UNIT II

VIRTUAL MACHINES AND VIRTUALIZATION OF CLUSTERS AND DATA CENTERS

SYLLABUS:

Virtual Machines and Virtualization of Clusters and Data Centers

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

3.1 IMPLEMENTATION LEVELS OF VIRTUALIZATION

Virtualization is a computer architecture technology by which multiple virtual machines (VMs) are multiplexed in the same hardware machine. The purpose of a VM is to enhance resource sharing by many users and improve computer performance in terms of resource utilization and application flexibility. Hardware resources (CPU, memory, I/O devices, etc.) or software resources (operating system and software libraries) can be virtualized in various functional layers. The idea is to separate the hardware from the software to yield better system efficiency.

3.1.1 Levels of Virtualization Implementation

A traditional computer runs with a host operating system specially tailored for its hardware architecture, as shown in Figure 3.1(a). After virtualization, different user applications managed by their own operating systems (guest OS) can run on the same hardware, independent of the host OS. This is often done by adding additional software, called a virtualization layer known as hypervisor or virtual machine monitor (VMM)as shown in Figure 3.1(b). The VMs are shown in the upper boxes, where applications run with their own guest OS over the virtualized CPU, memory, and I/O resources. The virtualization software creates the abstraction of VMs by interposing a virtualization layer at various levels of a computer system. Common virtualization layers include the instruction set architecture (ISA) level, hardware level, operating system level, library support level, and application level (see Figure 3.2).

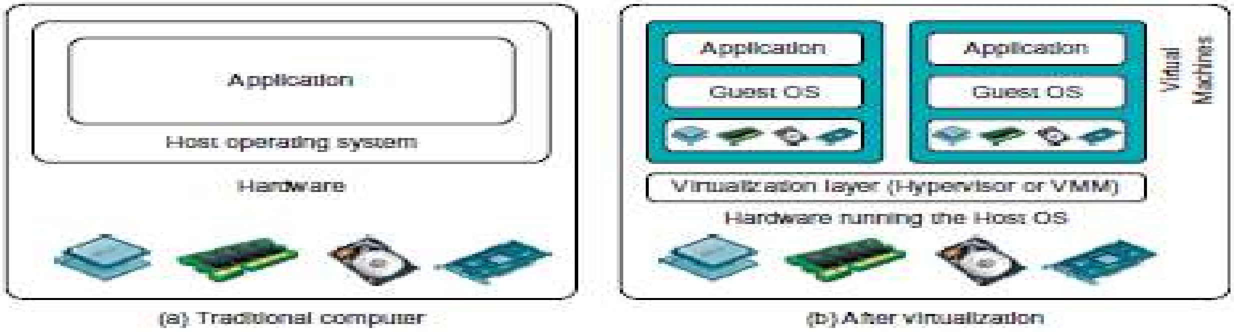

FIGURE 3.1

The architecture of a computer system before and after virtualization, where VMM stands for virtual machine monitor.

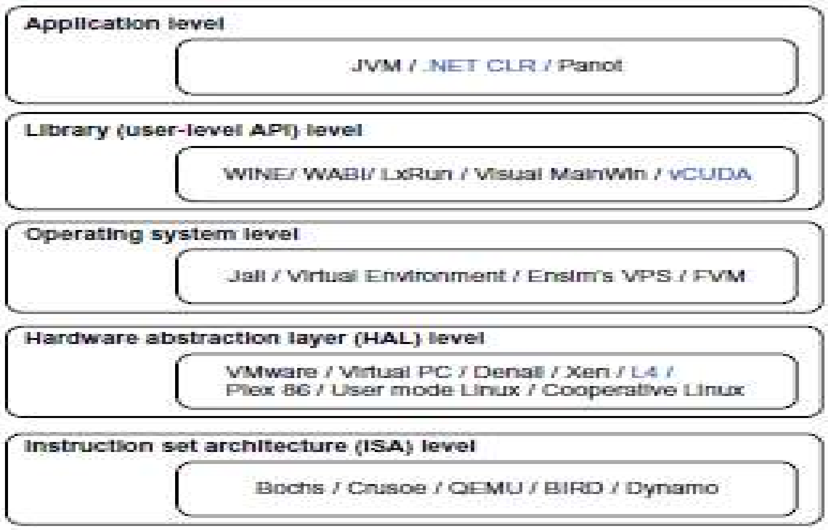

FIGURE 3.2

Virtualization ranging from hardware to applications in five abstraction levels.

3.1.1.1 Instruction Set Architecture Level

At the ISA level, virtualization is performed by emulating a given ISA by the ISA of the host machine. The basic emulation method is through code interpretation. An interpreter program interprets the source instructions to target instructions one by one. One source instruction may require tens or hundreds of native target instructions to perform its function. Obviously, this process is relatively slow. For better performance, dynamic binary translation is desired. This approach translates basic blocks of dynamic source instructions to target instructions. The basic blocks can also be extended to program traces or super blocks to increase translation efficiency. Instruction set emulation requires binary translation and optimization. A virtual instruction set architecture (V-ISA) thus requires adding a processor-specific software translation layer to the compiler.

3.1.1.2 Hardware Abstraction Level

Hardware-level virtualization is performed right on top of the bare hardware. On the one hand, this approach generates a virtual hardware environment for a VM. On the other hand, the process manages the underlying hardware through virtualization. The idea is to virtualize a computer's resources, such as its processors, memory, and I/O devices. The intention is to upgrade the hardware utilization rate by multiple users concurrently.

3.1.1.3 Operating System Level

This refers to an abstraction layer between traditional OS and user applications. OS-level virtualization creates isolated containers on a single physical server and the OS instances to utilize the hardware and software in data centers. The containers behave like real servers. OS-level virtualization is commonly used in creating virtual hosting environments to allocate hardware resources among a large number of mutually distrusting users.

3.1.1.4 Library Support Level

Most applications use APIs exported by user-level libraries rather than using lengthy system calls by the OS. Since most systems provide well-documented APIs, such an interface becomes another candidate for virtualization. Virtualization with library interfaces is possible by controlling the communication link between applications and the rest of a system through API hooks.

