CHAPTER

Virtual Machines and Virtualization of Clusters and Data Centers

3

CHAPTER OUTLINE

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. For example, MIPS binary code can run on an x86-based host machine with the help of ISA emulation. With this approach, it is possible to run a large amount of legacy binary code written for various processors on any given new hardware host machine. Instruction set emulation leads to virtual ISAs created on any hardware machine.

The basic emulation method is through *code interpretation*. An interpreter program interprets the source instructions to target instructions one by one. 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. The idea was implemented in the IBM VM/370 in the 1960s. More recently, the Xen hypervisor has been applied to virtualize x86-based machines to run Linux or other guest OS applications. We will discuss hardware virtualization approaches in more detail in Section 3.3.

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. It is also used, to a lesser extent, in consolidating server hardware by moving services on separate hosts into containers or VMs on one server. OS-level virtualization is depicted in Section 3.1.3.

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. The software tool WINE has implemented this approach to support Windows applications on top of UNIX hosts. Another example is the vCUDA which allows applications executing within VMs to leverage GPU hardware acceleration. This approach is detailed in Section 3.1.4.

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

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. The former arises when more than one VM is running on the same machine.

The hardware resource requirements, such as memory, of each VM are reduced, but the sum of them is greater than that of the real machine installed. The latter qualification is required because of the intervening level of software and the effect of any other VMs concurrently existing on the same hardware. Obviously, these two differences pertain to performance, while the function a VMM provides stays the same as that of a real machine. However, the identical environment requirement excludes the behavior of the usual time-sharing operating system from being classed as a VMM.

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 intervention by the VMM. Table 3.2 compares four hypervisors and VMMs that are in use today.

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. Not all processors satisfy these requirements for a VMM. A VMM is tightly related to the architectures of processors. It is difficult to

Table 3.2 Comparison of Four VMM and Hypervisor Software Packages

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

implement a VMM for some types of processors, such as the x86. Specific limitations include the inability to trap on some privileged instructions. 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. For example, a task may need only a single CPU during some phases of execution but may need hundreds of CPUs at other times. 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?

As mentioned earlier, it is slow to initialize a hardware-level VM because each VM creates its own image from scratch. 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. Figure 3.3 illustrates operating system virtualization from the point of view of a machine stack.

3.1.3.2 Advantages of OS Extensions

Compared to hardware-level virtualization, the benefits of OS extensions are twofold: (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. In cloud

