant barrier has been performance. Today, the capillary diffusion of the Internet connection and the advancements in computing technology have made virtualization an interesting opportunity to deliver on-demand IT infrastructure and services. Despite its renewed popularity, this technology has benefits and also drawbacks.

3.5.1 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 is another advantage of virtualization, especially for execution virtualization techniques. Virtual machine instances are normally represented by one or more files that can be easily transported with respect to physical systems. Moreover, they also tend to be self-contained since they do not have other dependencies besides the virtual machine manager for their use. Portability and self-containment simplify their administration. Java programs are "compiled once and run everywhere"; they only require that the Java virtual machine be installed on the host. The same applies to hardware-level virtualization. It is in fact possible to build our own operating environment within a virtual machine instance and bring it with us wherever we go, as though we had our own laptop. This concept is also an enabler for migration techniques in a server consolidation scenario.

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. Since the guest program is executed in a virtual environment, there is very limited opportunity for the guest program to damage the underlying hardware. Moreover, it is expected that there will be fewer virtual machine managers with respect to the number of virtual machine instances managed.

Finally, 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. This is a prerequisite for server consolidation, which allows adjusting the number of active physical resources dynamically according to the current load of the system, thus creating the opportunity to save in terms of energy consumption and to be less impacting on the environment.

3.5.2 The other side of the coin: disadvantages

Virtualization also has downsides. The most evident is represented by a performance decrease of guest systems as a result of the intermediation performed by the virtualization layer. In addition, suboptimal use of the host because of the abstraction layer introduced by virtualization management software can lead to a very inefficient utilization of the host or a degraded user experience. Less evident, but perhaps more dangerous, are the implications for security, which are mostly due to the ability to emulate a different execution environment.

3.5.2.1 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 consideration 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 physical 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 hardware 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 performance is a serious concern.

3.5.2.2 Inefficiency and degraded user experience

Virtualization can sometime 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. For example, in the first version of Java the support for graphic programming was very limited and the look and feel of applications was very poor compared to native applications. These issues have been resolved by providing a new framework called *Swing* for designing the user interface, and further improvements have been done by integrating support for the OpenGL libraries in the software development kit.

3.5.2.3 Security holes and new threats

Virtualization opens the door to a new and unexpected form of *phishing*.[7] The capability of emulating 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. Examples of these kinds of malware are BluePill and SubVirt. BluePill, malware targeting the AMD processor family, moves the execution of the installed OS within a virtual machine. The original version of SubVirt was developed as a prototype by Microsoft through collaboration with Michigan University. SubVirt infects the guest OS, and when the virtual machine is rebooted, it gains control of the host. The diffusion of such kinds of malware is facilitated by the fact that originally, hardware and CPUs were not manufactured with virtualization in mind. In particular, the existing instruction sets cannot be simply changed or updated to suit the needs of virtualization. Recently, both Intel and AMD have introduced hardware support for virtualization with Intel VT and AMD Pacifica, respectively.

The same considerations can be made for programming-level virtual machines: Modified versions of the runtime environment can access sensitive information or monitor the memory locations utilized by guest applications while these are executed. To make this possible, the original version of the runtime environment needs to be replaced by the modified one, which can generally happen if the malware is run within an administrative context or a security hole of the host operating system is exploited.

3.6 Technology examples

A wide range of virtualization technology is available especially for virtualizing computing environments. In this section, we discuss the most relevant technologies and approaches utilized in the field. Cloud-specific solutions are discussed in the next chapter.

[7]*Phishing* is a term that identifies a malicious practice aimed at capturing sensitive user information, such as usernames and passwords, by recreating an environment identical in functionalities and appearance to the one that manages this information. Phishing most commonly occurs on the Web, where the user is redirected to a malicious website that is a replica of the original and the purpose of which is to collect the information to impersonate the user on the original Website (e.g., a bank site) and access the user's confidential data.

3.6.1 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. Citrix also offers it as a commercial solution, XenSource. 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). At the basis of all these solutions is the Xen Hypervisor, which constitutes the core technology of Xen. Recently Xen has been advanced to support full virtualization using hardware-assisted virtualization.

Xen is the most popular implementation of *paravirtualization*, which, in contrast with full virtualization, allows high-performance execution of guest operating systems. This is made possible by eliminating the performance loss while executing instructions that require special management. This is done by modifying portions of the guest operating systems run by Xen with reference to the execution of such instructions. Therefore it is not a transparent solution for implementing virtualization. This is particularly true for x86, which is the most popular architecture on commodity machines and servers.