3.1.1.5 User-Application Level

Virtualization at the application level virtualizes an application as a VM. On a traditional OS, an application often runs as a process. Therefore, application-level virtualization is also known as process-level virtualization. The most popular approach is to deploy high level language (HLL) VMs. In this scenario, the virtualization layer sits as an application program on top of the operating system, and the layer exports an abstraction of a VM that can run programs written and compiled to a particular abstract machine definition. Any program written in the HLL and compiled for this VM will be able to run on it.

3.1.2 VMM Design Requirements and Providers

As mentioned earlier, hardware-level virtualization inserts a layer between real hardware and traditional operating systems. This layer is commonly called the Virtual Machine Monitor (VMM) and it manages the hardware resources of a computing system. Each time programs access the hardware the VMM captures the process. In this sense, the VMM acts as a traditional OS. One hardware component, such as the CPU, can be virtualized as several virtual copies. There are three requirements for a VMM.

First, a VMM should provide an environment for programs which is essentially identical to the original machine.

Second, programs run in this environment should show, at worst, only minor decreases in speed.

Third, a VMM should be in complete control of the system resources. Any program run under a VMM should exhibit a function identical to that which it runs on the original machine directly. Two possible exceptions in terms of differences are permitted with this requirement: differences caused by the availability of system resources and differences caused by timing dependencies.

A VMM should demonstrate efficiency in using the VMs. Compared with a physical machine, no one prefers a VMM if its efficiency is too low. Traditional emulators and complete software interpreters (simulators) emulate each instruction by means of functions or macros. Such a method provides the most flexible solutions for VMMs. However, emulators or simulators are too slow to be used as real machines. To guarantee the efficiency of a VMM, a statistically dominant subset of the virtual processor's instructions needs to be executed directly by the real processor, with no software involvement by the VMM. Complete control of these resources by a VMM includes the following aspects: (1) The VMM is responsible for allocating hardware resources for programs; (2) it is not possible for a program to access any resource not explicitly allocated to it; and (3) it is possible under certain circumstances for a VMM to regain control of resources already allocated. If a processor is not designed to support virtualization primarily, it is necessary to modify the hardware to satisfy the three requirements for a VMM. This is known as hardware-assisted virtualization.

3.1.3 Virtualization Support at the OS Level

With the help of VM technology, a new computing mode known as cloud computing is emerging. Cloud computing is transforming the computing landscape by shifting the hardware and staffing costs of managing a computational center to third parties, just like banks. However, cloud computing has at least two challenges. The first is the ability to use a variable number of physical machines and VM instances depending on the needs of a problem. The second challenge concerns the slow operation of instantiating new VMs. Currently, new VMs originate either as fresh boots or as replicates of a template VM, unaware of the current application state. Therefore, to better support cloud computing, a large amount of research and development should be done.

3.1.3.1 Why OS-Level Virtualization?

In a cloud computing environment, perhaps thousands of VMs need to be initialized simultaneously. Besides slow operation, storing the VM images also becomes an issue. As a matter of fact, there is considerable repeated content among VM images. Moreover, full virtualization at the hardware level also has the disadvantages of slow performance and low density, and the need for para-virtualization to modify the guest OS. To reduce the performance overhead of hardware-level virtualization, even hardware modification is needed. OS-level virtualization provides a feasible solution for these hardware-level virtualization issues.

Operating system virtualization inserts a virtualization layer inside an operating system to partition a machine's physical resources. It enables multiple isolated VMs within a single operating system kernel. This kind of VM is often called a virtual execution environment (VE), Virtual Private System (VPS), or simply container. From the user's point of view, VEs look like real servers. This means a VE has its own set of processes, file system, user accounts, network interfaces with IP addresses, routing tables, firewall rules, and other personal settings. Although VEs can be customized for different people, they share the same operating system kernel. Therefore, OS-level virtualization is also called single-OS image virtualization.

3.1.3.2 Advantages of OS Extensions

(1) VMs at the operating system level have minimal startup/shutdown costs, low resource requirements, and high scalability; and (2) for an OS-level VM, it is possible for a VM and its host environment to synchronize state changes when necessary. These benefits can be achieved via two mechanisms of OS-level virtualization: (1) All OS-level VMs on the same physical machine share a single operating system kernel; and (2) the virtualization layer can be designed in a way that allows processes in VMs to access as many resources of the host machine as possible, but never to modify them.

3.1.3.3 Disadvantages of OS Extensions

The main disadvantage of OS extensions is that all the VMs at operating system level on a single container must have the same kind of guest operating system. That is, although different OS-level VMs may have different operating system distributions, they must pertain to the same operating system family.

3.1.4 Middleware Support for Virtualization

Library-level virtualization is also known as user-level Application Binary Interface (ABI) or API emulation. This type of virtualization can create execution environments for running alien programs on a platform rather than creating a VM to run the entire operating system. API call interception and remapping are the key functions performed. The various library-level virtualization systems are the Windows Application Binary Interface (WABI), lxrun, WINE, Visual MainWin, and vCUDA .

The WABI offers middleware to convert Windows system calls to Solaris system calls. Lxrun is really a system call emulator that enables Linux applications written for x86 hosts to run on UNIX systems. Similarly, Wine offers library support for virtualizing x86 processors to run Windows applications on UNIX hosts. Visual MainWin offers a compiler support system to develop Windows applications using Visual Studio to run on some UNIX hosts.

3.2 VIRTUALIZATION STRUCTURES/TOOLS AND MECHANISMS

In general, there are three typical classes of VM architecture. Figure 3.1 showed the architectures of a machine before and after virtualization. Before virtualization, the operating system manages the hardware. After virtualization, a virtualization layer is inserted between the hardware and the operating system. In such a case, the virtualization layer is responsible for converting portions of the real hardware into virtual hardware. Therefore, different operating systems such as Linux and Windows can run on the same physical machine, simultaneously. Depending on the position of the virtualization layer, there are several classes of VM architectures, namely the **hypervisor architecture, para-virtualization, and host-based virtualization**. The hypervisor is also known as the VMM (Virtual Machine Monitor). They both perform the same virtualization operations.

3.2.1 Hypervisor and Xen Architecture

The hypervisor supports hardware-level virtualization (see Figure 3.1(b)) on bare metal devices like CPU, memory, disk and network interfaces. The hypervisor software sits directly between the physical hardware and its OS. This virtualization layer is referred to as either the VMM or the hypervisor. The hypervisor provides hyper calls for the guest OSes and applications. Depending on the functionality, a hypervisor can assume a micro-kernel architecture like the Microsoft Hyper-V or it can assume a monolithic hypervisor architecture like the VMware ESX for server virtualization.