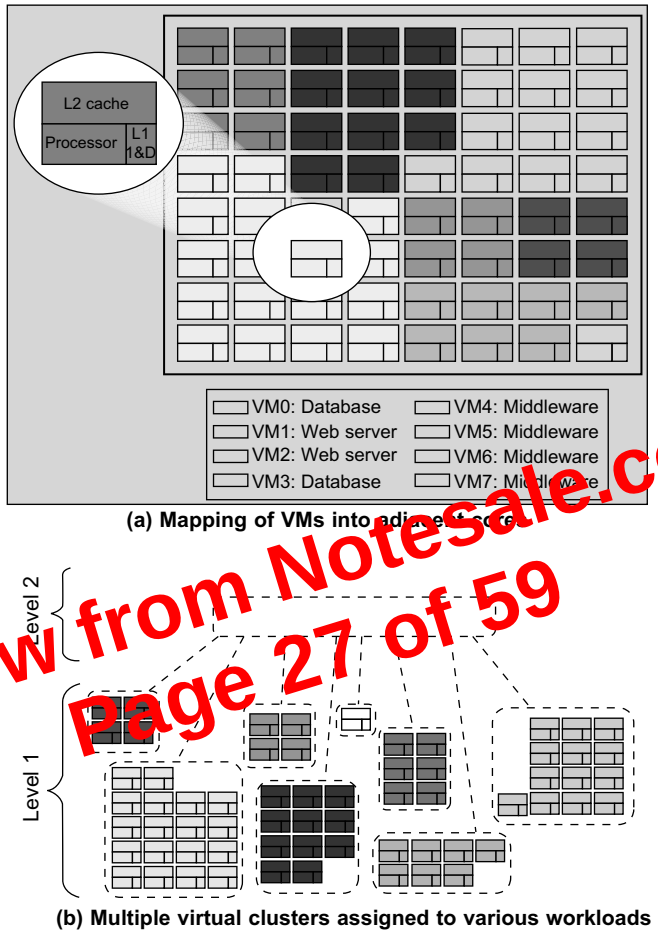

FIGURE 3.17

CMP server consolidation by space-sharing of VMs into many cores forming multiple virtual clusters to execute various workloads.

(*Courtesy of Marty and Hill [39]*)

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. In Chapter 2, we studied various clustering techniques on physical machines. Here, we introduce virtual clusters and study its properties as well as explore their potential applications. In this section, we will study three critical design issues of virtual clusters: *live migration* of VMs, *memory and file migrations*, and *dynamic deployment* of virtual clusters.

migration of VIOLIN environments does pay off. Of course, the gain in shared resource utilization will benefit many users, and the performance gain varies with different adaptation scenarios. We leave readers to trace the execution of another scenario in Problem 3.17 at the end of this chapter to tell the differences. Virtual networking is a fundamental component of the VIOLIN system.

3.5 VIRTUALIZATION FOR DATA-CENTER AUTOMATION

Data centers have grown rapidly in recent years, and all major IT companies are pouring their resources into building new data centers. In addition, Google, Yahoo!, Amazon, Microsoft, HP, Apple, and IBM are all in the game. All these companies have invested billions of dollars in data-center construction and automation. 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. From 2006 to 2011, according to an IDC 2007 report on the growth of virtualization and its market distribution in major IT sectors, in 2006, virtualization has a market share of $1,044 million in business and enterprise opportunities. The majority was dominated by production consolidation and software development. Virtualization is moving towards enhancing mobility, reducing planned downtime (for maintenance), and increasing the number of virtual clients.

The latest virtualization development highlights *high availability* (HA), backup services, workload balancing, and further increases in client bases. IDC projected that automation, service orientation, policy-based, and variable costs in the virtualization market. The total business opportunities may increase to $3.2 billion by 2011. The major market share moves to the areas of HA, utility computing, production consolidation, and client bases. In what follows, we will discuss server consolidation, virtual storage, OS support, and trust management in automated data-center designs.

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 noninteractive 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. Noninteractive 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. Figure 3.29 illustrates server virtualization in a data center. In this case, the granularity of resource optimization is focused on the CPU, memory, and network interfaces.

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 [63]	CLI, GUI, Portal, WS	VMware ESX, ESXi	VMware vCloud partners	Data protection, vStorage, VMFS, DRM, HA

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.

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. We will study Eucalyptus and vSphere 4 in the next two examples.

Example 3.12 Eucalyptus for Virtual Networking of Private Cloud

Eucalyptus is an open source software system (Figure 3.27) intended mainly for supporting Infrastructure as a Service (IaaS) clouds. The system primarily supports virtual networking and the management of VMs; virtual storage is not supported. Its purpose is to build private clouds that can interact with end users through Ethernet or the Internet. The system also supports interaction with other private clouds or public clouds over the Internet. The system is short on security and other desired features for general-purpose grid or cloud applications.

The designers of Eucalyptus [45] implemented each high-level system component as a stand-alone web service. Each web service exposes a well-defined language-agnostic API in the form of a WSDL document containing both operations that the service can perform and input/output data structures.