Figure 3.11 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. Guest

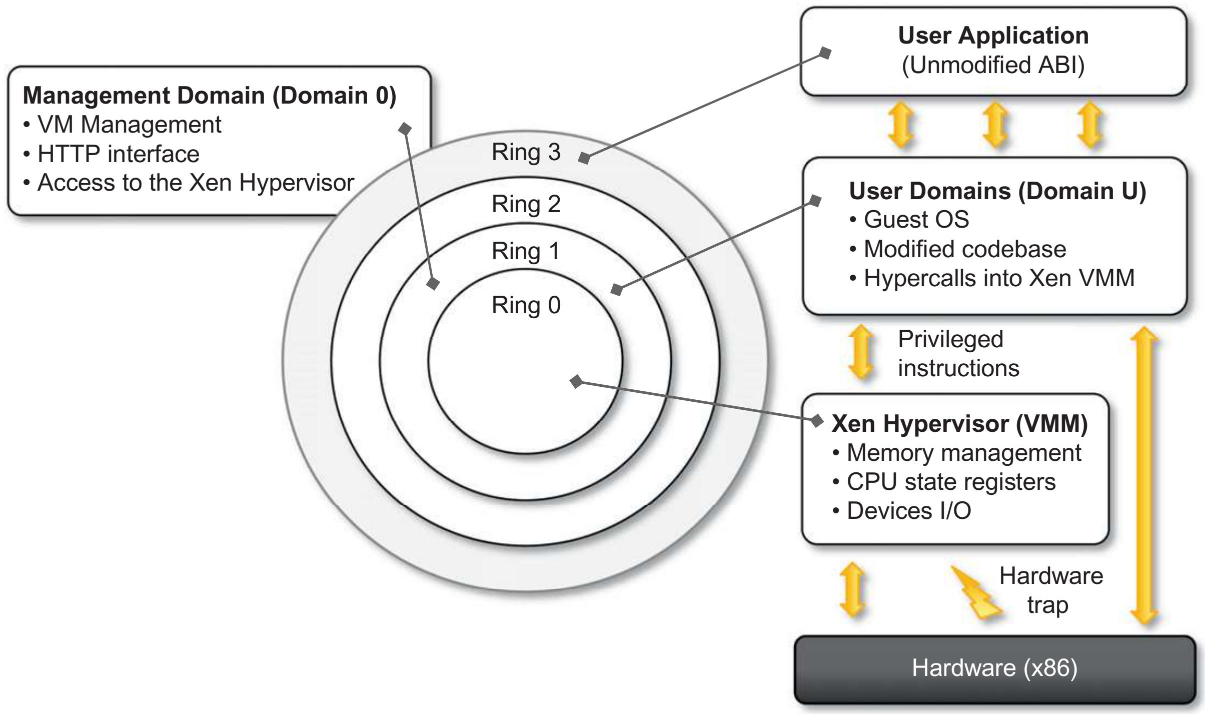

FIGURE 3.11

Xen architecture and guest OS management.

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, configuration, 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. Almost all the most popular operating systems, except OS/2, utilize only two levels: Ring 0 for the kernel code, and Ring 3 for user application and nonprivileged OS code. This provides the opportunity for Xen to implement virtualization by executing the hypervisor in Ring 0, Domain 0, and all the other domains running guest operating systems—generally referred to as *Domain U*—in Ring 1, while the user applications are run in Ring 3. This allows Xen to maintain the ABI unchanged, thus allowing an easy switch to Xen-virtualized solutions from an application point of view. 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. More precisely, this condition holds in a scenario where it is not possible to leverage hardware-assisted virtualization, which allows running the hypervisor in Ring -1 and the guest operating system in Ring 0. Therefore, Xen exhibits some limitations in the case of legacy hardware and legacy operating systems. In fact, these cannot be modified to be run in Ring 1 safely since their codebase is not accessible and, at the same time, the underlying hardware does not provide any support to run the hypervisor in a more privileged mode than Ring 0. 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. It can be observed that the problem is now becoming less and less crucial since both new releases of operating systems are designed to be virtualization aware and the new hardware supports x86 virtualization.

3.6.2 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 abstraction 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.

Besides these two core solutions, VMware provides additional tools and software that simplify the use of virtualization technology either in a desktop environment, with tools enhancing the integration of virtual guests with the host, or in a server environment, with solutions for building and managing virtual computing infrastructures.

3.6.2.1 Full virtualization and binary translation

VMware is well known for the capability to virtualize x86 architectures, which runs unmodified on top of their hypervisors. With the new generation of hardware architectures and the introduction of *hardware-assisted virtualization* (Intel VT-x and AMD V) in 2006, full virtualization is made possible with hardware support, but before that date, the use of *dynamic binary translation* was the only solution that allowed running x86 guest operating systems unmodified in a virtualized environment.

As discussed before, x86 architecture design does not satisfy the first theorem of virtualization, since the set of sensitive instructions is not a subset of the privileged instructions. This causes a different behavior when such instructions are not executed in Ring 0, which is the normal case in a virtualization scenario where the guest OS is run in Ring 1. Generally, a trap is generated and the way it is managed differentiates the solutions in which virtualization is implemented for x86 hardware. In the case of dynamic binary translation, the trap triggers the translation of the offending instructions into an equivalent set of instructions that achieves the same goal without generating exceptions. Moreover, to improve performance, the equivalent set of instruction is cached so that translation is no longer necessary for further occurrences of the same instructions. Figure 3.12 gives an idea of the process.

This approach has both advantages and disadvantages. The major advantage is that guests can run unmodified in a virtualized environment, which is a crucial feature for operating systems for which source code is not available. This is the case, for example, of operating systems in the Windows family. Binary translation is a more portable solution for full virtualization. On the other hand, translating instructions at runtime introduces an additional overhead that is not present in other approaches (paravirtualization or hardware-assisted virtualization). Even though such disadvantage exists, binary translation is applied to only a subset of the instruction set, whereas the others are managed through direct execution on the underlying hardware. This somehow reduces the impact on performance of binary translation.