A micro-kernel hypervisor includes only the basic and unchanging functions (such as physical memory management and processor scheduling). The device drivers and other changeable components are outside the hypervisor. A monolithic hypervisor implements all the aforementioned functions, including those of the device drivers. Therefore, the size of the hypervisor code of a micro-kernel hypervisor is smaller than that of a monolithic hypervisor. Essentially, a hypervisor must be able to convert physical devices into virtual resources dedicated for the deployed VM to use.

3.2.2 Binary Translation with Full Virtualization

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

3.2.2.1 Full Virtualization

With full virtualization, noncritical instructions run on the hardware directly while critical instructions are discovered and replaced with traps into the VMM to be emulated by software. Both the hypervisor and VMM approaches are considered full virtualization. Noncritical instructions do not control hardware or threaten the security of the system, but critical instructions do. Therefore, running noncritical instructions on hardware not only can promote efficiency, but also can ensure system security.

3.2.2.2 Binary Translation of Guest OS Requests Using a VMM

This approach was implemented by VMware and many other software companies. As shown in Figure 3.6, VMware puts the VMM at Ring 0 and the guest OS at Ring 1. The VMM scans the instruction stream and identifies the privileged, control- and behavior-sensitive instructions. When these instructions are identified, they are trapped into the VMM, which emulates the behavior of these instructions. The method used in this emulation is called binary translation. Therefore, full virtualization combines binary translation and direct execution.

FIGURE 3.6
Indirect execution of complex instructions via binary translation of guest OS requests using the VMM plus direct execution of simple instructions on the same host.

3.2.2.3 Host-Based Virtualization

An alternative VM architecture is to install a virtualization layer on top of the host OS. This host OS is still responsible for managing the hardware. The guest OSes are installed and run on top of the virtualization layer. Dedicated applications may run on the VMs. Certainly, some other applications can also run with the host OS directly. This host based architecture has some distinct advantages, as enumerated next. First, the user can install this VM architecture without modifying the host OS. The virtualizing software can rely on the host OS to provide device drivers and other low-level services. This will simplify the VM design and ease its deployment. Second, the host-based approach appeals to many host machine configurations. Compared to the hypervisor/VMM architecture, the performance of the host-based architecture may also be low. When an application requests hardware access, it involves four layers of mapping which downgrades performance significantly. When the ISA of a guest OS is different from the ISA of the underlying hardware, binary translation must be adopted. Although the host-based architecture has flexibility, the performance is too low to be useful in practice.

3.2.3 Para-Virtualization with Compiler Support

Para-virtualization needs to modify the guest operating systems. A para-virtualized VM provides special APIs requiring substantial OS modifications in user applications. Performance degradation is a critical issue of a virtualized system. No one wants to use a VM if it is much slower than using a physical machine. The virtualization layer can be inserted at different positions in a machine software stack. However, para-virtualization attempts to reduce the virtualization overhead, and thus improve performance by modifying only the guest OS kernel.

Figure 3.7 illustrates the concept of a para-virtualized VM architecture. The guest operating systems are para-virtualized. They are assisted by an intelligent compiler to replace the nonvirtualizable OS instructions by hypercalls as illustrated in Figure 3.8. The traditional x86 processor offers four instruction execution rings: Rings 0, 1, 2, and 3. The lower the ring number, the higher the privilege of instruction being executed. The OS is responsible for managing the hardware and the privileged instructions to execute at Ring 0, while user-level applications run at Ring 3. The best example of para-virtualization is the KVM to be described below.

3.2.3.1 Para-Virtualization Architecture

When the x86 processor is virtualized, a virtualization layer is inserted between the hardware and the OS.

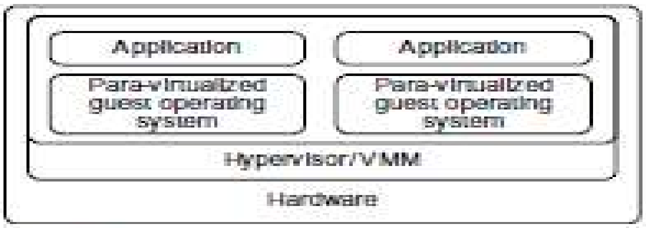

FIGURE 3.7

Para-virtualized VM architecture, which involves modifying the guest OS kernel to replace nonvirtualizable instructions with hypercalls for the hypervisor or the VMM to carry out the virtualization process (See Figure 3.8 for more details.)

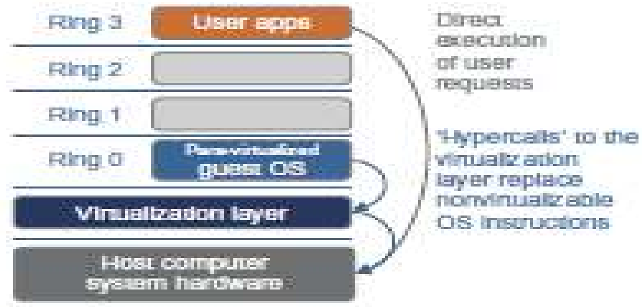

FIGURE 3.8

The use of a para-virtualized guest OS assisted by an intelligent compiler to replace nonvirtualizable OS instructions by hypercalls.

(Courtesy of VMWare [71])

According to the x86 ring definition, the virtualization layer should also be installed at Ring 0. Different instructions at Ring 0 may cause some problems. In Figure 3.8, we show that para-virtualization replaces non-virtualizable instructions with hyper calls that communicate directly with the hypervisor or VMM. However, when the guest OS kernel is modified for virtualization, it can no longer run on the hardware directly. Compared with full virtualization, para-virtualization is relatively easy and more practical. The main problem in full virtualization is its low performance in binary translation. To speed up binary translation is difficult. Therefore, many virtualization products employ the para-virtualization architecture. The popular Xen, KVM, and VMware ESX are good examples.