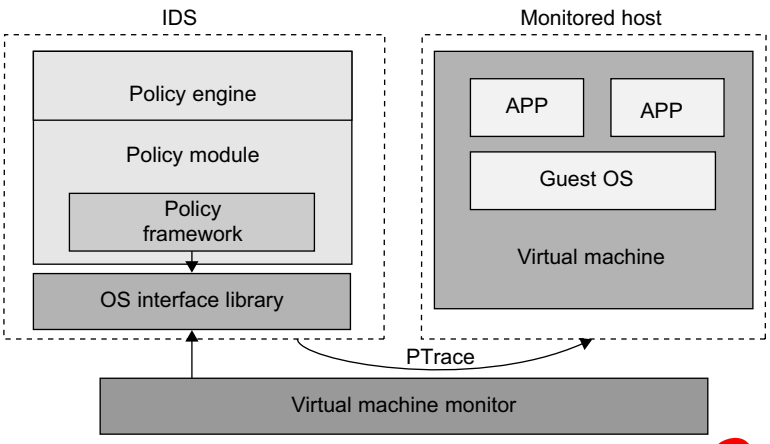

FIGURE 3.29

The architecture of livewire for intrusion detection using a dedicated VM

(courtesy of Garfinkel and Rosenblum, 2002 [17])

and has the same privilege to access the hardware as well as the VMM. Garfinkel and Rosenblum [17] have proposed an IDS to run on a VMM as a high-privileged VM. Figure 3.29 illustrates the concept.

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 a 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.

Example 3.14 EMC Establishment of Trusted Zones for Protection of Virtual Clusters Provided to Multiple Tenants

EMC and VMware have joined forces in building security middleware for trust management in distributed systems and private clouds. The concept of *trusted zones* was established as part of the virtual infrastructure. Figure 3.30 illustrates the concept of creating trusted zones for virtual clusters (multiple

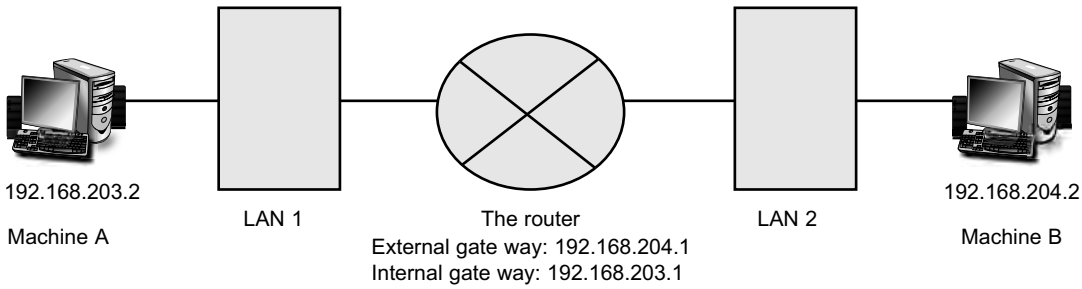

FIGURE 3.31

The topological structure of the virtual LAN.

Problem 3.10

Study the relevant papers [33,63,74] on asymmetric or heterogeneous chip multitprocessors (CMP). Write a study report to survey the area, identify the key research issues, review the current development and open research challenges lying ahead.

Problem 3.11

Study the relevant papers [17,28,30,66] on network on chip (NoC) and virtualization of NoC resources in multi-core CMP designs and applications. Repeat Problem 3.10 with a survey report after the research study.

Problem 3.12

Hardware and software resource deployment are often complicated and time-consuming. Automatic VM deployment can significantly reduce the time to instantiate new services or reallocate resources depending on user needs. Visit the following web site for more information. http://wiki.systemimager.org/index.php/Automating_Xen_VM_deployment_with_SystemImager. Report your experience with automatic deployment using the SystemImager and Xen-tools.

Problem 3.13

Design an experiment to analyze the performance of Xen live migration for I/O read-intensive applications. The performance merits include the time consumed by the precopy phase, the downtime, the time used by the pull phase, and the total migration time.

Problem 3.14

Design an experiment to test the performance of Xen live migration for I/O write-intensive applications. The performance metrics include the time consumed by the precopy phase, the downtime, the time used by the pull phase, and the total migration time. Compare the results with those from Problem 3.13.