CPU virtualization is only a component of a fully virtualized hardware environment. VMware achieves full virtualization by providing virtual representation of memory and I/O devices. Memory virtualization constitutes another challenge of virtualized environments and can deeply impact performance without the appropriate hardware support. The main reason is the presence of a *memory management unit (MMU)*, which needs to be emulated as part of the virtual hardware. Especially in the case of *hosted hypervisors* (Type II), where the virtual MMU and the host-OS MMU are traversed sequentially before getting to the physical memory page, the impact on performance can be significant. To avoid nested translation, the *translation look-aside buffer (TLB)* in the virtual MMU directly maps physical pages, and the performance slowdown only occurs in case of a TLB miss.

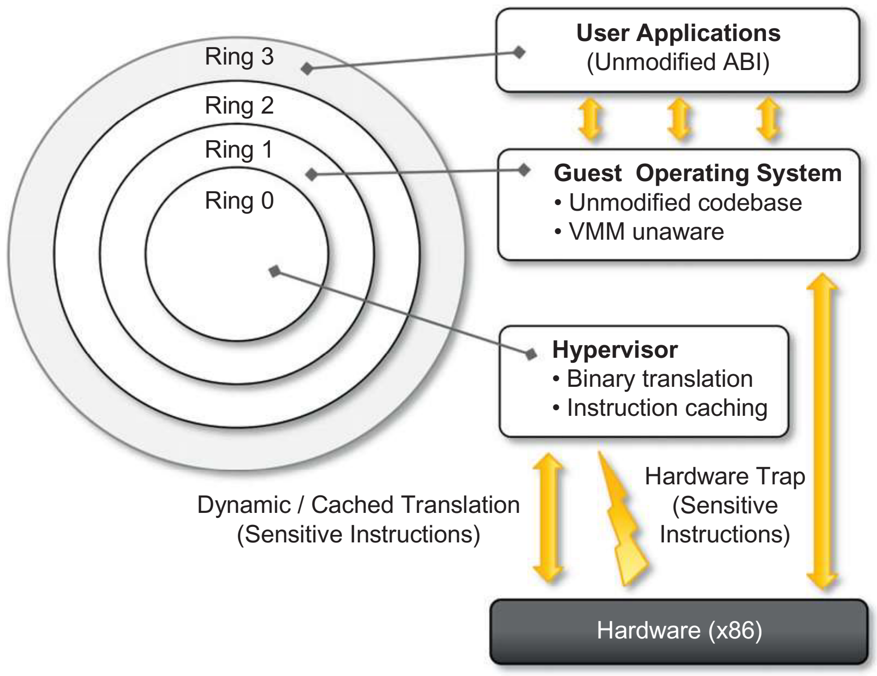

FIGURE 3.12

A full virtualization reference model.

Finally, VMware also provides full virtualization of I/O devices such as network controllers and other peripherals such as keyboard, mouse, disks, and universal serial bus (USB) controllers.

3.6.2.2 Virtualization solutions

VMware is a pioneer in virtualization technology and offers a collection of virtualization solutions covering the entire range of the market, from desktop computing to enterprise computing and infrastructure virtualization.

End-user (desktop) virtualization

VMware supports virtualization of operating system environments and single applications on end-user computers. The first option is the most popular and allows installing a different operating systems and applications in a completely isolated environment from the hosting operating system. Specific VMware software—*VMware Workstation*, for Windows operating systems, and *VMware Fusion*, for Mac OS X environments—is installed in the host operating system to create virtual machines and manage their execution. Besides the creation of an isolated computing environment, the two products allow a guest operating system to leverage the resources of the host machine (USB devices, folder sharing, and integration with the graphical user interface (GUI) of the host operating system). Figure 3.13 provides an overview of the architecture of these systems.

The virtualization environment is created by an application installed in guest operating systems, which provides those operating systems with full hardware virtualization of the underlying

