Cloud Computing

Unit – 1

UNIT I: Systems modeling, Clustering and virtualization:
Scalable Computing over the Internet, Technologies for Network based systems, System models for Distributed and Cloud Computing, Software environments for distributed systems and clouds, Performance, Security And Energy Efficiency.

SCALABLE COMPUTING OVER THE INTERNET

Over the past 60 years, computing technology has undergone a series of platform and environment changes. In this section, we assess evolutionary changes in machine architecture, operating system platform, network connectivity, and application workload. Instead of using a centralized computer to solve computational problems, a parallel and distributed computing system uses multiple computers to solve large-scale problems over the Internet. Thus, distributed computing becomes data-intensive and network-centric.

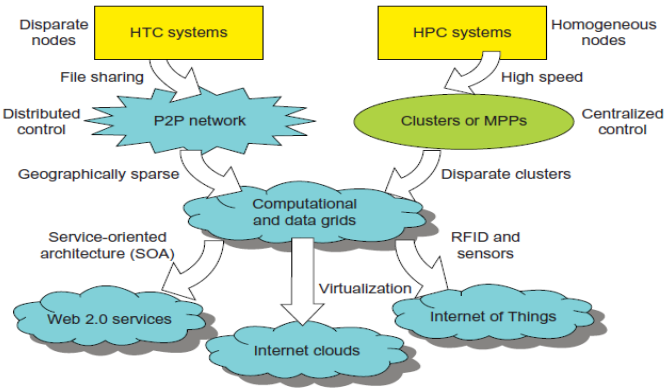
The Age of Internet Computing

Billions of people use the Internet every day. As a result, supercomputer sites and large data centers must provide high-performance computing services to huge numbers of Internet users concurrently. Because of this high demand, the Linpack Benchmark for high-performance computing (HPC) applications is no longer optimal for measuring system performance.

The emergence of computing clouds instead demands high-throughput computing (HTC) systems built with parallel and distributed computing technologies

High-Performance Computing

For many years, HPC systems emphasize the raw speed performance. The speed of HPC systems has increased from Gflops in the early 1990s to now Pflops in 2010.
This improvement was driven mainly by the demands from scientific, engineering, and manufacturing communities. For example, the Top 500 most powerful computer systems in the world are measured by floating-point speed in Linpack benchmark results. However, the number of supercomputer users is limited to less than 10% of all computer users. Today, the majority of computer users are using desktop computers or large servers when they conduct Internet searches and market-driven computing tasks.



High-Throughput Computing

The development of market-oriented high-end computing systems is undergoing a strategic change from an HPC paradigm to an HTC paradigm.

This HTC paradigm pays more attention to high-flux computing. The main application for high-flux computing is in Internet searches and web services by millions or more users simultaneously.

The performance goal thus shifts to measure high throughput or the number of tasks completed per unit of time. HTC technology needs to not only improve in terms of batch processing speed, but also address the acute problems of cost, energy savings, security, and reliability at many data and enterprise computing centers. This book will address both HPC and HTC systems to meet the demands of all computer users.

Three New Computing Paradigms

With the introduction of SOA, Web 2.0 services become available. Advances in virtualization make it possible to see the growth of Internet clouds as a new computing paradigm. The maturity of radio-frequency identification (RFID), Global Positioning System (GPS), and sensor technologies has triggered the development of the Internet of Things (IoT).

When the Internet was introduced in 1969, Leonard Klienrock of UCLA declared: "As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of computer utilities, which like present electric and telephone utilities, will service individual homes and offices across the country." Many people have redefined the term "computer" since that time. In 1984, John Gage of Sun Microsystems created the slogan, "The network is the computer."

In 2008, David Patterson of UC Berkeley said, "The data center is the computer. There are dramatic differences between developing software for millions to use as a service versus distributing software to run on their PCs." Recently, Rajkumar Buyya of Melbourne University simply said: "The cloud is the computer."

Computing Paradigm Distinctions

The high-technology community has argued for many years about the precise definitions of centralized computing, parallel computing, distributed computing, and cloud computing. In general, distributed computing is the opposite of centralized computing.

The field of parallel computing overlaps with distributed computing to a great extent, and cloud computing overlaps with distributed, centralized, and parallel computing.

• Centralized computing This is a computing paradigm by which all computer resources are centralized in one physical system. All resources (processors, memory, and storage) are fully shared and tightly coupled within one integrated OS. Many data centers and supercomputers are centralized systems, but they are used in parallel, distributed, and cloud computing applications.
• Parallel computing In parallel computing, all processors are either tightly coupled with centralized shared memory or loosely coupled with distributed memory. Some authors refer to this discipline as parallel processing. Interprocessor communication is accomplished through shared memory or via message passing.

A computer system capable of parallel computing is commonly known as a parallel computer. Programs running in a parallel computer are called parallel programs. The process of writing parallel programs is often referred to as parallel programming.

• Distributed computing This is a field of computer science/engineering that studies distributed systems. A distributed system consists of multiple autonomous computers, each having its own private memory, communicating through a computer network.

Information exchange in a distributed system is accomplished through message passing. A computer program that runs in a distributed system is known as a distributed program. The process of writing distributed programs is referred to as distributed programming.

• Cloud computing An Internet cloud of resources can be either a centralized or a distributed computing system. The cloud applies parallel or distributed computing, or both. Clouds can be built with physical or virtualized resources over large data centers that are centralized or distributed. Some authors consider cloud computing to be a form of utility computing or service computing.

Distributed System Families

Since the mid-1990s, technologies for building P2P networks and networks of clusters have been consolidated into many national projects designed to establish wide area computing infrastructures, known as computational grids or data grids.

Meeting these goals requires to yield the following design objectives:

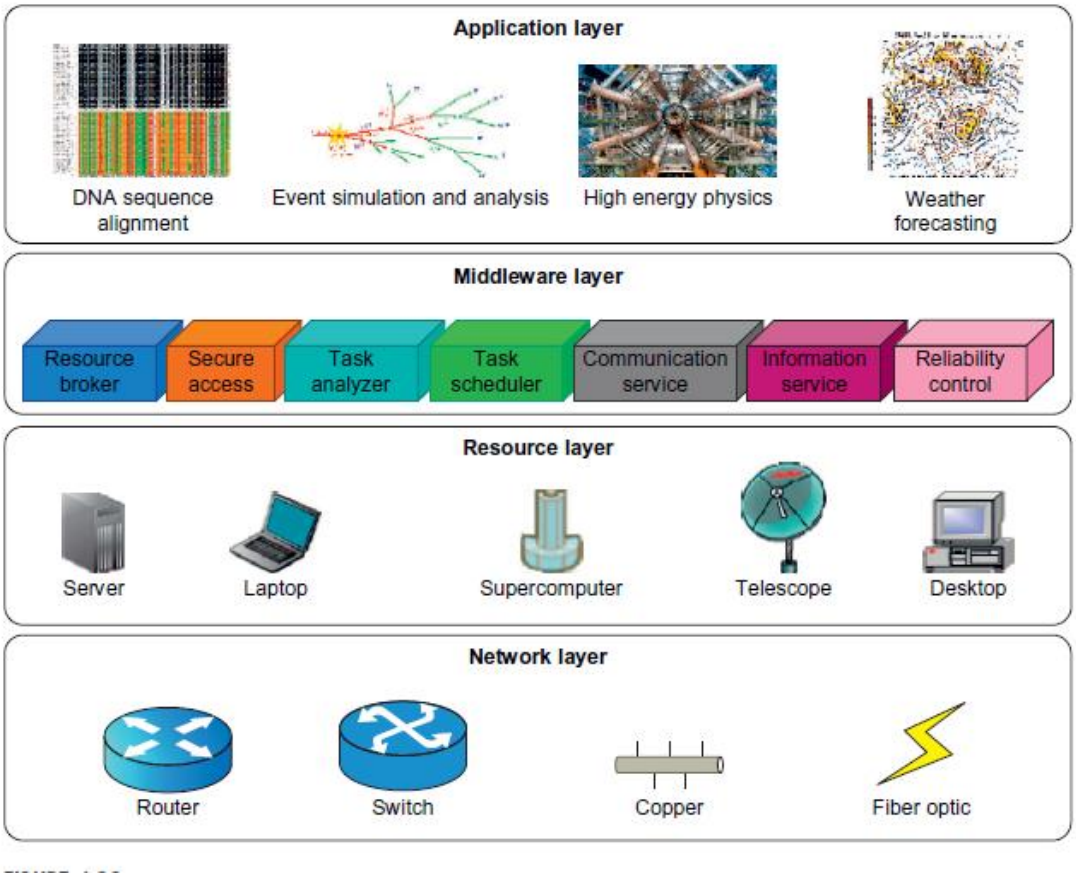
• Efficiency measures the utilization rate of resources in an execution model by exploiting massive parallelism in HPC. For HTC, efficiency is more closely related to job throughput, data access, storage, and power efficiency.
• Dependability measures the reliability and self-management from the chip to the system and application levels. The purpose is to provide high-throughput service with Quality of Service (QoS) assurance, even under failure conditions.
• Adaptation in the programming model measures the ability to support billions of job requests over massive data sets and virtualized cloud resources under various workload and service models.
• Flexibility in application deployment measures the ability of distributed systems to run well in both HPC (science and engineering) and HTC (business) applications.

Degrees of Parallelism

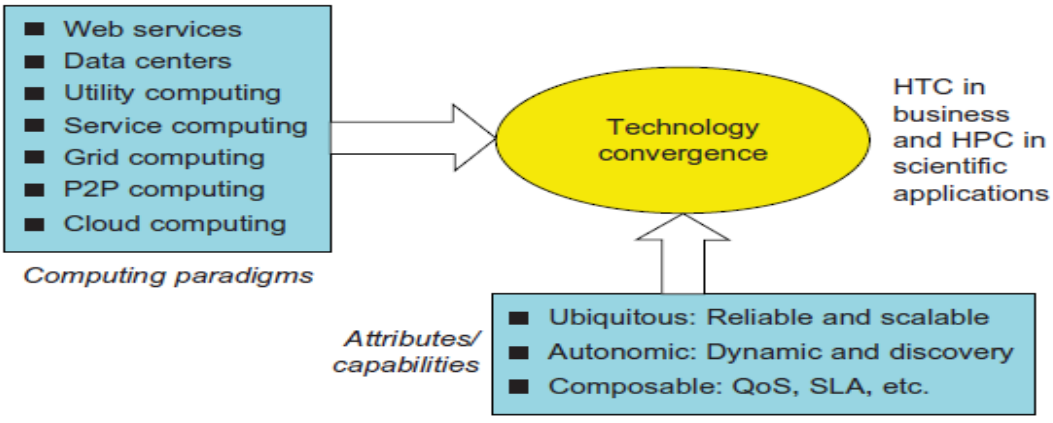
Fifty years ago, when hardware was bulky and expensive, most computers were designed in a bit-serial fashion. In this scenario, bit-level parallelism (BLP) converts bit-serial processing to word-level processing gradually. Over the years, users graduated from 4-bit microprocessors to 8-,16-, 32-, and 64-bit CPUs. This led us to the next wave of improvement, known as instruction-level parallelism (ILP), in which the processor executes multiple instructions simultaneously rather than only one instruction at a time.

For the past 30 years, we have practiced ILP through pipelining, superscalar computing, VLIW (very long instruction word) architectures, and multithreading. ILP requires branch prediction, dynamic scheduling, speculation, and compiler support to work efficiently. Data-level parallelism (DLP) was made popular through SIMD (single instruction, multiple data) and vector machines using vector or array types of instructions. DLP requires even more hardware support and compiler assistance to work properly.

Ever since the introduction of multicore processors and chip multiprocessors (CMPs), we have been exploring task-level parallelism (TLP). A modern processor explores all of the aforementioned parallelism types. In fact, BLP, ILP, and DLP are well supported by advances in hardware and compilers. However, TLP is far from being very successful due to difficulty in programming and compilation of code for efficient execution on multicore CMPs.



Utility computing focuses on a business model in which customers receive computing resources from a paid service provider. All grid/cloud platforms are regarded as utility service providers. However, cloud computing offers a broader concept than utility computing. Distributed cloud applications run on any available servers in some edge networks. Major technological challenges include all aspects of computer science and engineering.