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

3.3.1 Hardware Support for Virtualization

Modern operating systems and processors permit multiple processes to run simultaneously. If there is no protection mechanism in a processor, all instructions from different processes will access the hardware directly and cause a system crash. Therefore, all processors have at least two modes, user mode and supervisor mode, to ensure controlled access of critical hardware. Instructions running in supervisor mode are called privileged instructions. Other instructions are unprivileged instructions. In a virtualized enviro-nment, it is more difficult to make OS and applications run correctly because there are more layers in the machine stack.

3.3.2 CPU Virtualization

A VM is a duplicate of an existing computer system in which a majority of the VM instructions are executed on the host processor in native mode. Thus, unprivileged instructions of VMs run directly on the host machine for higher efficiency. Other critical instructions should be handled carefully for correctness and stability. The critical instructions are divided into three categories: privileged instructions, control sensitive instructions, and behavior-sensitive instructions. Privileged instructions execute in a privileged mode and will be trapped if executed outside this mode. Control-sensitive instructions attempt to change the configuration of resources used. Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.

A CPU architecture is virtualizable if it supports the ability to run the VM's privileged and unprivileged instructions in the CPU's user mode while the VMM runs in supervisor mode. When the privileged instructions including control and behavior-sensitive instructions of a VM are executed, they are trapped in the VMM. In this case, the VMM acts as a unified mediator for hardware access from different VMs to guarantee the correctness and stability of the whole system. However, not all CPU architectures are virtualizable.

3.3.2.1 Hardware-Assisted CPU Virtualization

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

3.3.3 Memory Virtualization

Virtual memory virtualization is similar to the virtual memory support provided by modern operating systems. In a traditional execution environment, the operating system maintains mappings of virtual memory to machine memory using page tables, which is a one-stage mapping from virtual memory to machine memory. All modern x86 CPUs include a memory management unit (MMU) and a translation lookaside buffer (TLB) to optimize virtual memory performance. However, in a virtual execution environment, virtual memory virtualization involves sharing the physical system memory in RAM and dynamically allocating it to the physical memory of the VMs. That means a two-stage mapping process should be maintained by the guest OS and the VMM, respectively: virtual memory to physical memory and physical memory to machine memory. Furthermore, MMU virtualization should be supported, which is transparent to the guest OS. The guest OS continues to control the mapping of virtual addresses to the physical memory addresses of VMs. But the guest OS cannot directly access the actual machine memory. The VMM is responsible for mapping the guest physical memory to the actual machine memory. Figure 3.12 shows the two-level memory mapping procedure.

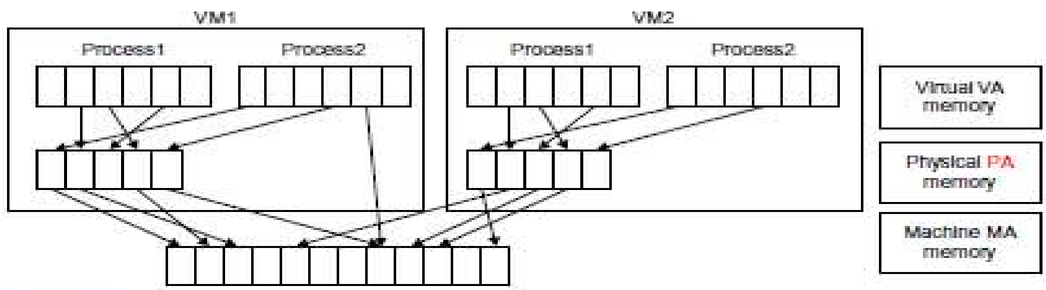

FIGURE 3.12
Two-level memory mapping procedure.

Since each page table of the guest OSes has a separate page table in the VMM corresponding to it, the VMM page table is called the shadow page table. Nested page tables add another layer of indirection to virtual memory. The MMU already handles virtual-to-physical translations as defined by the OS. Then the physical memory addresses are translated to machine addresses using another set of page tables defined by the hypervisor. Since modern operating systems maintain a set of page tables for every process, the shadow page tables will get flooded. Consequently, the performance overhead and cost of memory will be very high.

VMware uses shadow page tables to perform virtual-memory-to-machine-memory address translation. Processors use TLB hardware to map the virtual memory directly to the machine memory to avoid the two levels of translation on every access. When the guest OS changes the virtual memory to a physical memory mapping, the VMM updates the shadow page tables to enable a direct lookup.

3.3.4 I/O Virtualization

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

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

The para-virtualization method of I/O virtualization is typically used in Xen. It is also known as the split driver model consisting of a frontend driver and a backend driver. The frontend driver is running in Domain U and the backend driver is running in Domain 0. They interact with each other via a block of shared memory. The frontend driver manages the I/O requests of the guest OS and the backend driver is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs. Although para-I/O-virtualization achieves better device performance than full device emulation, it comes with a higher CPU overhead.

Direct I/O virtualization lets the VM access devices directly. It can achieve close-to-native performance without high CPU costs. However, current direct I/O virtualization implementations focus on networking for mainframes. There are a lot of challenges for commodity hardware devices.

3.3.5 Virtualization in Multi-Core Processors

Virtualizing a multi-core processor is relatively more complicated than virtualizing a uni-core processor. Though multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip, muti-core virtualiuzation has raised some new challenges to computer architects, compiler constructors, system designers, and application programmers. There are mainly two difficulties: Application programs must be parallelized to use all cores fully, and software must explicitly assign tasks to the cores, which is a very complex problem.

Concerning the first challenge, new programming models, languages, and libraries are needed to make parallel programming easier. The second challenge has spawned research involving scheduling algorithms and resource management policies. The main challenge called dynamic heterogeneity is emerging to mix the fat CPU core and thin GPU cores on the same chip, which further complicates the multi-core or many-core resource management. The dynamic heterogeneity of hardware infrastructure mainly comes from less reliable transistors and increased complexity in using the transistors.

3.3.5.1 Physical versus Virtual Processor Cores

A multi-core virtualization method is to allow hardware designers to get an abstraction of the low-level details of the processor cores. This technique alleviates the burden and inefficiency of managing hardware resources by software. It is located under the ISA and remains unmodified by the operating system or VMM (hypervisor).