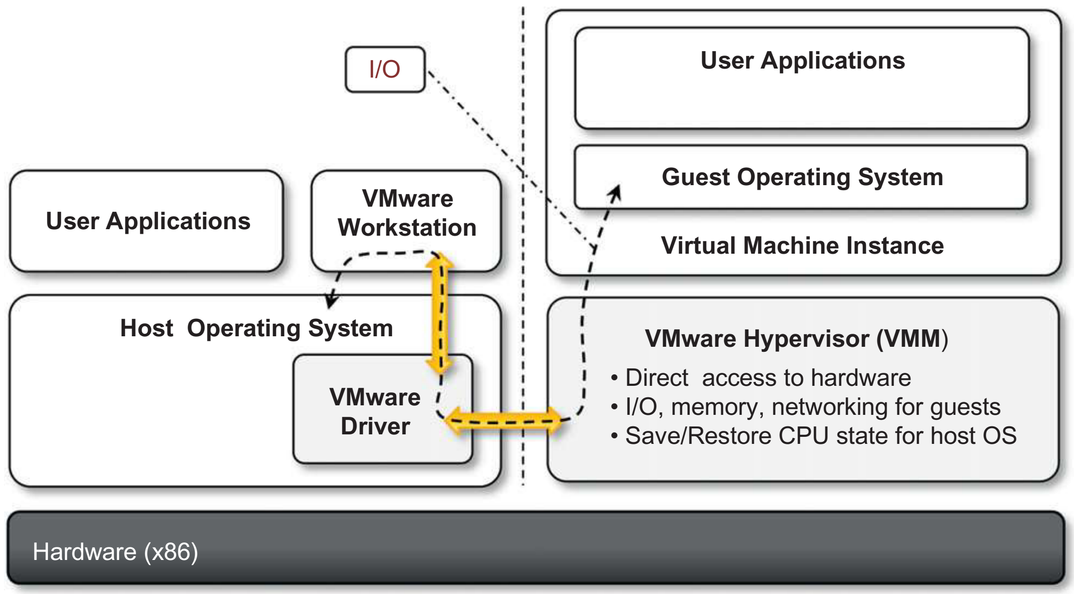

FIGURE 3.13

VMware workstation architecture.

hardware. This is done by installing a specific driver in the host operating system that provides two main services:

- It deploys a virtual machine manager that can run in privileged mode.
- It provides hooks for the VMware application to process specific I/O requests eventually by relaying such requests to the host operating system via system calls.

Using this architecture—also called *Hosted Virtual Machine Architecture*—it is possible to both isolate virtual machine instances within the memory space of a single application and provide reasonable performance, since the intervention of the VMware application is required only for instructions, such as device I/O, that require binary translation. Instructions that can be directly executed are managed by the virtual machine manager, which takes control of the CPU and the MMU and alternates its activity with the host OS. Virtual machine images are saved in a collection of files on the host file system, and both VMware Workstation and VMware Fusion allow creation of new images, pause their execution, create snapshots, and undo operations by rolling back to a previous state of the virtual machine.

Other solutions related to the virtualization of end-user computing environments include VMware Player, VMware ACE, and VMware ThinApp. VMware Player is a reduced version of VMware Workstation that allows creating and playing virtual machines in a Windows or Linux operating environment. VMware ACE, a similar product to VMware Workstation, creates policy-wrapped virtual machines for deploying secure corporate virtual environments on end-user computers. VMware ThinApp is a solution for application virtualization. It provides an isolated environment for applications in order to avoid conflicts due to versioning and incompatible applications. It detects all the changes to the operating environment made by the installation of a specific application and stores them together with the application binary into a package that can be run with VMware ThinApp.

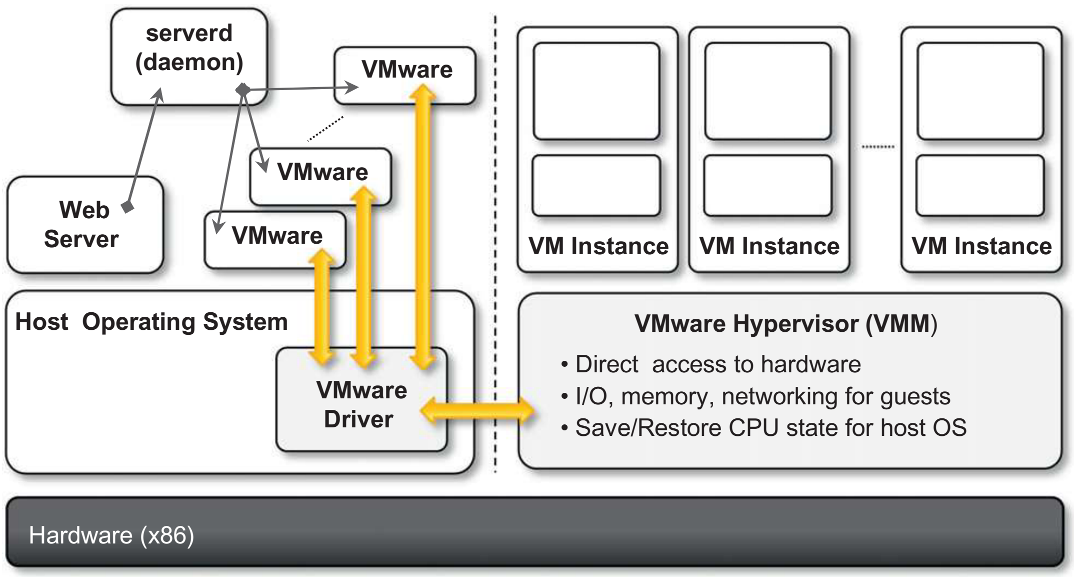

FIGURE 3.14

VMware GSX server architecture.

Server virtualization

VMware provided solutions for server virtualization with different approaches over time. Initial support for server virtualization was provided by VMware GSX server, which replicates the approach used for end-user computers and introduces remote management and scripting capabilities. The architecture of VMware GSX Server is depicted in Figure 3.14.

The architecture is mostly designed to serve the virtualization of Web servers. A daemon process, called *serverd*, controls and manages VMware application processes. These applications are then connected to the virtual machine instances by means of the VMware driver installed on the host operating system. Virtual machine instances are managed by the VMM as described previously. User requests for virtual machine management and provisioning are routed from the Web server through the VMM by means of *serverd*.