The Internet of Things

The concept of the IoT was introduced in 1999 at MIT [40]. The IoT refers to the networked interconnection of everyday objects, tools, devices, or computers. One can view the IoT as a wireless network of sensors that interconnect all things in our daily life. These things can be large or small and they vary with respect to time and place. The idea is to tag every object using RFID or a related sensor or electronic technology such as GPS.

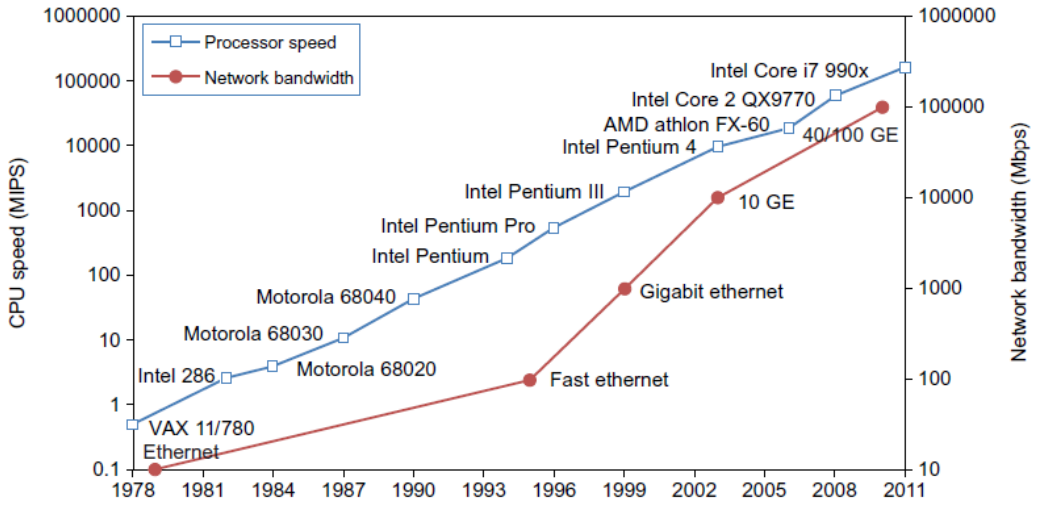
TECHNOLOGIES FOR NETWORK-BASED SYSTEMS

Multicore CPUs and Multithreading Technologies

Consider the growth of component and network technologies over the past 30 years. They are crucial to the development of HPC and HTC systems. In Figure 1.4, processor speed is measured in millions of instructions per second (MIPS) and network bandwidth is measured in megabits per second (Mbps) or gigabits per second (Gbps). The unit GE refers to 1 Gbps Ethernet bandwidth.

Advances in CPU Processors

Advanced CPUs or microprocessor chips assume a multicore architecture with dual, quad, six, or more processing cores. These processors exploit parallelism at ILP and TLP levels. Processor speed growth is plotted in the upper curve in across generations of microprocessors or CMPs.



Multicore CPU and Many-Core GPU Architectures

Multicore CPUs may increase from the tens of cores to hundreds or more in the future. But the CPU has reached its limit in terms of exploiting massive DLP due to the aforementioned memory wall problem. This has triggered the development of many-core GPUs with hundreds or more thin cores. Both IA-32 and IA-64 instruction set architectures are built into commercial CPUs. Now, x-86 processors have been extended to serve HPC and HTC systems in some high-end server processors.

Multithreading Technology & How GPUs Work

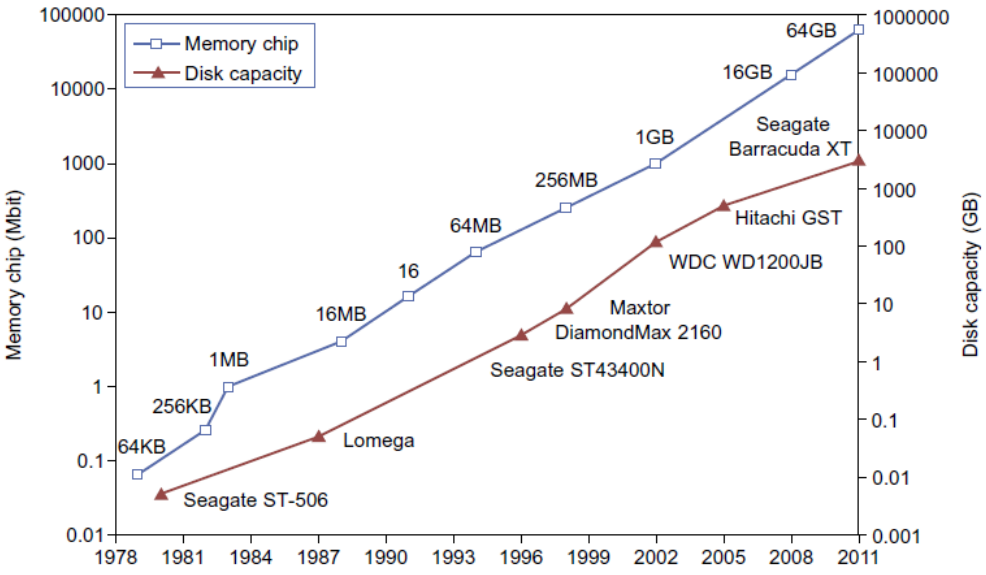
Memory, Storage, and Wide-Area Networking

Memory Technology

The growth of DRAM chip capacity from 16 KB in 1976 to 64 GB in 2011. This shows that memory chips have experienced a 4x increase in capacity every three years. Memory access time did not improve much in the past. In fact, the memory wall problem is getting worse as the processor gets faster. For hard drives, capacity increased from 260 MB in 1981 to 250 GB in 2004. The Seagate Barracuda XT hard drive reached 3 TB in 2011.

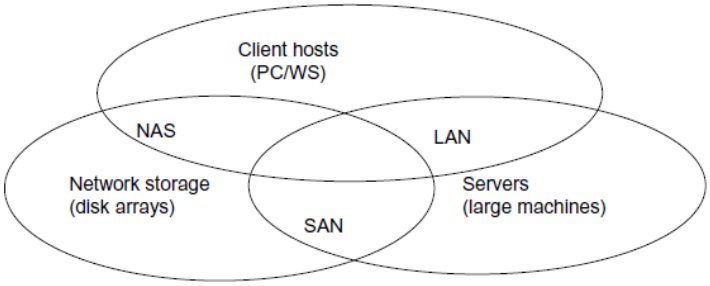
Disks and Storage Technology

Beyond 2011, disks or disk arrays have exceeded 3 TB in capacity. The lower curve in Figure shows the disk storage growth in 7 orders of magnitude in 33 years. The rapid growth of flash memory and solid-state drives (SSDs) also impacts the future of HPC and HTC systems. The mortality rate of SSD is not bad at all. A typical SSD can handle 300,000 to 1 million write cycles per block. So the SSD can last for several years, even under conditions of heavy write usage. Flash and SSD will demonstrate impressive speedups in many applications.



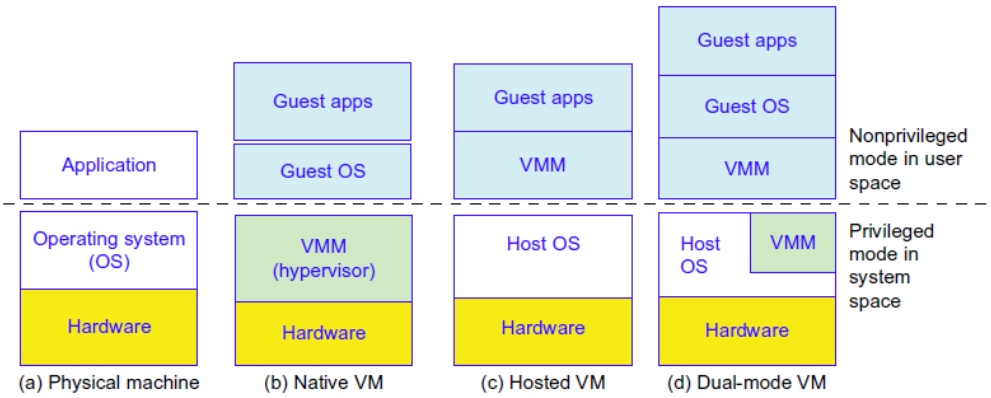
System-Area Interconnects

The nodes in small clusters are mostly interconnected by an Ethernet switch or a local area network (LAN). As Figure 1.11 shows, a LAN typically is used to connect client hosts to big servers. A storage area network (SAN) connects servers to network storage such as disk arrays. Network attached storage (NAS) connects client hosts directly to the disk arrays. All three types of networks often appear in a large cluster built with commercial network components.



Virtual Machines and Virtualization Middleware

A conventional computer has a single OS image. This offers a rigid architecture that tightly couples application software to a specific hardware platform. Some software running well on one machine may not be executable on another platform with a different instruction set under a fixed OS. Virtual machines (VMs) offer novel solutions to underutilized resources, application inflexibility, software manageability, and security concerns in existing physical machines.



Virtual Machines

The host machine is equipped with the physical hardware, as shown at the bottom of the figure. An example is an x-86 architecture desktop running its installed Windows OS, as shown in part (a) of the figure.

The VM can be provisioned for any hardware system. The VM is built with virtual resources managed by a guest OS to run a specific application. Between the VMs and the host platform, one needs to deploy a middleware layer called a virtual machine monitor (VMM). Figure 1.12(b) shows a native VM installed with the use of a VMM called a hypervisor in privileged mode. For example, the hardware has x-86 architecture running the Windows system.

The guest OS could be a Linux system and the hypervisor is the XEN system developed at Cambridge University. This hypervisor approach is also called bare-metal VM, because the hypervisor handles the bare hardware (CPU, memory, and I/O) directly. Another architecture is the host VM shown in Figure 1.12(c). Here the VMM runs in nonprivileged mode. The host OS need not be modified. The VM can also be implemented with a dual mode, as shown in Figure 1.12(d).

Part of the VMM runs at the user level and another part runs at the supervisor level. In this case, the host OS may have to be modified to some extent. Multiple VMs can be ported to a given hardware system to support the virtualization process. The VM approach offers hardware independence of the OS and applications. The user application running on its dedicated OS could be bundled together as a virtual

appliance that can be ported to any hardware platform. The VM could run on an OS different from that of the host computer.

VM Primitive Operations

The VMM provides the VM abstraction to the guest OS. With full virtualization, the VMM exports a VM abstraction identical to the physical machine so that a standard OS such as Windows 2000 or Linux can run just as it would on the physical hardware. Low-level VMM operations are indicated by Mendel Rosenblum [41] and illustrated in Figure 1.13.

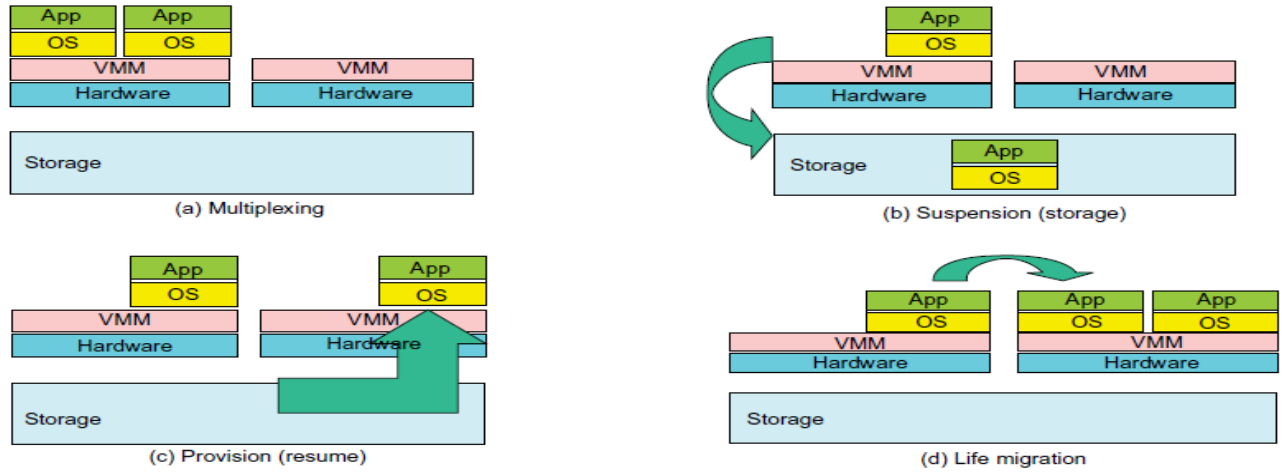


FIGURE 1.13
VM multiplexing, suspension, provision, and migration in a distributed computing environment.