3.3.5.2 Virtual Hierarchy

The emerging many-core chip multi processors (CMPs) provides a new computing landscape. Instead of supporting time-sharing jobs on one or a few cores, we can use the abundant cores in a space-sharing, where single-threaded or multithreaded jobs are simultaneously assigned to separate groups of cores for long time intervals. To optimize for space-shared workloads, they propose using virtual hierarchies to overlay a coherence and caching hierarchy onto a physical processor. Unlike a fixed physical hierarchy, a virtual hierarchy can adapt to fit how the work is space shared for improved performance and performance isolation.

Today's many-core CMPs use a physical hierarchy of two or more cache levels that statically determine the cache allocation and mapping. A virtual hierarchy is a cache hierarchy that can adapt to fit the workload or mix of workloads. The hierarchy's first level locates data blocks close to the cores needing them for faster access, establishes a shared-cache domain, and establishes a point of coherence for faster communication.

When a miss leaves a tile, it first attempts to locate the block (or sharers) within the first level. The first level can also provide isolation between independent workloads. A miss at the L1 cache can invoke the L2 access.

3.4 VIRTUAL CLUSTERS AND RESOURCE MANAGEMENT

A physical cluster is a collection of servers (physical machines) interconnected by a physical network such as a LAN. The three critical design issues of virtual clusters: live migration of VMs, memory and file migrations, and dynamic deployment of virtual clusters.

When a traditional VM is initialized, the administrator needs to manually write configuration information or specify the configuration sources. When more VMs join a network, an inefficient configuration always causes problems with overloading or underutilization.. Most virtualization platforms, including Xen Server and VMware ESX Server, support a bridging mode which allows all domains to appear on the network as individual hosts. By using this mode, VMs can communicate with one another freely through the virtual network interface card and configure the network automatically.

3.4.1 Physical versus Virtual Clusters

Virtual clusters are built with VMs installed at distributed servers from one or more physical clusters. The VMs in a virtual cluster are interconnected logically by a virtual network across several physical networks.

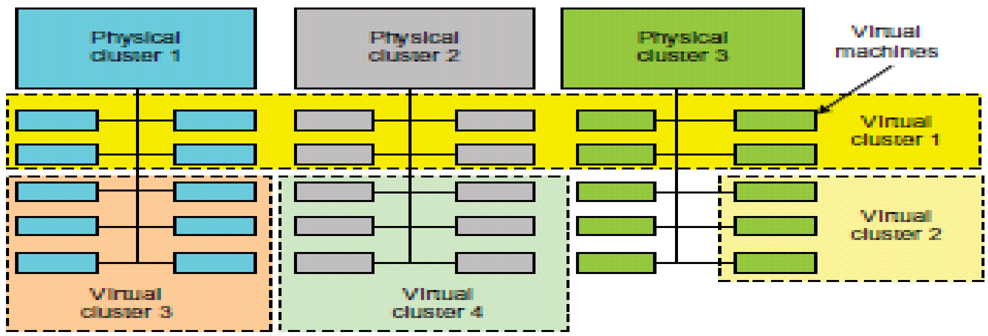

FIGURE 3.18

A cloud platform with four virtual clusters over three physical clusters shaded differently.

Figure 3.18 illustrates the concepts of virtual clusters and physical clusters. Each virtual cluster is formed with physical machines or a VM hosted by multiple physical clusters. The virtual cluster boundaries are shown as distinct boundaries. The provisioning of VMs to a virtual cluster is done dynamically to have the following interesting properties:

- The virtual cluster nodes can be either physical or virtual machines. Multiple VMs running with different OS's can be deployed on the same physical node.
- A VM runs with a guest OS, which is often different from the host OS, that manages the resources in the physical machine, where the VM is implemented.
- The purpose of using VMs is to consolidate multiple functionalities on the same server. This will greatly enhance server utilization and application flexibility.
- VMs can be colonized (replicated) in multiple servers for the purpose of promoting distributed parallel-ism, fault tolerance, and disaster recovery.
- The size (number of nodes) of a virtual cluster can grow or shrink dynamically, similar to the way an overlay network varies in size in a peer-to-peer (P2P) network.

- The failure of any physical nodes may disable some VMs installed on the failing nodes. But the failure of VMs will not pull down the host system.

Since system virtualization has been widely used, it is necessary to effectively manage VMs running on a mass of physical computing nodes (also called virtual clusters) and consequently build a high-performance virtualized computing environment. This involves virtual cluster deployment, monitoring and management over large-scale clusters, as well as resource scheduling, load balancing, server consolidation, fault tolerance, and other techniques. The different node colors in Figure 3.18 refer to different virtual clusters. In a virtual cluster system, it is quite important to store the large number of VM images efficiently.

Three physical clusters are shown on the left side of Figure 3.18. Four virtual clusters are created on the right, over the physical clusters. The physical machines are also called host systems. In contrast, the VMs are guest systems. The host and guest systems may run with different operating systems. Each VM can be installed on a remote server or replicated on multiple servers belonging to the same or different physical clusters. The boundary of a virtual cluster can change as VM nodes are added, removed, or migrated dynamically over time.

3.4.1.1 Fast Deployment and Effective Scheduling

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

The live migration of VMs allows workloads of one node to transfer to another node. However, it does not guarantee that VMs can randomly migrate among themselves. In fact, the potential overhead caused by live migrations of VMs cannot be ignored. The overhead may have serious negative effects on cluster utilization, throughput, and QoS issues. Another advantage of virtualization is load balancing of applications in a virtual cluster. Load balancing can be achieved using the load index and frequency of user logins. Dynamically adjusting loads among nodes by live migration of VMs is desired, when the loads on cluster nodes become quite unbalanced.

3.4.1.2 High-Performance Virtual Storage

The template VM can be distributed to several physical hosts in the cluster to customize the VMs. In addition, existing software packages reduce the time for customization as well as switching virtual environments. It is important to efficiently manage the disk spaces occupied by template software packages. Some storage architecture design can be applied to reduce duplicated blocks in a distributed file system of virtual clusters. Hash values are used to compare the contents of data blocks. Users have their own profiles which store the identification of the data blocks for corresponding VMs in a user-specific virtual cluster. New blocks are created when users modify the corresponding data. Newly created blocks are identified in the users' profiles.

Basically, there are four steps to deploy a group of VMs onto a target cluster: preparing the disk image, configuring the VMs, choosing the destination nodes, and executing the VM deployment command on every host. Many systems use templates to simplify the disk image preparation process. A template a disk image that includes a preinstalled operating system with or without certain application software.