VMware ESX Server and its enhanced version, VMWare ESXi Server, are examples of the hypervisor-based approach. Both can be installed on bare metal servers and provide services for virtual machine management. The two solutions provide the same services but differ in the internal architecture, more specifically in the organization of the hypervisor kernel. VMware ESX embeds a modified version of a Linux operating system, which provides access through a service console to hypervisor. VMware ESXi implements a very thin OS layer and replaces the service console with interfaces and services for remote management, thus considerably reducing the hypervisor code size and memory footprint.

The architecture of VMware ESXi is displayed in Figure 3.15. The base of the infrastructure is the VMkernel, which is a thin Portable Operating System Interface (POSIX) compliant operating system that provides the minimal functionality for processes and thread management, file system, I/O stacks, and resource scheduling. The kernel is accessible through specific APIs called User world API. These

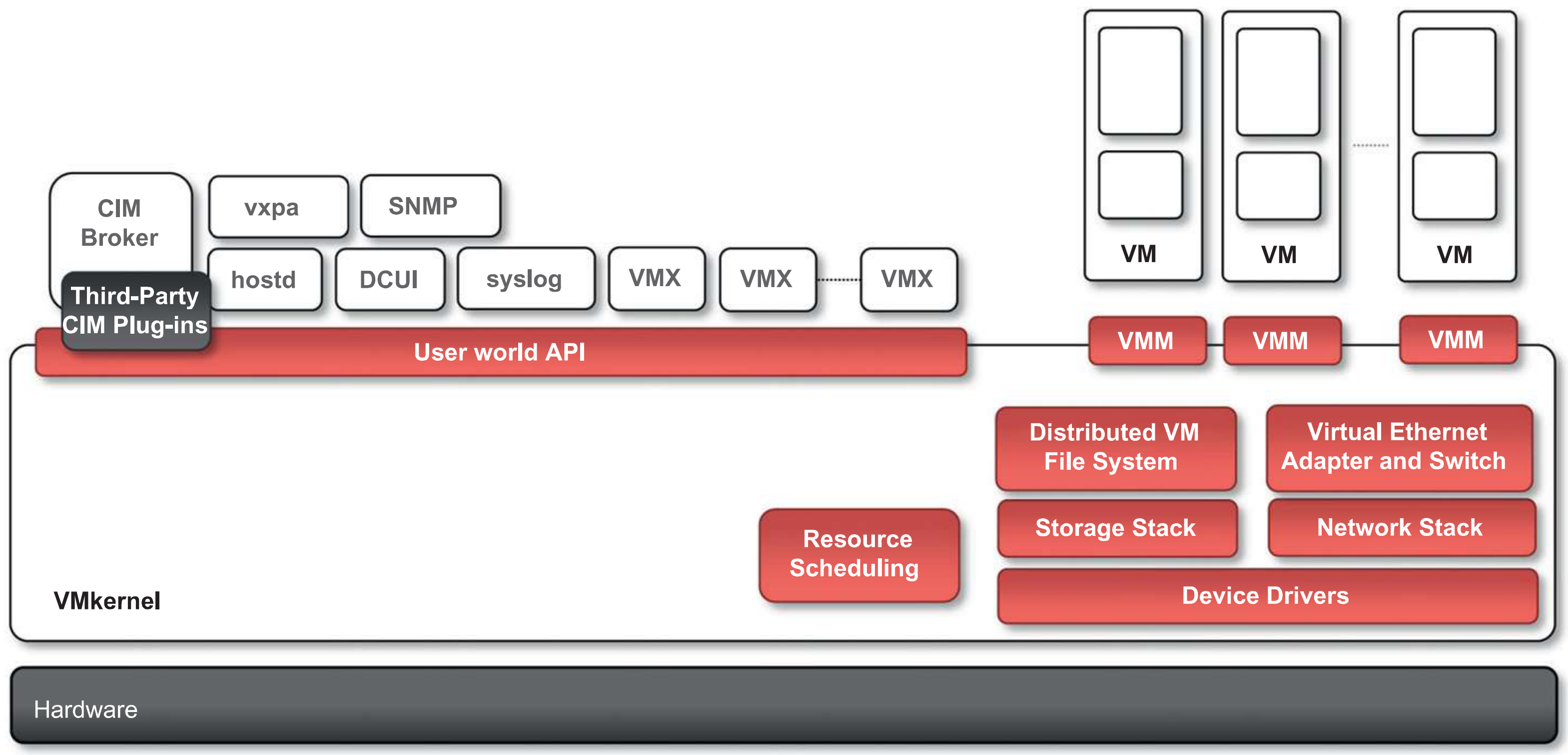

FIGURE 3.15

VMware ESXi server architecture.

APIs are utilized by all the agents that provide supporting activities for the management of virtual machines. Remote management of an ESXi server is provided by the CIM Broker, a system agent that acts as a gateway to the VMkernel for clients by using the *Common Information Model (CIM)*[8] protocol. The ESXi installation can also be managed locally by a *Direct Client User Interface (DCUI)*, which provides a BIOS-like interface for the management of local users.

Infrastructure virtualization and cloud computing solutions

VMware provides a set of products covering the entire stack of cloud computing, from infrastructure management to Software-as-a-Service solutions hosted in the cloud. Figure 3.16 gives an overview of the different solutions offered and how they relate to each other.