First, the VMs can be multiplexed between hardware machines, as shown in Figure (a).
Second, a VM can be suspended and stored in stable storage, as shown in Figure (b).
Third, a suspended VM can be resumed or provisioned to a new hardware platform, as shown in Figure (c).
Finally, a VM can be migrated from one hardware platform to another, as shown in Figure (d).

Virtual Infrastructures

Physical resources for compute, storage, and networking at the bottom of Figure are mapped to the needy applications embedded in various VMs at the top. Hardware and software are then separated. Virtual infrastructure is what connects resources to distributed applications. It is a dynamic mapping of system resources to specific applications. The result is decreased costs and increased efficiency and responsiveness. Virtualization for server consolidation and containment is a good example of this.

Data Center Virtualization for Cloud Computing

Data Center Growth and Cost Breakdown

A large data center may be built with thousands of servers. Smaller data centers are typically built with hundreds of servers. The cost to build and maintain data center servers has increased over the years. According to a 2009 IDC report, typically only 30 percent of data center costs goes toward purchasing IT equipment (such as servers and disks), 33 percent is attributed to the chiller, 18 percent to the uninterruptible power supply (UPS), 9 percent to computer room air conditioning (CRAC), and the remaining 7 percent to power distribution, lighting, and transformer costs. Thus, about 60 percent of the cost to run a data center is allocated to management and maintenance. The server purchase cost did not

increase much with time. The cost of electricity and cooling did increase from 5 percent to 14 percent in 15 years.

SYSTEM MODELS FOR DISTRIBUTED AND CLOUD COMPUTING

Distributed and cloud computing systems are built over a large number of autonomous computer nodes. These node machines are interconnected by SANs, LANs, or WANs in a hierarchical manner. With today's networking technology, a few LAN switches can easily connect hundreds of machines as a working cluster. A WAN can connect many local clusters to form a very large cluster of clusters. In this sense, one can build a massive system with millions of computers connected to edge networks.

Massive systems are considered highly scalable, and can reach web-scale connectivity, either physically or logically. In Table 1.2, massive systems are classified into four groups: clusters, P2P networks, computing grids, and Internet clouds over huge data centers. In terms of node number, these four system classes may involve hundreds, thousands, or even millions of computers as participating nodes.

Table 1.2 Classification of Parallel and Distributed Computing Systems

Functionality, Applications	Computer Clusters [10,28,38]	Peer-to-Peer Networks [34,46]	Data/ Computational Grids [6,18,51]	Cloud Platforms [1,9,11,12,30]
Architecture, Network Connectivity, and Size	Network of compute nodes interconnected by SAN, LAN, or WAN hierarchically	Flexible network of client machines logically connected by an overlay network	Heterogeneous clusters interconnected by high-speed network links over selected resource sites	Virtualized cluster of servers over data centers via SLA
Control and Resources Management	Homogeneous nodes with distributed control, running UNIX or Linux	Autonomous client nodes, free in and out, with self-organization	Centralized control, server-oriented with authenticated security	Dynamic resource provisioning of servers, storage, and networks
Applications and Network-centric Services	High-performance computing, search engines, and web services, etc.	Most appealing to business file sharing, content delivery, and social networking	Distributed supercomputing, global problem solving, and data center services	Upgraded web search, utility computing, and outsourced computing services
Representative Operational Systems	Google search engine, SunBlade, IBM Road Runner, Cray XT4, etc.	Gnutella, eMule, BitTorrent, Napster, KaZaA, Skype, JXTA	TeraGrid, GriPhyN, UK EGEE, D-Grid, ChinaGrid, etc.	Google App Engine, IBM Bluecloud, AWS, and Microsoft Azure

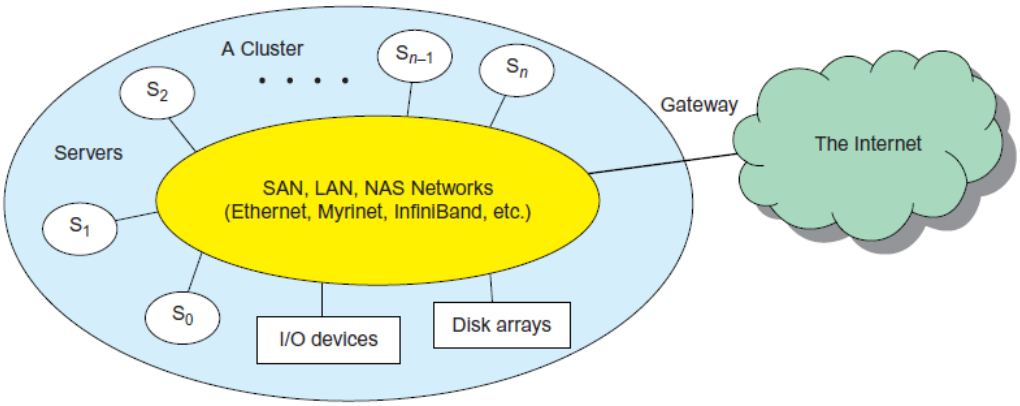
Clusters of Cooperative Computers

A computing cluster consists of interconnected stand-alone computers which work cooperatively as a single integrated computing resource. In the past, clustered computer systems have demonstrated impressive results in handling heavy workloads with large data sets.

Cluster Architecture

Figure 1.15 shows the architecture of a typical server cluster built around a low-latency, high bandwidth interconnection network. This network can be as simple as a SAN (e.g., Myrinet) or a LAN (e.g., Ethernet). To build a larger cluster with more nodes, the interconnection network can be built with multiple levels of Gigabit Ethernet, Myrinet, or InfiniBand switches. Through hierarchical construction using a SAN, LAN, or WAN, one can build scalable clusters with an increasing number of nodes. The cluster is connected to

the Internet via a virtual private network (VPN) gateway. The gateway IP address locates the cluster. The system image of a computer is decided by the way the OS manages the shared cluster resources. Most clusters have loosely coupled node computers. All resources of a server node are managed by their own OS. Thus, most clusters have multiple system images as a result of having many autonomous nodes under different OS control.



Single-System Image

Greg Pfister [38] has indicated that an ideal cluster should merge multiple system images into a single-system image (SSI). Cluster designers desire a cluster operating system or some middleware to support SSI at various levels, including the sharing of CPUs, memory, and I/O across all cluster nodes.

Hardware, Software, and Middleware Support

Clusters exploring massive parallelism are commonly known as MPPs. Almost all HPC clusters in the Top 500 list are also MPPs. The building blocks are computer nodes (PCs, workstations, servers, or SMP), special communication software such as PVM or MPI, and a network interface card in each computer node. Most clusters run under the Linux OS. The computer nodes are interconnected by a high-bandwidth network (such as Gigabit Ethernet, Myrinet, InfiniBand, etc.). Special cluster middleware supports are needed to create SSI or high availability (HA). Both sequential and parallel applications can run on the cluster, and special parallel environments are needed to facilitate use of the cluster resources.

Grid Computing Infrastructures

Internet services such as the Telnet command enables a local computer to connect to a remote computer. A web service such as HTTP enables remote access of remote web pages. Grid computing is envisioned to allow close interaction among applications running on distant computers simultaneously.

Computational Grids

Like an electric utility power grid, a computing grid offers an infrastructure that couples computers, software/middleware, special instruments, and people and sensors together. The grid is often constructed across LAN, WAN, or Internet backbone networks at a regional, national, or global scale. Enterprises or organizations present grids as integrated computing resources. They can also be viewed as virtual platforms to support virtual organizations. The computers used in a grid are primarily workstations, servers, clusters, and supercomputers. Personal computers, laptops, and PDAs can be used as access devices to a grid system.

Grid Families

Grid technology demands new distributed computing models, software/middleware support, network protocols, and hardware infrastructures. National grid projects are followed by industrial grid platform development by IBM, Microsoft, Sun, HP, Dell, Cisco, EMC, Platform Computing, and others. New grid

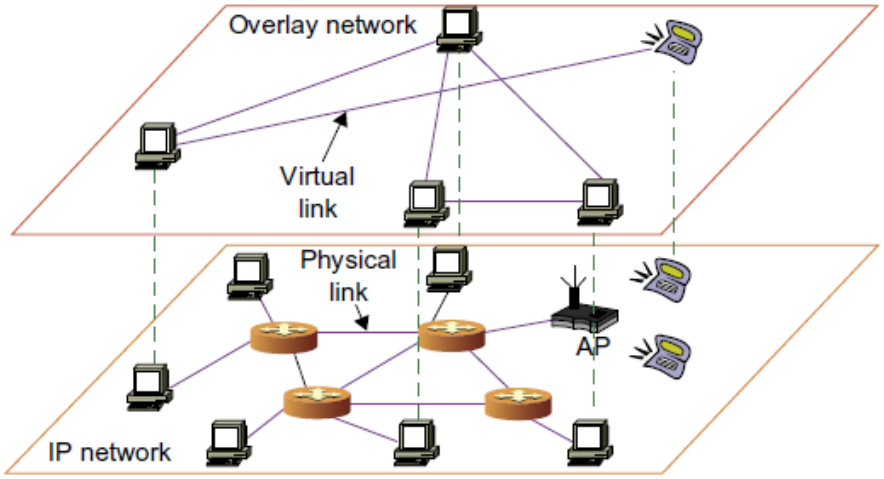
service providers (GSPs) and new grid applications have emerged rapidly, similar to the growth of Internet and web services in the past two decades.

Peer-to-Peer Network Families

An example of a well-established distributed system is the client-server architecture. In this scenario, client machines (PCs and workstations) are connected to a central server for compute, e-mail, file access, and database applications. The P2P architecture offers a distributed model of networked systems. First, a P2P network is client-oriented instead of server-oriented. In this section, P2P systems are introduced at the physical level and overlay networks at the logical level.

Overlay Networks

Data items or files are distributed in the participating peers. Based on communication or file-sharing needs, the peer IDs form an overlay network at the logical level.



Cloud Computing over the Internet

"A cloud is a pool of virtualized computer resources. A cloud can host a variety of different workloads, including batch-style backend jobs and interactive and user-facing applications." Based on this definition, a cloud allows workloads to be deployed and scaled out quickly through rapid provisioning of virtual or physical machines. The cloud supports redundant, self-recovering, highly scalable programming models that allow workloads to recover from many unavoidable hardware/software failures. Finally, the cloud system should be able to monitor resource use in real time to enable rebalancing of allocations when needed.

Internet Clouds

Cloud computing applies a virtualized platform with elastic resources on demand by provisioning hardware, software, and data sets dynamically. The idea is to move desktop computing to a service-oriented platform using server clusters and huge databases at data centers. Cloud computing leverages its low cost and simplicity to benefit both users and providers.