Users choose a proper template according to their requirements and make a duplicate of it as their own disk image. Templates could implement the COW (Copy on Write) format. A new COW backup file is very small and easy to create and transfer. Therefore, it definitely reduces disk space consumption. In addition, VM deployment time is much shorter than that of copying the whole raw image file.

Every VM is configured with a name, disk image, network setting, and allocated CPU and memory. One needs to record each VM configuration into a file. However, this method is inefficient when managing a large group of VMs. VMs with the same configurations could use pre-edited profiles to simplify the process. In this scenario, the system configures the VMs according to the chosen profile. Most configuration items use the same settings, while some of them, such as UUID, VM name, and IP address, are assigned with automatically calculated values. Normally, users do not care which host is running their VM. A strategy to choose the proper destination host for any VM is needed. The deployment principle is to fulfill the VM requirement and to balance workloads among the whole host network.

3.4.2 Live VM Migration Steps and Performance Effects

In a cluster built with mixed nodes of host and guest systems, the normal method of operation is to run everything on the physical machine. When a VM fails, its role could be replaced by another VM on a different node, as long as they both run with the same guest OS. In other words, a physical node can fail over to a VM on another host.

This is different from physical-to-physical failover in a traditional physical cluster. The advantage is enhanced failover flexibility. The potential drawback is that a VM must stop playing its role if its residing host node fails. However, this problem can be mitigated with VM life migration.

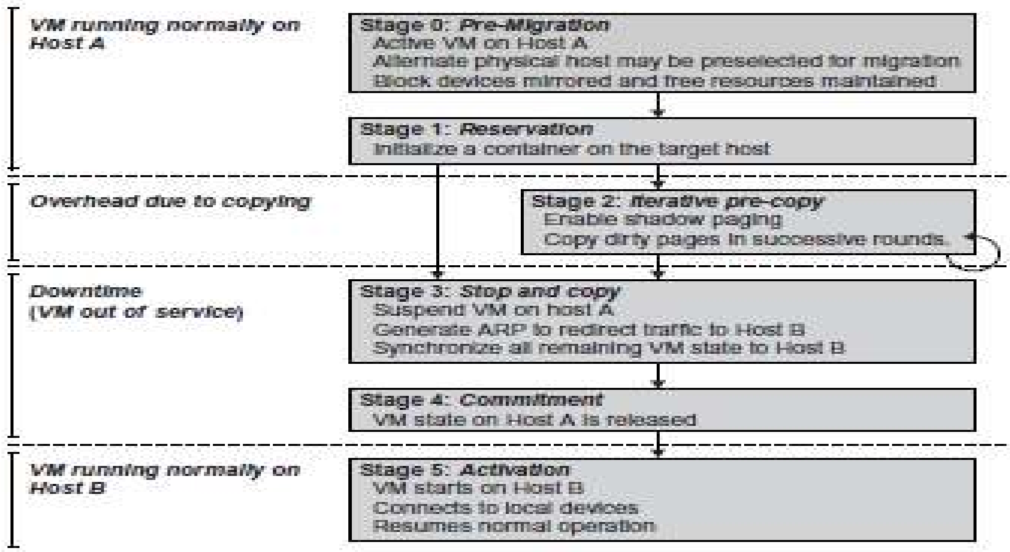

FIGURE 3.20

Live migration process of a VM from one host to another.

Figure 3.20 shows the process of life migration of a VM from host A to host B. The migration copies the VM state file from the storage area to the host machine.

There are four ways to manage a virtual cluster. First, you can use a guest-based manager, by which the cluster manager resides on a guest system. In this case, multiple VMs form a virtual cluster.

Second, you can build a cluster manager on the host systems. The host-based manager supervises the guest systems and can restart the guest system on another physical machine. A third way to manage a virtual cluster is to use an independent cluster manager on both the host and guest systems. This will make infrastructure management more complex, however.

Finally, you can use an integrated cluster on the guest and host systems. This means the manager must be designed to distinguish between virtualized resources and physical resources.

As shown in Figure 3.20, live migration of a VM consists of the following six steps:

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

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

Step 3: Suspend the VM and copy the last portion of the data. The migrating VM's execution is suspended when the last round's memory data is transferred. Other non memory data such as CPU and network states should be sent as well. During this step, the VM is stopped and its applications will no longer run. This "service unavailable" time is called the "downtime" of migration, which should be as short as possible so that it can be negligible to users.
Steps 4 and 5: Commit and activate the new host. After all the needed data is copied, on the destination host, the VM reloads the states and recovers the execution of programs in it, and the service provided by this VM continues. Then the network connection is redirected to the new VM and the dependency to the source host is cleared. The whole migration process finishes by removing the original VM from the source host.

3.4.3 Migration of Memory, Files, and Network Resources

Since clusters have a high initial cost of ownership, including space, power conditioning, and cooling equipment, leasing or sharing access to a common cluster is an attractive solution when demands vary over time. Early configuration and management systems focus on expressive and scalable mechanisms for defining clusters for specific types of service, and physically partition cluster nodes among those types. When one system migrates to another physical node, we should consider the following issues.

3.4.3.1 Memory Migration

Moving the memory instance of a VM from one physical host to another can be approached in any number of ways. Memory migration can be in a range of hundreds of megabytes to a few gigabytes in a typical system today, and it needs to be done in an efficient manner. The Internet Suspend-Resume (ISR) technique exploits temporal locality as memory states are likely to have considerable overlap in the suspended and the resumed instances of a VM. Temporal locality refers to the fact that the memory states differ only by the amount of work done since a VM was last suspended before being initiated for migration.

To exploit temporal locality, each file in the file system is represented as a tree of small sub-files. A copy of this tree exists in both the suspended and resumed VM instances. The advantage of using a tree-based representation of files is that the caching ensures the transmission of only those files which have been changed. The ISR technique deals with situations where the migration of live machines is not a necessity. Predictably, the downtime (the period during which the service is unavailable due to there being no currently executing instance of a VM) is high, compared to some of the other techniques discussed later.