ESX and ESXi constitute the building blocks of the solution for virtual infrastructure management: A pool of virtualized servers is tied together and remotely managed as a whole by VMware vSphere. As a virtualization platform it provides a set of basic services besides virtual compute services: Virtual file system, virtual storage, and virtual network constitute the core of the infrastructure; application services, such as virtual machine migration, storage migration, data recovery, and

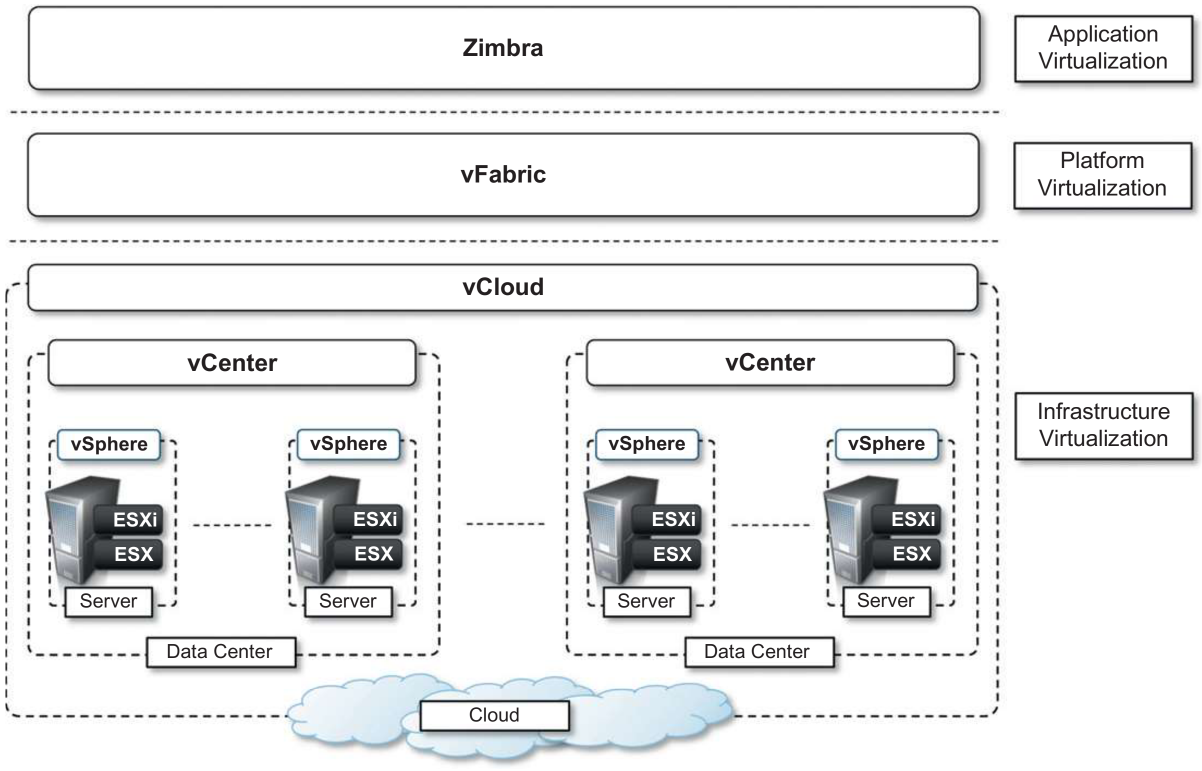

FIGURE 3.16

VMware Cloud Solution stack.

[8]Common Information Model (CIM) is a Distributed Management Task Force standard for defining management information for systems, applications, and services. See http://dmtf.org/standards/cim.

security zones, complete the services offered by vSphere. The management of the infrastructure is operated by VMware vCenter, which provides centralized administration and management of vSphere installations in a data center environment. A collection of virtualized data centers are turned into a Infrastructure-as-a-Service cloud by VMware vCloud, which allows service providers to make available to end users virtual computing environments on demand on a pay-per-use basis. A Web portal provides access to the provisioning services of vCloud, and end users can self-provision virtual machines by choosing from available templates and setting up virtual networks among virtual instances.

VMware also provides a solution for application development in the cloud with VMware vFabric, which is a set of components that facilitate the development of scalable Web applications on top of a virtualized infrastructure. vFabric is a collection of components for application monitoring, scalable data management, and scalable execution and provisioning of Java Web applications.

Finally, at the top of the cloud computing stack, VMware provides Zimbra, a solution for office automation, messaging, and collaboration that is completely hosted in the cloud and accessible from anywhere. This is an SaaS solution that integrates various features into a single software platform providing email and collaboration management.

3.6.2.3 Observations

Initially starting with a solution for fully virtualized x86 hardware, VMware has grown over time and now provides a complete offering for virtualizing hardware, infrastructure, applications, and services, thus covering every segment of the cloud computing market. Even though full x86 virtualization is the core technology of VMware, over time paravirtualization features have been integrated into some of the solutions offered by the vendor, especially after the introduction of hardware-assisted virtualization. For instance, the implementation of some device emulations and the VMware Tools suite that allows enhanced integration with the guest and the host operating environment. Also, VMware has strongly contributed to the development and standardization of a vendor-independent *Virtual Machine Interface (VMI)*, which allows for a general and host-agnostic approach to paravirtualization.