The Cloud Landscape

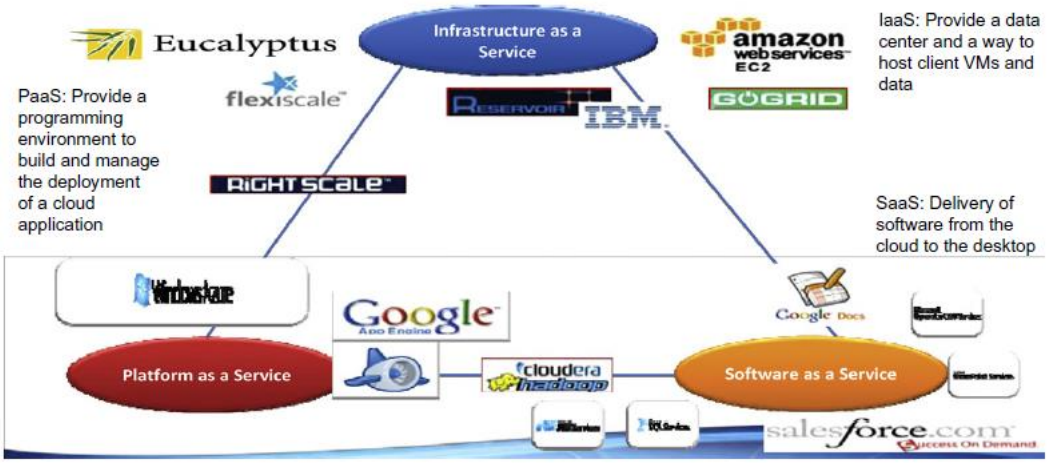
Traditionally, a distributed computing system tends to be owned and operated by an autonomous administrative domain (e.g., a research laboratory or company) for on-premises computing needs. However, these traditional systems have encountered several performance bottlenecks: constant system

maintenance, poor utilization, and increasing costs associated with hardware/software upgrades. Cloud computing as an on-demand computing paradigm resolves or relieves us from these problems.

• Infrastructure as a Service (IaaS) This model puts together infrastructures demanded by users−namely servers, storage, networks, and the data center fabric. The user can deploy and run on multiple VMs running guest OSes on specific applications. The user does not manage or control the underlying cloud infrastructure, but can specify when to request and release the needed resources.

• Platform as a Service (PaaS) This model enables the user to deploy user-built applications onto a virtualized cloud platform. PaaS includes middleware, databases, development tools, and some runtime support such as Web 2.0 and Java. The platform includes both hardware and software integrated with specific programming interfaces. The provider supplies the API and software tools (e.g., Java, Python, Web 2.0, .NET). The user is freed from managing the cloud infrastructure.

• Software as a Service (SaaS) This refers to browser-initiated application software over thousands of paid cloud customers. The SaaS model applies to business processes, industry applications, consumer relationship management (CRM), enterprise resources planning (ERP), human resources (HR), and collaborative applications. On the customer side, there is no upfront investment in servers or software licensing. On the provider side, costs are rather low, compared with conventional hosting of user applications.



SOFTWARE ENVIRONMENTS FOR DISTRIBUTED SYSTEMS AND CLOUDS

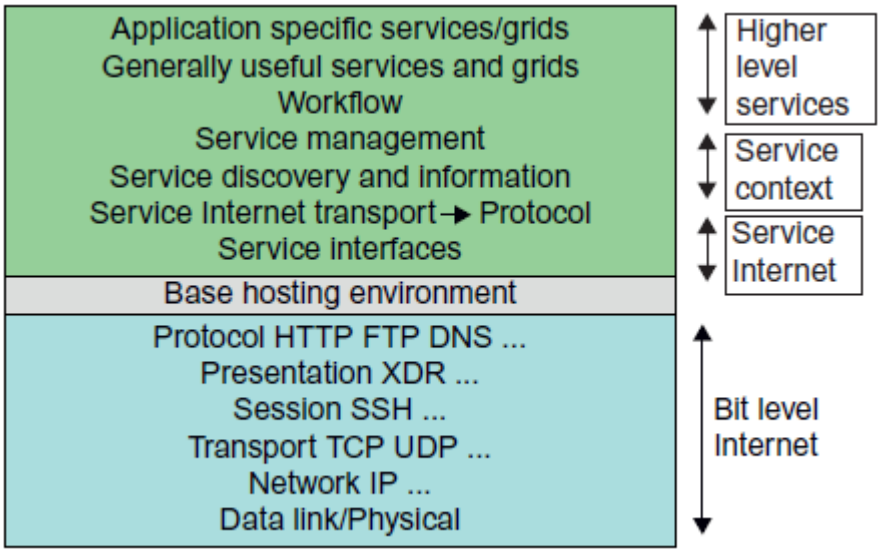
Service-Oriented Architecture (SOA)

In grids/web services, Java, and CORBA, an entity is, respectively, a service, a Java object, and a CORBA distributed object in a variety of languages. These architectures build on the traditional seven Open Systems Interconnection (OSI) layers that provide the base networking abstractions. On top of this we have a base software environment, which would be .NET or Apache Axis for web services, the Java Virtual Machine for Java, and a broker network for CORBA. On top of this base environment one would build a higher level environment reflecting the special features of the distributed computing environment.

Layered Architecture for Web Services and Grids

The entity interfaces correspond to the Web Services Description Language (WSDL), Java method, and CORBA interface definition language (IDL) specifications in these example distributed systems. These interfaces are linked with customized, high-level communication systems: SOAP, RMI, and IIOP in the three examples. These communication systems support features including particular message patterns (such as Remote Procedure Call or RPC), fault recovery, and specialized routing. Often, these communication systems are built on message-oriented middleware (enterprise bus) infrastructure such as Web- Sphere MQ or Java Message Service (JMS) which provide rich functionality and support virtualization of routing, senders, and recipients.

In the case of fault tolerance, the features in the Web Services Reliable Messaging (WSRM) framework mimic the OSI layer capability (as in TCP fault tolerance) modified to match the different abstractions (such as messages versus packets, virtualized addressing) at the entity levels. Security is a critical capability that either uses or reimplements the capabilities seen in concepts such as Internet Protocol Security (IPsec) and secure sockets in the OSI layers. Entity communication is supported by higher level services for registries, metadata, and management.



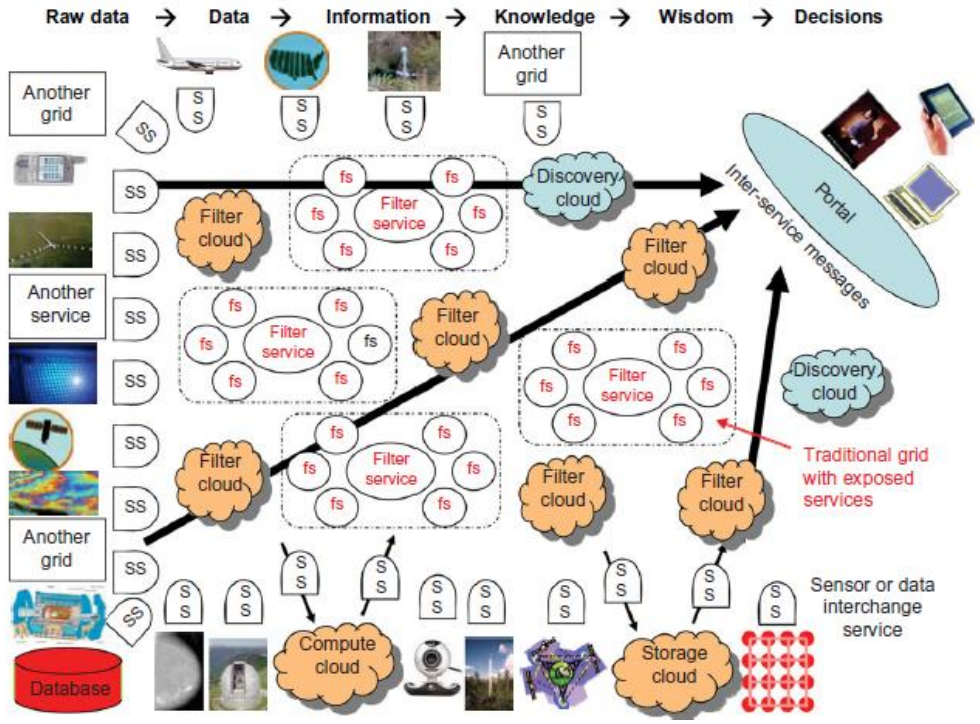
Web Services and Tools

Loose coupling and support of heterogeneous implementations make services more attractive than distributed objects. The above picture corresponds to two choices of service architecture: web services or REST systems. Both web services and REST systems have very distinct approaches to building reliable interoperable systems. In web services, one aims to fully specify all aspects of the service and its environment. This specification is carried with communicated messages using Simple Object Access Protocol (SOAP). The hosting environment then becomes a universal distributed operating system with fully distributed capability carried by SOAP messages. This approach has mixed success as it has been hard to agree on key parts of the protocol and even harder to efficiently implement the protocol by software such as Apache Axis.

The Evolution of SOA

As shown in Figure 1.21, service-oriented architecture (SOA) has evolved over the years. SOA applies to building grids, clouds, grids of clouds, clouds of grids, clouds of clouds (also known as interclouds), and systems of systems in general. A large number of sensors provide data-collection services, denoted in the

figure as SS (sensor service). A sensor can be a ZigBee device, a Bluetooth device, a WiFi access point, a personal computer, a GPA, or a wireless phone, among other things. Raw data is collected by sensor services. All the SS devices interact with large or small computers, many forms of grids, databases, the compute cloud, the storage cloud, the filter cloud, the discovery cloud, and so on. Filter services (fs in the figure) are used to eliminate unwanted raw data, in order to respond to specific requests from the web, the grid, or web services.



Grids versus Clouds

The boundary between grids and clouds are getting blurred in recent years. For web services, workflow technologies are used to coordinate or orchestrate services with certain specifications used to define critical business process models such as two-phase transactions.

In general, a grid system applies static resources, while a cloud emphasizes elastic resources. For some researchers, the differences between grids and clouds are limited only in dynamic resource allocation based on virtualization and autonomic computing. One can build a grid out of multiple clouds. This type of grid can do a better job than a pure cloud, because it can explicitly support negotiated resource allocation. Thus one may end up building with a system of systems: such as a cloud of clouds, a grid of clouds, or a cloud of grids, or inter-clouds as a basic SOA architecture.

Trends toward Distributed Operating Systems

A distributed system inherently has multiple system images. This is mainly due to the fact that all node machines run with an independent operating system. To promote resource sharing and fast communication among node machines, it is best to have a distributed OS that manages all resources coherently and efficiently. Such a system is most likely to be a closed system, and it will likely rely on message passing and RPCs for internode communications. It should be pointed out that a distributed OS is crucial for upgrading the performance, efficiency, and flexibility of distributed applications.

PERFORMANCE, SECURITY, AND ENERGY EFFICIENCY

Performance Metrics and Scalability Analysis

Performance metrics are needed to measure various distributed systems. In this section, we will discuss various dimensions of scalability and performance laws. Then we will examine system scalability against OS images and the limiting factors encountered.

Performance Metrics

In a distributed system, performance is attributed to a large number of factors. System throughput is often measured in MIPS, Tflops (tera floating-point operations per second), or TPS (transactions per second). Other measures include job response time and network latency. An interconnection network that has low latency and high bandwidth is preferred. System overhead is often attributed to OS boot time, compile time, I/O data rate, and the runtime support system used. Other performance-related metrics include the QoS for Internet and web services; system availability and dependability; and security resilience for system defense against network attacks.

Dimensions of Scalability

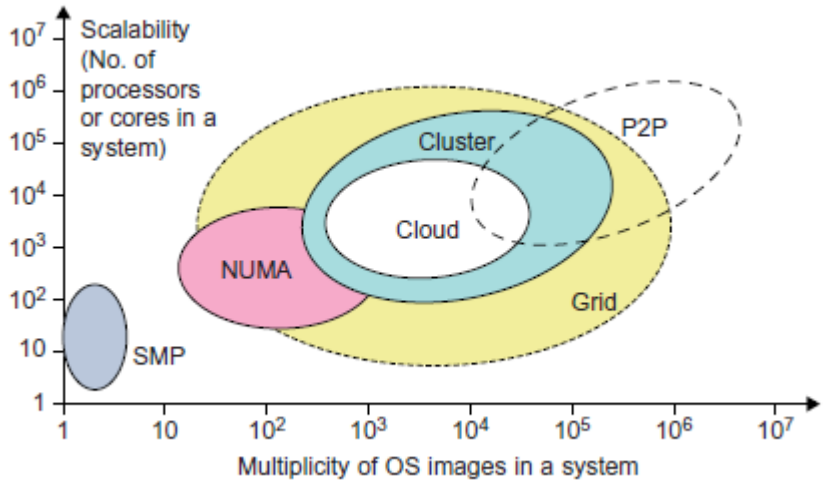
Users want to have a distributed system that can achieve scalable performance. Any resource upgrade in a system should be backward compatible with existing hardware and software resources.