3.4.3.2 File System Migration

To support VM migration, a system must provide each VM with a consistent, location-independent view of the file system that is available on all hosts. A simple way to achieve this is to provide each VM with its own virtual disk which the file system is mapped to and transport the contents of this virtual disk along with the other states of the VM. However, due to the current trend of high capacity disks, migration of the contents of an entire disk over a network is not a viable solution. Another way is to have a global file system across all machines where a VM could be located. This way removes the need to copy files from one machine to another because all files are network accessible. A distributed file system is used in ISR serving as a transport mechanism for propagating a suspended VM state. The actual file systems themselves are not mapped onto the distributed file system. Instead, the VMM only accesses its local file system. The relevant VM files are explicitly copied into the local file system for a resume operation and taken out of the local file system for a suspend operation. This approach relieves developers from the complexities of implementing several different file system calls for different distributed file systems. It also essentially disassociates the VMM from any particular distributed file system semantics. However, this decoupling means that the VMM has to store the contents of each VM's virtual disks in its local files, which have to be moved around with the other state information of that VM.

In smart copying, the VMM exploits spatial locality. Typically, people often move between the same small number of locations, such as their home and office. In these conditions, it is possible to transmit only the difference between the two file systems at suspending and resuming locations. This technique significantly reduces the amount of actual physical data that has to be moved. In situations where there is no locality to exploit, a different approach is to synthesize much of the state at the resuming site. On many systems, user files only form a small fraction of the actual data on disk. Operating system and application software account for the majority of storage space. The proactive state transfer solution works in those cases where the resuming site can be predicted with reasonable confidence.

3.4.3.3 Network Migration

A migrating VM should maintain all open network connections without relying on forwarding mechanisms on the original host or on support from mobility or redirection mechanisms. To enable remote systems to locate and communicate with a VM, each VM must be assigned a virtual IP address known to other entities. This address can be distinct from the IP address of the host machine where the VM is currently located. Each VM can also have its own distinct virtual MAC address. The VMM maintains a mapping of the virtual IP and MAC addresses to their corresponding VMs. In general, a migrating VM includes all the protocol states and carries its IP address with it. If the source and destination machines of a VM migration are typically connected to a single switched LAN, an unsolicited ARP reply from the migrating host is provided advertising that the IP has moved to a new location. This solves the open network connection problem by reconfiguring all the peers to send future packets to a new location. Although a few packets that have already been transmitted might be lost, there are no other problems with this mechanism. Alternatively, on a switched network, the migrating OS can keep its original Ethernet MAC address and rely on the network switch to detect its move to a new port.

3.4.3.4 Live Migration of VM using Xen

Live migration means moving a VM from one physical node to another while keeping its OS environment and applications unbroken. This capability is being increasingly utilized in today's enterprise environments to provide efficient online system maintenance, reconfiguration, load balancing, and proactive fault toler-ance. It provides desirable features to satisfy requirements for computing resources in modern computing systems, including server consolidation, performance isolation, and ease of management. As a result, many implementations are available which support the feature using disparate functionalities. Traditional migra-tion suspends VMs before the transportation and then resumes them at the end of the process. By importing the precopy mechanism, a VM could be live migrated without stopping the VM and keep the applications running during the migration.

Live migration is a key feature of system virtualization technologies. Here, we focus on VM migration within a cluster environment where a network-accessible storage system, such as storage area network SAN or network attached storage (NAS), is employed. Only memory and CPU status needs to be transferred from the source node to the target node. Live migration techniques mainly use the precopy approach, which first transfers all memory pages, and then only copies modified pages during the last round iteratively. The VM service downtime is expected to be minimal by using iterative copy operations. In the precopy phase, although a VM service is still available, much performance degradation will occur because the migration daemon continually consumes network bandwidth to transfer dirty pages in each round. Moreover, the maximum number of iterations must be set because not all applications' dirty pages are ensured to converge to a small writable working set over multiple rounds. In fact, these issues with the pre-copy approach are caused by the large amount of transferred data during the whole migration process.

Another strategy of post-copy is introduced for live migration of VMs. Here, all memory pages are transferred only once during the whole migration process and the baseline total migration time is reduced. But the downtime is much higher than that of pre-copy due to the latency of fetching pages from the source node before the VM can be resumed on the target. With the advent of multi-core or many-core machines, abundant CPU resources are available. Even if several VMs reside on a same multi-core machine, CPU resources are still rich because physical CPUs are frequently amenable to multiplexing. We can exploit these copious CPU resources to compress page frames and the amount of transferred data can be significantly reduced. Memory compression algorithms typically have little memory overhead. Decompression is simple and very fast and requires no memory for decompression.

3.5 VIRTUALIZATION FOR DATA-CENTER AUTOMATION

Data-center automation means that huge volumes of hardware, software, and database resources in these data centers can be allocated dynamically to millions of Internet users simultaneously, with guaranteed QoS and cost-effectiveness. This automation process is triggered by the growth of virtualization products and cloud computing services. The latest virtualization development highlights high availability (HA), backup services, workload balancing, and further increases in client bases.

3.5.1 Server Consolidation in Data Centers

In data centers, a large number of heterogeneous workloads can run on servers at various times. These heterogeneous workloads can be roughly divided into two categories: chatty workloads and non-interactive workloads. Chatty workloads may burst at some point and return to a silent state at some other point. A web video service is an example of this, whereby a lot of people use it at night and few people use it during the day. Non-interactive workloads do not require people's efforts to make progress after they are submitted. High-performance computing is a typical example of this. At various stages, the requirements for resources of these workloads are dramatically different. However, to guarantee that a workload will always be able to cope with all demand levels, the workload is statically allocated enough resources so that peak demand is satisfied.

Therefore, it is common that most servers in data centers are underutilized. A large amount of hardware, space, power, and management cost of these servers is wasted. Server consolidation is an approach to improve the low utility ratio of hardware resources by reducing the number of physical servers. Among several server consolidation techniques such as centralized and physical consolidation, virtualization-based server consolidation is the most powerful. Data centers need to optimize their resource management. Yet these techniques are performed with the granularity of a full server machine, which makes resource management far from well optimized. Server virtualization enables smaller resource allocation than a physical machine. In general, the use of VMs increases resource management complexity. This causes a challenge in terms of how to improve resource utilization as well as guarantee QoS in data centers. In detail, server virtualization has the following side effects:

- Consolidation enhances hardware utilization. Many underutilized servers are consolidated into fewer servers to enhance resource utilization. Consolidation also facilitates backup services and disaster recovery.