3.6.3 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. Hyper-V is currently shipped as a component of Windows Server 2008 R2 that installs the hypervisor as a role within the server.

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 3.17 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

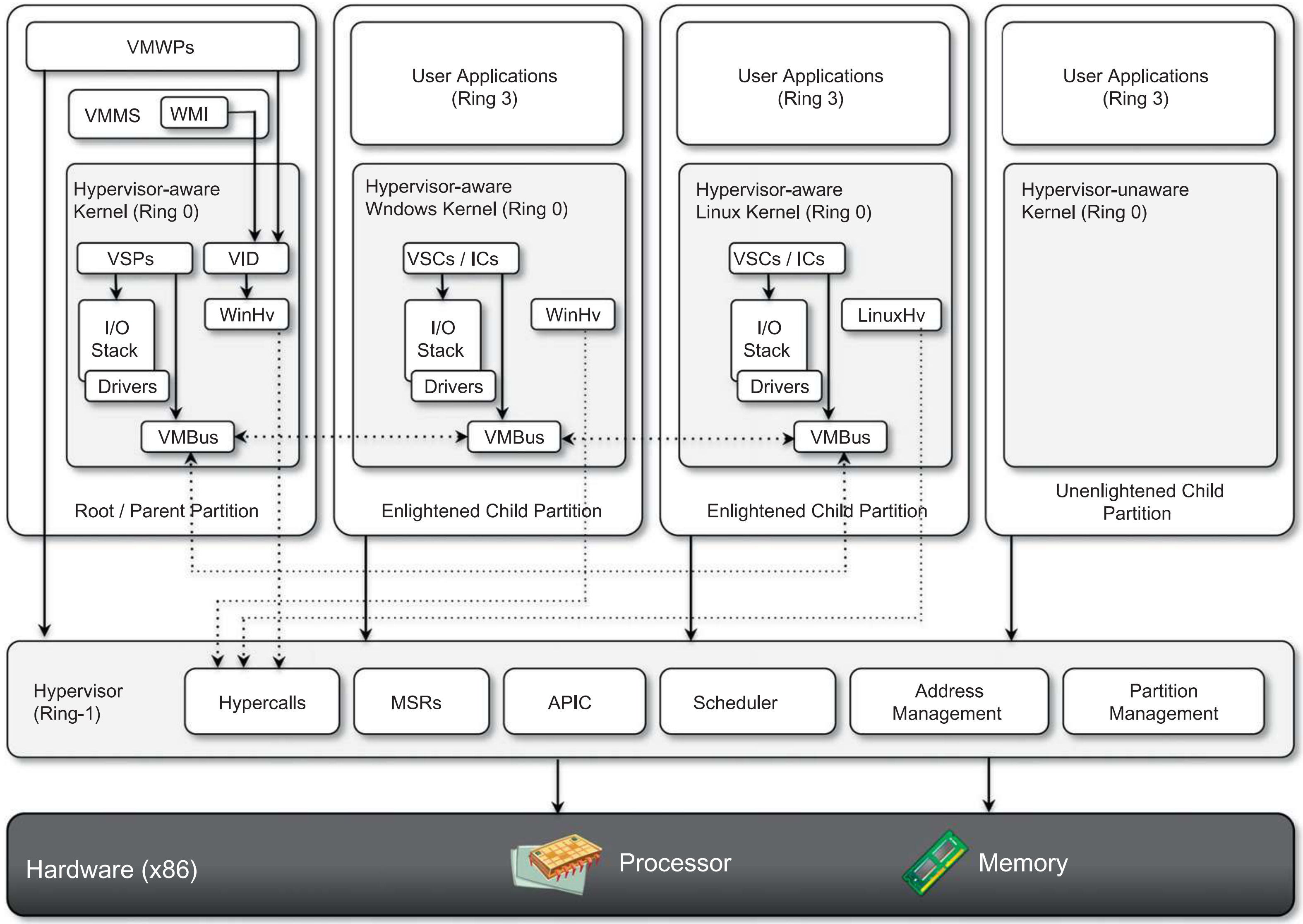
VMWPs
VMMS
WMI
Hypervisor-aware Kernel (Ring 0)
VSPs
VID
I/O Stack
Drivers
WinHv
VMBus
Root / Parent Partition
User Applications (Ring 3)
Hypervisor-aware Wndows Kernel (Ring 0)
VSCs / ICs
I/O Stack
Drivers
WinHv
VMBus
Enlightened Child Partition
User Applications (Ring 3)
Hypervisor-aware Linux Kernel (Ring 0)
VSCs / ICs
I/O Stack
Drivers
LinuxHv
VMBus
Enlightened Child Partition
User Applications (Ring 3)
Hypervisor-unaware Kernel (Ring 0)
Unenlightened Child Partition
Hypervisor (Ring-1)
Hypercalls
MSRs
APIC
Scheduler
Address Management
Partition Management
Hardware (x86)
Processor
Memory

FIGURE 3.17

Microsoft Hyper-V architecture.

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 with Xen. 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.

Enlightened I/O and synthetic devices

Enlightened I/O provides an optimized way to perform I/O operations, allowing guest operating systems to leverage an interpartition communication channel rather than traversing the hardware emulation stack provided by the hypervisor. This option is only available to guest operating systems that are hypervisor aware. Enlightened I/O leverages VMBus, an interpartition communication channel that is used to exchange data between partitions (child and parent) and is utilized mostly for the implementation of virtual device drivers for guest operating systems.