• Size scalability This refers to achieving higher performance or more functionality by increasing the machine size. The word "size" refers to adding processors, cache, memory, storage, or I/O channels. The most obvious way to determine size scalability is to simply count the number of processors installed. Not all parallel computer or distributed architectures are equally sizescalable. For example, the IBM S2 was scaled up to 512 processors in 1997. But in 2008, the IBM BlueGene/L system scaled up to 65,000 processors.

• Software scalability This refers to upgrades in the OS or compilers, adding mathematical and engineering libraries, porting new application software, and installing more user-friendly programming environments. Some software upgrades may not work with large system configurations. Testing and fine-tuning of new software on larger systems is a nontrivial job.

• Application scalability This refers to matching problem size scalability with machine size scalability. Problem size affects the size of the data set or the workload increase. Instead of increasing machine size, users can enlarge the problem size to enhance system efficiency or cost-effectiveness.

• Technology scalability This refers to a system that can adapt to changes in building technologies, such as the component and networking technologies discussed in Section 3.1. When scaling a system design with new technology one must consider three aspects: time, space, and heterogeneity. (1) Time refers to generation scalability. When changing to new-generation processors, one must consider the impact to the motherboard, power supply, packaging and cooling, and so forth. Based on past experience, most systems upgrade their commodity processors every three to five years. (2) Space is related to packaging and energy concerns. Technology scalability demands harmony and portability among suppliers. (3) Heterogeneity refers to the use of hardware components or software packages from different vendors. Heterogeneity may limit the scalability.



Amdahl's Law

Consider the execution of a given program on a uniprocessor workstation with a total execution time of T minutes. Now, let's say the program has been parallelized or partitioned for parallel execution on a cluster of many processing nodes. Assume that a fraction α of the code must be executed sequentially, called the *sequential bottleneck*. Therefore, $(1-\alpha)$ of the code can be compiled for parallel execution by n processors. The total execution time of the program is calculated by $\alpha T + (1-\alpha)T/n$, where the first term is the sequential execution time on a single processor and the second term is the parallel execution time on n processing nodes.

All system or communication overhead is ignored here. The I/O time or exception handling time is also not included in the following speedup analysis. Amdahl's Law states that the *speedup factor* of using the n-processor system over the use of a single processor is expressed by:

$$\text{Speedup} = S = T/[\alpha T + (1-\alpha)T/n] = 1/[\alpha + (1-\alpha)/n]$$

Fault Tolerance and System Availability - System Availability

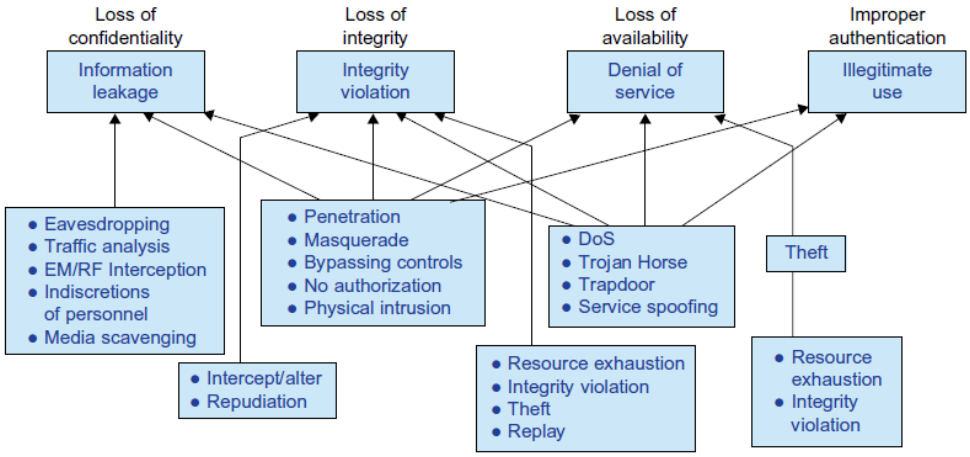
HA (high availability) is desired in all clusters, grids, P2P networks, and cloud systems. A system is highly available if it has a long mean time to failure (MTTF) and a short mean time to repair (MTTR). System availability is formally defined as follows:

System Availability =MTTF/(MTTF +MTTR)
System availability is attributed to many factors. All hardware, software, and network components may fail. Any failure that will pull down the operation of the entire system is called a single point of failure. The rule of thumb is to design a dependable computing system with no single point of failure. Adding hardware redundancy, increasing component reliability, and designing for testability will help to enhance system availability and dependability.

Network Threats and Data Integrity - Threats to Systems and Networks
Network viruses have threatened many users in widespread attacks. These incidents have created a worm epidemic by pulling down many routers and servers, and are responsible for the loss of billions of dollars

in business, government, and services. Figure 1.25 summarizes various attack types and their potential damage to users. As the figure shows, information leaks lead to a loss of confidentiality.

Loss of data integrity may be caused by user alteration, Trojan horses, and service spoofing attacks. A denial of service (DoS) results in a loss of system operation and Internet connections.



Copyright Protection

Collusive piracy is the main source of intellectual property violations within the boundary of a P2P network. Paid clients (colluders) may illegally share copyrighted content files with unpaid clients (pirates). Online piracy has hindered the use of open P2P networks for commercial content delivery.

Energy Efficiency in Distributed Computing

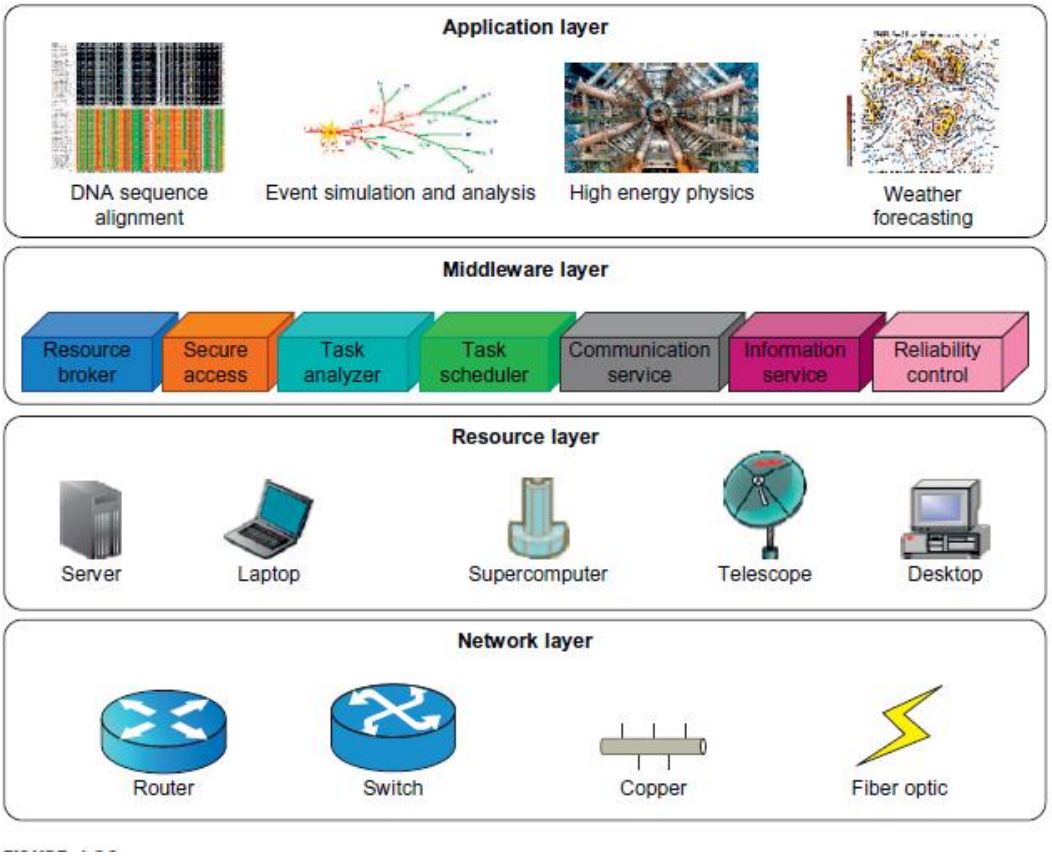
Primary performance goals in conventional parallel and distributed computing systems are high performance and high throughput, considering some form of performance reliability (e.g., fault tolerance and security). However, these systems recently encountered new challenging issues including energy efficiency, and workload and resource outsourcing. These emerging issues are crucial not only on their own, but also for the sustainability of large-scale computing systems in general. This section reviews energy consumption issues in servers and HPC systems, an area known as distributed power management (DPM).

Energy Consumption of Unused Servers

To run a server farm (data center) a company has to spend a huge amount of money for hardware, software, operational support, and energy every year. Therefore, companies should thoroughly identify whether their installed server farm (more specifically, the volume of provisioned resources) is at an appropriate level, particularly in terms of utilization. It was estimated in the past that, on average, one-sixth (15 percent) of the full-time servers in a company are left powered on without being actively used (i.e., they are idling) on a daily basis. This indicates that with 44 million servers in the world, around 4.7 million servers are not doing any useful work.

Reducing Energy in Active Servers

In addition to identifying unused/underutilized servers for energy savings, it is also necessary to apply appropriate techniques to decrease energy consumption in active distributed systems with negligible influence on their performance.



CHAPTER

Virtual Machines and Virtualization of Clusters and Data Centers

3

CHAPTER OUTLINE

SUMMARY

The reincarnation of *virtual machines* (VMs) presents a great opportunity for parallel, cluster, grid, cloud, and distributed computing. Virtualization technology benefits the computer and IT industries by enabling users to share expensive hardware resources by multiplexing VMs on the same set of hardware hosts. This chapter covers virtualization levels, VM architectures, virtual networking, virtual cluster construction, and virtualized data-center design and automation in cloud computing. In particular, the designs of dynamically structured clusters, grids, and clouds are presented with VMs and virtual clusters.

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 idea of VMs can be dated back to the 1960s [53]. 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. This virtualization technology has been revitalized as the demand for distributed and cloud computing increased sharply in recent years [41].

The idea is to separate the hardware from the software to yield better system efficiency. For example, computer users gained access to much enlarged memory space when the concept of *virtual memory* was introduced. Similarly, virtualization techniques can be applied to enhance the use of compute engines, networks, and storage. In this chapter we will discuss VMs and their applications for building distributed systems. According to a 2009 Gartner Report, virtualization was the top strategic technology poised to change the computer industry. With sufficient storage, any computer platform can be installed in another host computer, even if they use processors with different instruction sets and run with distinct operating systems on the same hardware.

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* as shown in Figure 3.1(b). This virtualization layer is known as *hypervisor* or *virtual machine monitor* (VMM) [54]. The VMs are shown in the upper boxes, where applications run with their own guest OS over the virtualized CPU, memory, and I/O resources.

The main function of the software layer for virtualization is to virtualize the physical hardware of a host machine into virtual resources to be used by the VMs, exclusively. This can be implemented at various operational levels, as we will discuss shortly. 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).