- This approach enables more agile provisioning and deployment of resources. In a virtual environment, the images of the guest OSes and their applications are readily cloned and reused.

- The total cost of ownership is reduced. In this sense, server virtualization causes deferred purchases of new servers, a smaller data-center footprint, lower maintenance costs, and lower power, cooling, and cabling requirements.

- This approach improves availability and business continuity. The crash of a guest OS has no effect on the host OS or any other guest OS. It becomes easier to transfer a VM from one server to another, because virtual servers are unaware of the underlying hardware.

To automate data-center operations, one must consider resource scheduling, architectural support, power management, automatic or autonomic resource management, performance of analytical models, and so on. In virtualized data centers, an efficient, on-demand, fine-grained scheduler is one of the key factors to improve resource utilization. Scheduling and reallocations can be done in a wide range of levels in a set of data centers. The levels match at least at the VM level, server level, and data-center level. Ideally, scheduling and resource reallocations should be done at all levels. However, due to the complexity of this, current techniques only focus on a single level or, at most, two levels.

3.5.2 Virtual Storage Management

The term “storage virtualization” was widely used before the rebirth of system virtualization. Yet the term has a different meaning in a system virtualization environment. Previously, storage virtualization was largely used to describe the aggregation and repartitioning of disks at very coarse time scales for use by physical machines. In system virtualization, virtual storage includes the storage managed by VMMs and guest OSes. Generally, the data stored in this environment can be classified into two categories: VM images and application data. The VM images are special to the virtual environment, while application data includes all other data which is the same as the data in traditional OS environments.

The most important aspects of system virtualization are encapsulation and isolation. Traditional operating systems and applications running on them can be encapsulated in VMs. Only one operating system runs in a virtualization while many applications run in the operating system. System virtualization allows multiple VMs to run on a physical machine and the VMs are completely isolated. To achieve encapsulation and isolation, both the system software and the hardware platform, such as CPUs and chipsets, are rapidly updated. However, storage is lagging. The storage systems become the main bottleneck of VM deployment.

In virtualization environments, a virtualization layer is inserted between the hardware and traditional operating systems or a traditional operating system is modified to support virtualization. This procedure complicates storage operations. On the one hand, storage management of the guest OS performs as though it is operating in real hard disks while the guest OS’s cannot access the hard disk directly. On the other hand, many guest OSes contest the hard disk when many VMs are running on a single physical machine. Therefore, storage management of the underlying VMM is much more complex than that of guest OSes (traditional OSes).

3.5.3 Cloud OS for Virtualized Data Centers

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

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

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

These VI managers are used to create VMs and aggregate them into virtual clusters as elastic resources. Nimbus and Eucalyptus support essentially virtual networks. OpenNebula has additional features to provision dynamic resources and make advance reservations. All three public VI managers apply Xen and KVM for virtualization. vSphere 4 uses the hypervisors ESX and ESXi from VMware. Only vSphere 4 supports virtual storage in addition to virtual networking and data protection.

3.5.4 Trust Management in Virtualized Data Centers

A VMM changes the computer architecture. It provides a layer of software between the operating systems and system hardware to create one or more VMs on a single physical platform. A VM entirely encapsulates the state of the guest operating system running inside it. Encapsulated machine state can be copied and shared over the network and removed like a normal file, which proposes a challenge to VM security. In general, a VMM can provide secure isolation and a VM accesses hardware resources through the control of the VMM, so the VMM is the base of the security of a virtual system. Normally, one VM is taken as a management VM to have some privileges such as creating, suspending, resuming, or deleting a VM. Once a hacker successfully enters the VMM or management VM, the whole system is in danger. A subtler problem arises in protocols that rely on the "freshness" of their random number source for generating session keys. Considering a VM, rolling back to a point after a random number has been chosen, but before it has been used, resumes execution; the random number, which must be "fresh" for security purposes, is reused. With a stream cipher, two different plaintexts could be encrypted under the same key stream, which could, in turn, expose both plaintexts if the plaintexts have sufficient redundancy. Non-cryptographic protocols that rely on freshness are also at risk. For example, the reuse of TCP initial sequence numbers can raise TCP hijacking attacks.

3.5.4.1 VM-Based Intrusion Detection

Intrusions are unauthorized access to a certain computer from local or network users and intrusion detection is used to recognize the unauthorized access. An intrusion detection system (IDS) is built on operating systems, and is based on the characteristics of intrusion actions. A typical IDS can be classified as a host-based IDS (HIDS) or a network-based IDS (NIDS), depending on the data source. A HIDS can be implemented on the monitored system. When the monitored system is attacked by hackers, the HIDS also faces the risk of being attacked. A NIDS is based on the flow of network traffic which can't detect fake actions.

Virtualization-based intrusion detection can isolate guest VMs on the same hardware platform. Even some VMs can be invaded successfully; they never influence other VMs, which is similar to the way in which a NIDS operates. Furthermore, a VMM monitors and audits access requests for hardware and system software. This can avoid fake actions and possess the merit of a HIDS. There are two different methods for implementing a VM-based IDS: Either the IDS is an independent process in each VM or a high-privileged VM on the VMM; or the IDS is integrated into the VMM and has the same privilege to access the hardware as well as the VMM.

The VM-based IDS contains a policy engine and a policy module. The policy framework can monitor events in different guest VMs by operating system interface library and PTrace indicates trace to secure policy of monitored host. It's difficult to predict and prevent all intrusions without delay. Therefore, an analysis of the intrusion action is extremely important after an intrusion occurs. At the time of this writing, most computer systems use logs to analyze attack actions, but it is hard to ensure the credibility and integrity of a log. The IDS log service is based on the operating system kernel. Thus, when an operating system is invaded by attackers, the log service should be unaffected.

Besides IDS, honeypots and honeynets are also prevalent in intrusion detection. They attract and provide a fake system view to attackers in order to protect the real system. In addition, the attack action can be analyzed, and secure IDS can be built. A honeypot is a purposely defective system that simulates an operating system to cheat and monitor the actions of an attacker. A honeypot can be divided into physical and virtual forms. A guest operating system and the applications running on it constitute a VM. The host operating system and VMM must be guaranteed to prevent attacks from the VM in a virtual honeypot.