The architecture of Enlightened I/O is described in Figure 3.17. There are three fundamental components: *VMBus*, *Virtual Service Providers (VSPs)*, and *Virtual Service Clients (VSCs)*. VMBus implements the channel and defines the protocol for communication between partitions. VSPs are kernel-level drivers that are deployed in the parent partition and provide access to the corresponding hardware devices. These interact with VSCs, which represent the virtual device drivers (also called *synthetic drivers*) seen by the guest operating systems in the child partitions. Operating systems supported by Hyper-V utilize this preferred communication channel to perform I/O for storage, networking, graphics, and input subsystems. This also results in enhanced performance in child-to-child I/O as a result of virtual networks between guest operating systems. Legacy operating systems, which are not hypervisor aware, can still be run by Hyper-V but rely on device driver emulation, which is managed by the hypervisor and is less efficient.

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 WMI[9] 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.

3.6.3.2 Cloud computing and infrastructure management

Hyper-V constitutes the basic building block of Microsoft virtualization infrastructure. Other components contribute to creating a fully featured platform for server virtualization.

To increase the performance of virtualized environments, a new version of Windows Server 2008, called *Windows Server Core*, has been released. This is a specific version of the operating

[9]*WMI* stands for *Windows Management Instrumentation*. This is a specification used in the Windows environment to provide access to the underlying hardware. The specification is based on providers that give authorized clients access to a specific subsystem of the hardware.

system with a reduced set of features and a smaller footprint. In particular, Windows Server Core has been designed by removing those features, which are not required in a server environment, such as the GUI component and other bulky components such as the .NET Framework and all the applications developed on top of it (for example, PowerShell). This design decision has both advantages and disadvantages. On the plus side, it allows for reduced maintenance (i.e., fewer software patches), reduced attack surface, reduced management, and less disk space. On the negative side, the embedded features are reduced. Still, there is the opportunity to leverage all the "removed features" by means of remote management from a fully featured Windows installation. For instance, administrators can use the PowerShell to remotely manage the Windows Server Core installation through WMI.

Another component that provides advanced management of virtual machines is *System Center Virtual Machine Manager (SCVMM) 2008*. This is a component of the Microsoft System Center suite, which brings into the suite the virtual infrastructure management capabilities from an IT life-cycle point of view. Essentially, SCVMM complements the basic features offered by Hyper-V with management capabilities, including:

- Management portal for the creation and management of virtual instances
- Virtual to Virtual (V2V) and Physical to Virtual (P2V) conversions
- Delegated administration
- Library functionality and deep PowerShell integration
- Intelligent placement of virtual machines in the managed environment
- Host capacity management

SCVMM has also been designed to work with other virtualization platforms such as VMware vSphere (ESX servers) but benefits most from the virtual infrastructure management implemented with Hyper-V.

3.6.3.3 Observations

Compared with Xen and VMware, Hyper-V is a hybrid solution because it leverages both paravirtualization techniques and full hardware virtualization.

The basic architecture of the hypervisor is based on paravirtualized architecture. The hypervisor exposes its services to the guest operating systems by means of hypercalls. Also, paravirtualized kernels can leverage VMBus for fast I/O operations. Moreover, partitions are conceptually similar to domains in Xen: The parent partition maps Domain 0, while child partitions map Domains U. The only difference is that the Xen hypervisor is installed on bare hardware and filters all the access to the underlying hardware, whereas Hyper-V is installed as a role in the existing operating system, and the way it interacts with partitions is quite similar to the strategy implemented by VMware, as we discussed.

The approach adopted by Hyper-V has both advantages and disadvantages. The advantages reside in a flexible virtualization platform supporting a wide range of guest operating systems. The disadvantages are represented by both hardware and software requirements. Hyper-V is compatible only with Windows Server 2008 and newer Windows Server platforms running on a x64 architecture. Moreover, it requires a 64-bit processor supporting hardware-assisted virtualization and data execution prevention. Finally, as noted above, Hyper-V is a role that can be installed on a existing operating system, while vSphere and Xen can be installed on the bare hardware.

SUMMARY

The term *virtualization* is a large umbrella under which a variety of technologies and concepts are classified. The common root of all the forms of virtualization is the ability to provide the illusion of a specific environment, whether a runtime environment, a storage facility, a network connection, or a remote desktop, by using some kind of emulation or abstraction layer. All these concepts play a fundamental role in building cloud computing infrastructure and services in which hardware, IT infrastructure, applications, and services are delivered on demand through the Internet or more generally via a network connection.

Review questions

1. What is virtualization and what are its benefits?
2. What are the characteristics of virtualized environments?
3. Discuss classification or taxonomy of virtualization at different levels.
4. Discuss the machine reference model of execution virtualization.
5. What are hardware virtualization techniques?
6. List and discuss different types of virtualization.
7. What are the benefits of virtualization in the context of cloud computing?
8. What are the disadvantages of virtualization?
9. What is Xen? Discuss its elements for virtualization.
10. Discuss the reference model of full virtualization.
11. Discuss the architecture of Hyper-V. Discuss its use in cloud computing.