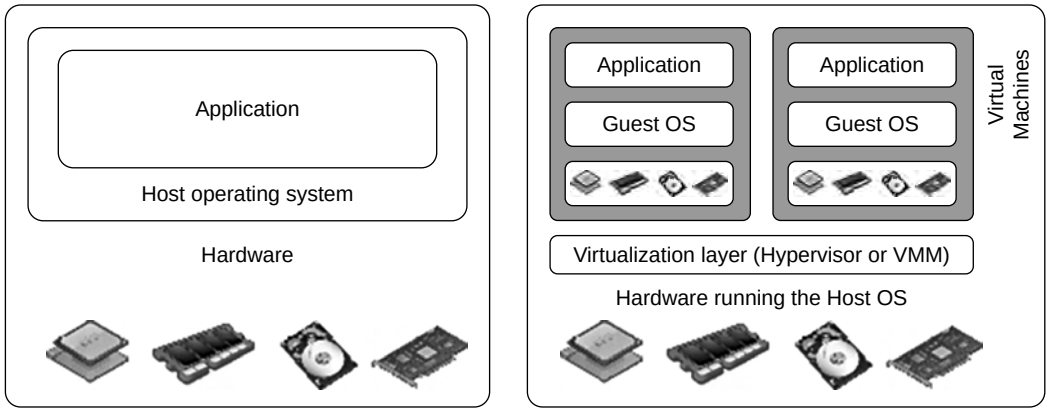


FIGURE 3.1

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

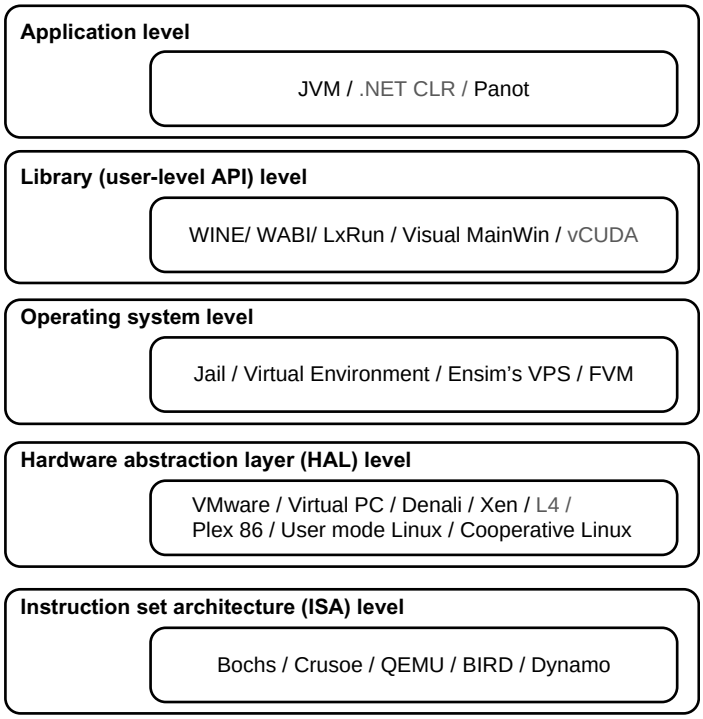


FIGURE 3.2

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. 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

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. The Microsoft .NET CLR and *Java Virtual Machine (JVM)* are two good examples of this class of VM.

Other forms of application-level virtualization are known as *application isolation*, *application sandboxing*, or *application streaming*. The process involves wrapping the application in a layer that is isolated from the host OS and other applications. The result is an application that is much easier to distribute and remove from user workstations. An example is the LANDesk application virtualization platform which deploys software applications as self-contained, executable files in an isolated environment without requiring installation, system modifications, or elevated security privileges.

3.1.1.6 Relative Merits of Different Approaches

Table 3.1 compares the relative merits of implementing virtualization at various levels. The column headings correspond to four technical merits. "Higher Performance" and "Application Flexibility" are self-explanatory. "Implementation Complexity" implies the cost to implement that particular virtualization level. "Application Isolation" refers to the effort required to isolate resources committed to different VMs. Each row corresponds to a particular level of virtualization.

The number of X's in the table cells reflects the advantage points of each implementation level. Five X's implies the best case and one X implies the worst case. Overall, hardware and OS support will yield the highest performance. However, the hardware and application levels are also the most expensive to implement. User isolation is the most difficult to achieve. ISA implementation offers the best application flexibility.

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. Therefore, several traditional operating systems which are the same or different can sit on the same set of hardware simultaneously.

Table 3.1 Relative Merits of Virtualization at Various Levels (More "X"'s Means Higher Merit, with a Maximum of 5 X's)

Level of Implementation	Higher Performance	Application Flexibility	Implementation Complexity	Application Isolation
ISA	X	XXXXX	XXX	XXX
Hardware-level virtualization	XXXXX	XXX	XXXXX	XXXX
OS-level virtualization	XXXXX	XX	XXX	XX
Runtime library support	XXX	XX	XX	XX
User application level	XX	XX	XXXXX	XXXXX

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

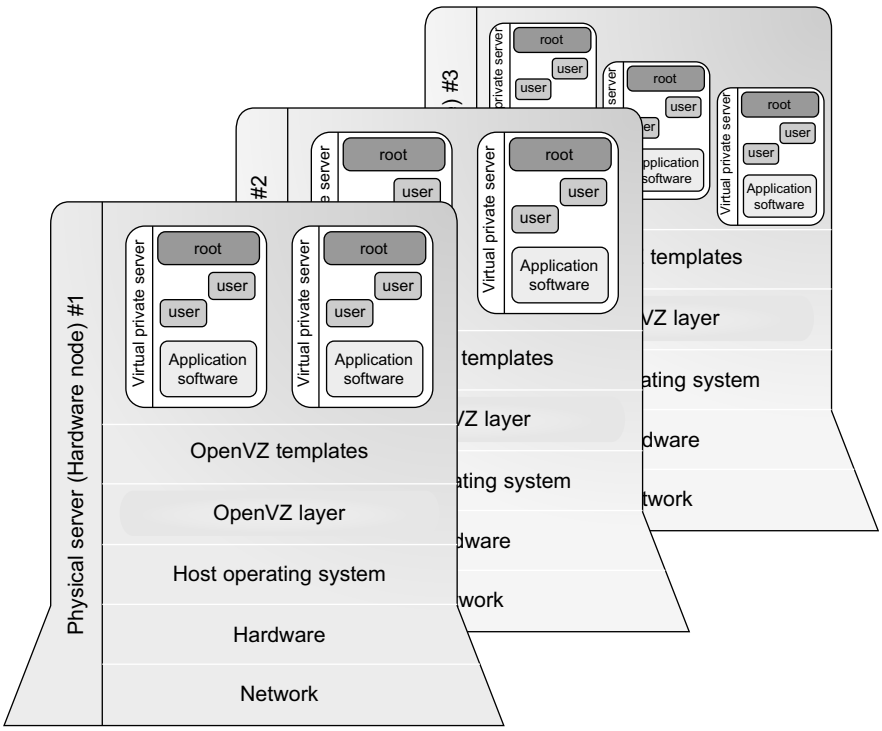


FIGURE 3.3

The OpenVZ virtualization layer inside the host OS, which provides some OS images to create VMs quickly. (*Courtesy of OpenVZ User's Guide [65]*)

computing, the first and second benefits can be used to overcome the defects of slow initialization of VMs at the hardware level, and being unaware of the current application state, respectively.

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. For example, a Windows distribution such as Windows XP cannot run on a Linux-based container. However, users of cloud computing have various preferences. Some prefer Windows and others prefer Linux or other operating systems. Therefore, there is a challenge for OS-level virtualization in such cases.

Figure 3.3 illustrates the concept of OS-level virtualization. The virtualization layer is inserted inside the OS to partition the hardware resources for multiple VMs to run their applications in multiple virtual environments. To implement OS-level virtualization, isolated execution environments (VMs) should be created based on a single OS kernel. Furthermore, the access requests from a VM need to be redirected to the VM's local resource partition on the physical machine. For

example, the *chroot* command in a UNIX system can create several virtual root directories within a host OS. These virtual root directories are the root directories of all VMs created.

There are two ways to implement virtual root directories: duplicating common resources to each VM partition; or sharing most resources with the host environment and only creating private resource copies on the VM on demand. The first way incurs significant resource costs and overhead on a physical machine. This issue neutralizes the benefits of OS-level virtualization, compared with hardware-assisted virtualization. Therefore, OS-level virtualization is often a second choice.

3.1.3.4 Virtualization on Linux or Windows Platforms

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

However, most Linux platforms are not tied to a special kernel. In such a case, a host can run several VMs simultaneously on the same hardware. Table 3.3 summarizes several examples of OS-level virtualization tools that have been developed in recent years. Two OS tools (Linux vServer and OpenVZ) support Linux platforms to run other platform-based applications through virtualization. These two OS-level tools are illustrated in Example 3.1. The third tool, FVM, is an attempt specifically developed for virtualization on the Windows NT platform.

Example 3.1 Virtualization Support for the Linux Platform

OpenVZ is an OS-level tool designed to support Linux platforms to create virtual environments for running VMs under different guest OSes. OpenVZ is an open source container-based virtualization solution built on Linux. To support virtualization and isolation of various subsystems, limited resource management, and checkpointing, OpenVZ modifies the Linux kernel. The overall picture of the OpenVZ system is illustrated in Figure 3.3. Several VPSes can run simultaneously on a physical machine. These VPSes look like normal

Table 3.3 Virtualization Support for Linux and Windows NT Platforms

Virtualization Support and Source of Information	Brief Introduction on Functionality and Application Platforms
Linux vServer for Linux platforms (http://linux-vserver.org/)	Extends Linux kernels to implement a security mechanism to help build VMs by setting resource limits and file attributes and changing the root environment for VM isolation
OpenVZ for Linux platforms [65]; http://ftp.openvz.org/doc/OpenVZ-Users-Guide.pdf)	Supports virtualization by creating *virtual private servers (VPSes)*; the VPS has its own files, users, process tree, and virtual devices, which can be isolated from other VPSes, and checkpointing and live migration are supported
FVM (Feather-Weight Virtual Machines) for virtualizing the Windows NT platforms [78])	Uses system call interfaces to create VMs at the NY kernel space; multiple VMs are supported by virtualized namespace and copy-on-write

Linux servers. Each VPS has its own files, users and groups, process tree, virtual network, virtual devices, and IPC through semaphores and messages.

The resource management subsystem of OpenVZ consists of three components: two-level disk allocation, a two-level CPU scheduler, and a resource controller. The amount of disk space a VM can use is set by the OpenVZ server administrator. This is the first level of disk allocation. Each VM acts as a standard Linux system. Hence, the VM administrator is responsible for allocating disk space for each user and group. This is the second-level disk quota. The first-level CPU scheduler of OpenVZ decides which VM to give the time slice to, taking into account the virtual CPU priority and limit settings.

The second-level CPU scheduler is the same as that of Linux. OpenVZ has a set of about 20 parameters which are carefully chosen to cover all aspects of VM operation. Therefore, the resources that a VM can use are well controlled. OpenVZ also supports checkpointing and live migration. The complete state of a VM can quickly be saved to a disk file. This file can then be transferred to another physical machine and the VM can be restored there. It only takes a few seconds to complete the whole process. However, there is still a delay in processing because the established network connections are also migrated.

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. This section provides an overview of several library-level virtualization systems: namely the *Windows Application Binary Interface (WABI)*, lxrun, WINE, Visual MainWin, and vCUDA, which are summarized in Table 3.4.

Table 3.4 Middleware and Library Support for Virtualization

Middleware or Runtime Library and References or Web Link	Brief Introduction and Application Platforms
WABI (http://docs.sun.com/app/docs/doc/802-6306)	Middleware that converts Windows system calls running on x86 PCs to Solaris system calls running on SPARC workstations
Lxrun (Linux Run) (http://www.ugcs.caltech.edu/~steven/lxrun/)	A system call emulator that enables Linux applications written for x86 hosts to run on UNIX systems such as the SCO OpenServer
WINE (http://www.winehq.org/)	A library support system for virtualizing x86 processors to run Windows applications under Linux, FreeBSD, and Solaris
Visual MainWin (http://www.mainsoft.com/)	A compiler support system to develop Windows applications using Visual Studio to run on Solaris, Linux, and AIX hosts
vCUDA (Example 3.2) (IEEE *IPDPS* 2009 [57])	Virtualization support for using general-purpose GPUs to run data-intensive applications under a special guest OS

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. The vCUDA is explained in Example 3.2 with a graphical illustration in Figure 3.4.

Example 3.2 The vCUDA for Virtualization of General-Purpose GPUs

CUDA is a programming model and library for general-purpose GPUs. It leverages the high performance of GPUs to run compute-intensive applications on host operating systems. However, it is difficult to run CUDA applications on hardware-level VMs directly. vCUDA virtualizes the CUDA library and can be installed on guest OSes. When CUDA applications run on a guest OS and issue a call to the CUDA API, vCUDA intercepts the call and redirects it to the CUDA API running on the host OS. Figure 3.4 shows the basic concept of the vCUDA architecture [57].

The vCUDA employs a client-server model to implement CUDA virtualization. It consists of three user space components: the vCUDA library, a virtual GPU in the guest OS (which acts as a client), and the vCUDA stub in the host OS (which acts as a server). The vCUDA library resides in the guest OS as a substitute for the standard CUDA library. It is responsible for intercepting and redirecting API calls from the client to the stub. Besides these tasks, vCUDA also creates vGPUs and manages them.

The functionality of a vGPU is threefold: It abstracts the GPU structure and gives applications a uniform view of the underlying hardware; when a CUDA application in the guest OS allocates a device's memory the vGPU can return a local virtual address to the application and notify the remote stub to allocate the real device memory, and the vGPU is responsible for storing the CUDA API flow. The vCUDA stub receives

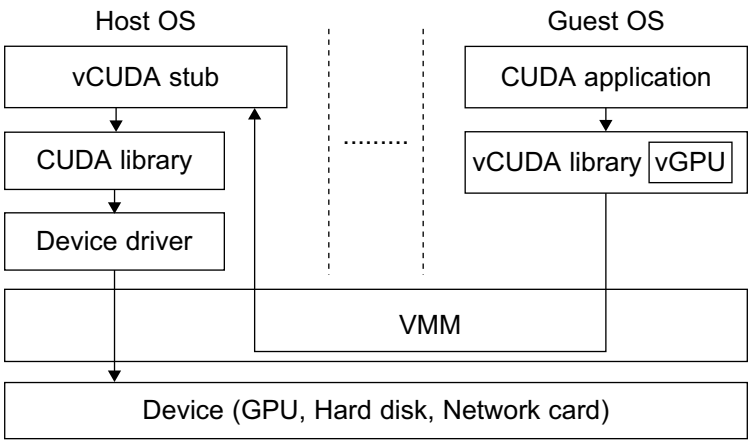


FIGURE 3.4

Basic concept of the vCUDA architecture.

(*Courtesy of Lin Shi, et al. [57]*)

and interprets remote requests and creates a corresponding execution context for the API calls from the guest OS, then returns the results to the guest OS. The vCUDA stub also manages actual physical resource allocation.

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, *paravirtualization*, 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 *hypercalls* for the guest OSes and applications. Depending on the functionality, a hypervisor can assume a *micro-kernel architecture* like the Microsoft Hyper-V. Or it can assume 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.

3.2.1.1 The Xen Architecture

Xen is an open source hypervisor program developed by Cambridge University. Xen is a micro-kernel hypervisor, which separates the policy from the mechanism. The Xen hypervisor implements all the mechanisms, leaving the policy to be handled by Domain 0, as shown in Figure 3.5. Xen does not include any device drivers natively [7]. It just provides a mechanism by which a guest OS can have direct access to the physical devices. As a result, the size of the Xen hypervisor is kept rather small. Xen provides a virtual environment located between the hardware and the OS. A number of vendors are in the process of developing commercial Xen hypervisors, among them are Citrix XenServer [62] and Oracle VM [42].

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

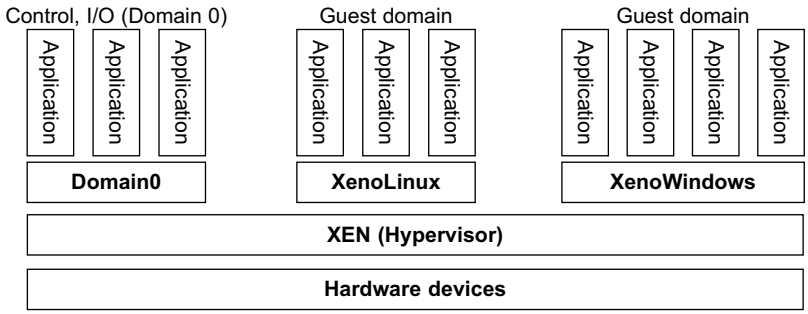


FIGURE 3.5

The Xen architecture's special domain 0 for control and I/O, and several guest domains for user applications. (*Courtesy of P. Barham, et al. [7]*)

particular controls the others. The guest OS, which has control ability, is called Domain 0, and the others are called Domain U. Domain 0 is a privileged guest OS of Xen. It is first loaded when Xen boots without any file system drivers being available. Domain 0 is designed to access hardware directly and manage devices. Therefore, one of the responsibilities of Domain 0 is to allocate and map hardware resources for the guest domains (the Domain U domains).

For example, Xen is based on Linux and its security level is C2. Its management VM is named Domain 0, which has the privilege to manage other VMs implemented on the same host. If Domain 0 is compromised, the hacker can control the entire system. So, in the VM system, security policies are needed to improve the security of Domain 0. Domain 0, behaving as a VMM, allows users to create, copy, save, read, modify, share, migrate, and roll back VMs as easily as manipulating a file, which flexibly provides tremendous benefits for users. Unfortunately, it also brings a series of security problems during the software life cycle and data lifetime.

Traditionally, a machine's lifetime can be envisioned as a straight line where the current state of the machine is a point that progresses monotonically as the software executes. During this time, configuration changes are made, software is installed, and patches are applied. In such an environment, the VM state is akin to a tree: At any point, execution can go into *N* different branches where multiple instances of a VM can exist at any point in this tree at any given time. VMs are allowed to roll back to previous states in their execution (e.g., to fix configuration errors) or rerun from the same point many times (e.g., as a means of distributing dynamic content or circulating a "live" system image).

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. Why are only critical instructions trapped into the VMM? This is because binary translation can incur a large performance overhead. 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. The guest OS is completely decoupled from the underlying hardware. Consequently, the guest OS is unaware that it is being virtualized.

The performance of full virtualization may not be ideal, because it involves binary translation which is rather time-consuming. In particular, the full virtualization of I/O-intensive applications is a really a big challenge. Binary translation employs a code cache to store translated hot instructions to improve performance, but it increases the cost of memory usage. At the time of this writing, the performance of full virtualization on the x86 architecture is typically 80 percent to 97 percent that of the host machine.

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.

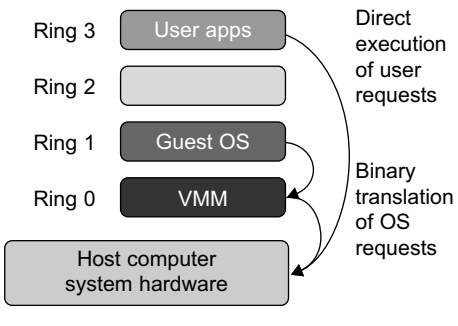


FIGURE 3.6

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. (*Courtesy of VM Ware [71]*)

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. 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 nonvirtualizable instructions with *hypercalls* 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.

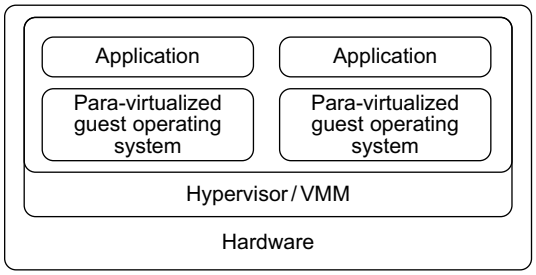


FIGURE 3.7

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.)

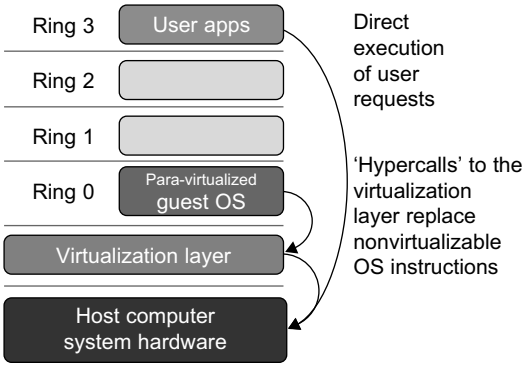


FIGURE 3.8

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

(*Courtesy of VMWare [71]*)

Although para-virtualization reduces the overhead, it has incurred other problems. First, its compatibility and portability may be in doubt, because it must support the unmodified OS as well. Second, the cost of maintaining para-virtualized OSes is high, because they may require deep OS kernel modifications. Finally, the performance advantage of para-virtualization varies greatly due to workload variations. 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.

3.2.3.2 KVM (Kernel-Based VM)

This is a Linux para-virtualization system—a part of the Linux version 2.6.20 kernel. Memory management and scheduling activities are carried out by the existing Linux kernel. The KVM does the rest, which makes it simpler than the hypervisor that controls the entire machine. KVM is a hardware-assisted para-virtualization tool, which improves performance and supports unmodified guest OSes such as Windows, Linux, Solaris, and other UNIX variants.

3.2.3.3 Para-Virtualization with Compiler Support

Unlike the full virtualization architecture which intercepts and emulates privileged and sensitive instructions at runtime, para-virtualization handles these instructions at compile time. The guest OS kernel is modified to replace the privileged and sensitive instructions with hypercalls to the hypervisor or VMM. Xen assumes such a para-virtualization architecture.

The guest OS running in a guest domain may run at Ring 1 instead of at Ring 0. This implies that the guest OS may not be able to execute some privileged and sensitive instructions. The privileged instructions are implemented by hypercalls to the hypervisor. After replacing the instructions with hypercalls, the modified guest OS emulates the behavior of the original guest OS. On an UNIX system, a system call involves an interrupt or service routine. The hypercalls apply a dedicated service routine in Xen.

Example 3.3 VMware ESX Server for Para-Virtualization

VMware pioneered the software market for virtualization. The company has developed virtualization tools for desktop systems and servers as well as virtual infrastructure for large data centers. ESX is a VMM or a hypervisor for bare-metal x86 symmetric multiprocessing (SMP) servers. It accesses hardware resources such as I/O directly and has complete resource management control. An ESX-enabled server consists of four components: a virtualization layer, a resource manager, hardware interface components, and a service console, as shown in Figure 3.9. To improve performance, the ESX server employs a para-virtualization architecture in which the VM kernel interacts directly with the hardware without involving the host OS.

The VMM layer virtualizes the physical hardware resources such as CPU, memory, network and disk controllers, and human interface devices. Every VM has its own set of virtual hardware resources. The resource manager allocates CPU, memory disk, and network bandwidth and maps them to the virtual hardware resource set of each VM created. Hardware interface components are the device drivers and the

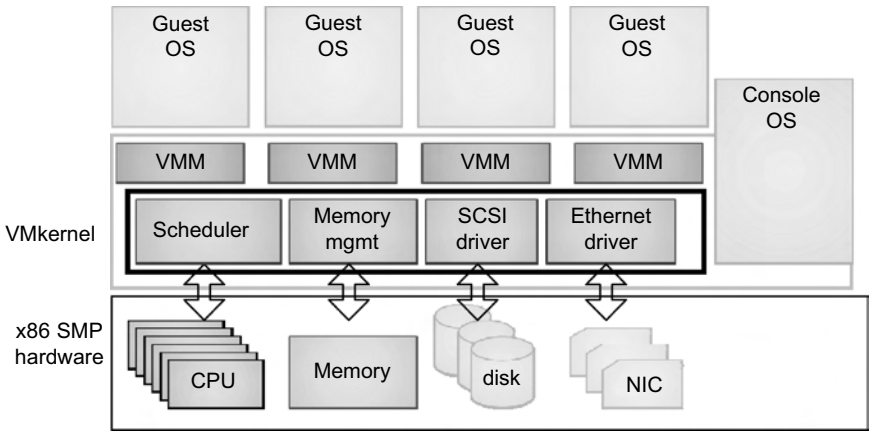


FIGURE 3.9

The VMware ESX server architecture using para-virtualization.

(*Courtesy of VMware [71]*)

VMware ESX Server File System. The service console is responsible for booting the system, initiating the execution of the VMM and resource manager, and relinquishing control to those layers. It also facilitates the process for system administrators.

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

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

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 environment, it is more difficult to make OSes and applications run correctly because there are more layers in the machine stack. Example 3.4 discusses Intel's hardware support approach.

At the time of this writing, many hardware virtualization products were available. The VMware Workstation is a VM software suite for x86 and x86-64 computers. This software suite allows users to set up multiple x86 and x86-64 virtual computers and to use one or more of these VMs simultaneously with the host operating system. The VMware Workstation assumes the host-based virtualization. Xen is a hypervisor for use in IA-32, x86-64, Itanium, and PowerPC 970 hosts. Actually, Xen modifies Linux as the lowest and most privileged layer, or a hypervisor.

One or more guest OS can run on top of the hypervisor. KVM (*Kernel-based Virtual Machine*) is a Linux kernel virtualization infrastructure. KVM can support hardware-assisted virtualization and paravirtualization by using the Intel VT-x or AMD-v and VirtIO framework, respectively. The VirtIO framework includes a paravirtual Ethernet card, a disk I/O controller, a balloon device for adjusting guest memory usage, and a VGA graphics interface using VMware drivers.

Example 3.4 Hardware Support for Virtualization in the Intel x86 Processor

Since software-based virtualization techniques are complicated and incur performance overhead, Intel provides a hardware-assist technique to make virtualization easy and improve performance. Figure 3.10 provides an overview of Intel's full virtualization techniques. For processor virtualization, Intel offers the VT-x or VT-i technique. VT-x adds a privileged mode (VMX Root Mode) and some instructions to processors. This enhancement traps all sensitive instructions in the VMM automatically. For memory virtualization, Intel offers the EPT, which translates the virtual address to the machine's physical addresses to improve performance. For I/O virtualization, Intel implements VT-d and VT-c to support this.

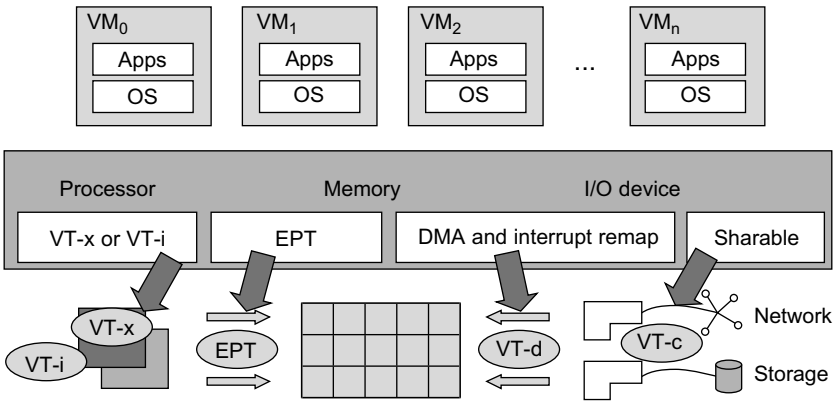


FIGURE 3.10

Intel hardware support for virtualization of processor, memory, and I/O devices.

(*Modified from [68], Courtesy of Lizhong Chen, USC*)

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. RISC CPU architectures can be naturally virtualized because all control- and behavior-sensitive instructions are privileged instructions. On the contrary, x86 CPU architectures are not primarily designed to support virtualization. This is because about 10 sensitive instructions, such as *SGDT* and *SMSW*, are not privileged instructions. When these instructions execute in virtualization, they cannot be trapped in the VMM.

On a native UNIX-like system, a system call triggers the *80h* interrupt and passes control to the OS kernel. The interrupt handler in the kernel is then invoked to process the system call. On a paravirtualization system such as Xen, a system call in the guest OS first triggers the *80h* interrupt normally. Almost at the same time, the *82h* interrupt in the hypervisor is triggered. Incidentally, control is passed on to the hypervisor as well. When the hypervisor completes its task for the guest OS system call, it passes control back to the guest OS kernel. Certainly, the guest OS kernel may also invoke the hypercall while it's running. Although paravirtualization of a CPU lets unmodified applications run in the VM, it causes a small performance penalty.

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.

Example 3.5 Intel Hardware-Assisted CPU Virtualization

Although x86 processors are not virtualizable primarily, great effort is taken to virtualize them. They are used widely in comparing RISC processors that the bulk of x86-based legacy systems cannot discard easily. Virtualization of x86 processors is detailed in the following sections. Intel's VT-x technology is an example of hardware-assisted virtualization, as shown in Figure 3.11. Intel calls the privilege level of x86 processors the VMX Root Mode. In order to control the start and stop of a VM and allocate a memory page to maintain the

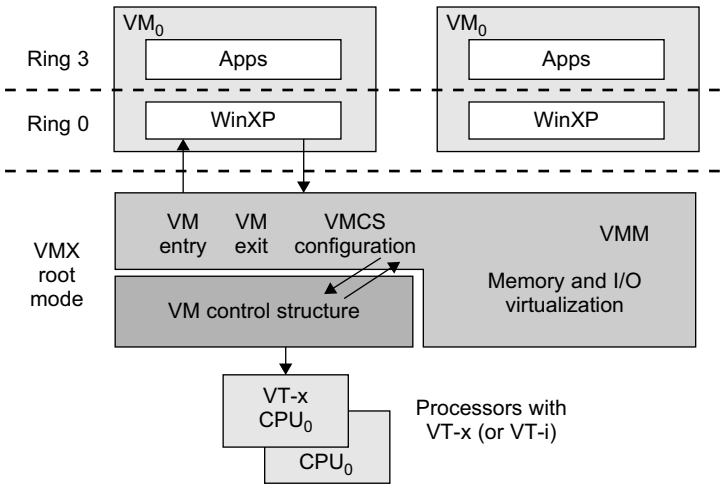


FIGURE 3.11

Intel hardware-assisted CPU virtualization.

(*Modified from [68], Courtesy of Lizhong Chen, USC*)

CPU state for VMs, a set of additional instructions is added. At the time of this writing, Xen, VMware, and the Microsoft Virtual PC all implement their hypervisors by using the VT-x technology.

Generally, hardware-assisted virtualization should have high efficiency. However, since the transition from the hypervisor to the guest OS incurs high overhead switches between processor modes, it sometimes cannot outperform binary translation. Hence, virtualization systems such as VMware now use a hybrid approach, in which a few tasks are offloaded to the hardware but the rest is still done in software. In addition, para-virtualization and hardware-assisted virtualization can be combined to improve the performance further.

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.

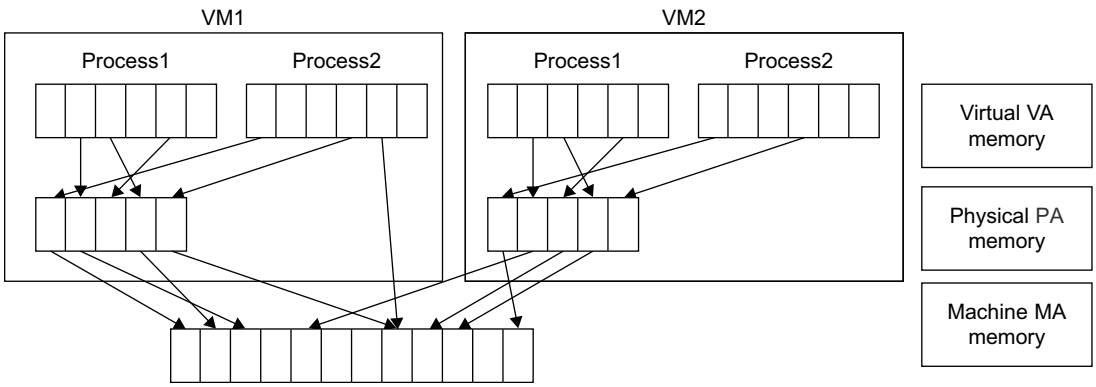


FIGURE 3.12

Two-level memory mapping procedure.

(*Courtesy of R. Rblig, et al. [68]*)

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. The AMD Barcelona processor has featured hardware-assisted memory virtualization since 2007. It provides hardware assistance to the two-stage address translation in a virtual execution environment by using a technology called nested paging.

Example 3.6 Extended Page Table by Intel for Memory Virtualization

Since the efficiency of the software shadow page table technique was too low, Intel developed a hardware-based EPT technique to improve it, as illustrated in Figure 3.13. In addition, Intel offers a Virtual Processor ID (VPID) to improve use of the TLB. Therefore, the performance of memory virtualization is greatly improved. In Figure 3.13, the page tables of the guest OS and EPT are all four-level.

When a virtual address needs to be translated, the CPU will first look for the L4 page table pointed to by Guest CR3. Since the address in Guest CR3 is a physical address in the guest OS, the CPU needs to convert the Guest CR3 GPA to the host physical address (HPA) using EPT. In this procedure, the CPU will check the EPT TLB to see if the translation is there. If there is no required translation in the EPT TLB, the CPU will look for it in the EPT. If the CPU cannot find the translation in the EPT, an EPT violation exception will be raised.

When the GPA of the L4 page table is obtained, the CPU will calculate the GPA of the L3 page table by using the GVA and the content of the L4 page table. If the entry corresponding to the GVA in the L4

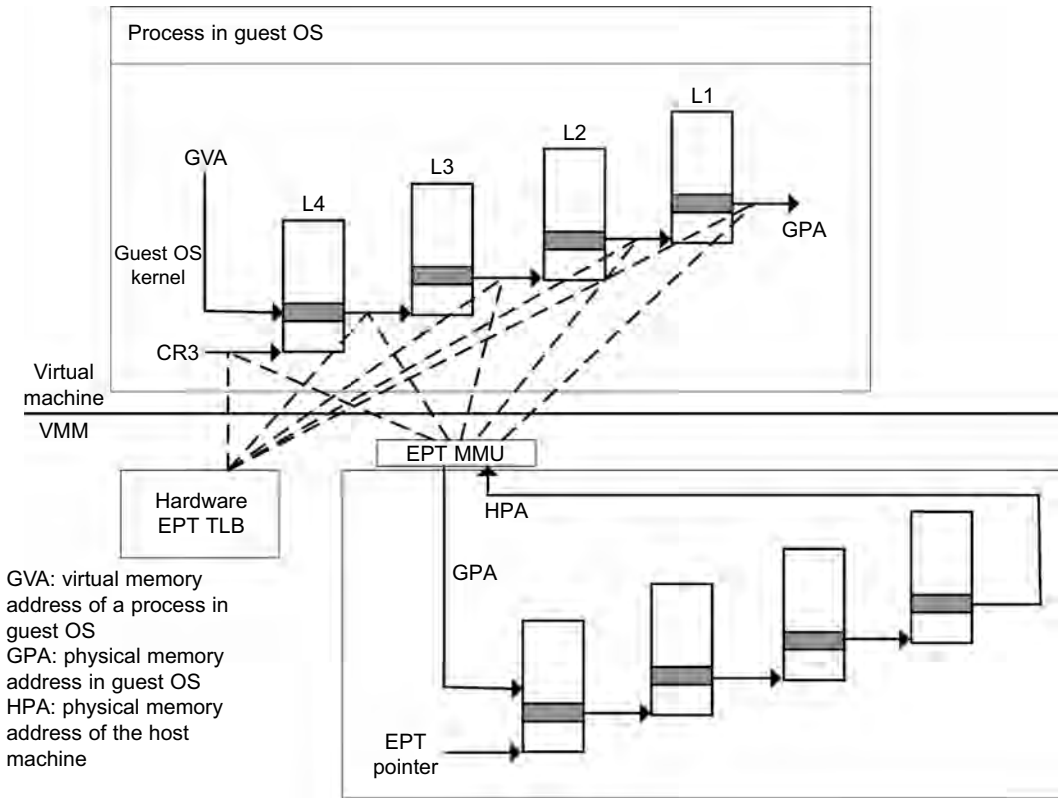


FIGURE 3.13

Memory virtualization using EPT by Intel (the EPT is also known as the shadow page table [68]).

page table is a page fault, the CPU will generate a page fault interrupt and will let the guest OS kernel handle the interrupt. When the PGA of the L3 page table is obtained, the CPU will look for the EPT to get the HPA of the L3 page table, as described earlier. To get the HPA corresponding to a GVA, the CPU needs to look for the EPT five times, and each time, the memory needs to be accessed four times. Therefore, there are 20 memory accesses in the worst case, which is still very slow. To overcome this shortcoming, Intel increased the size of the EPT TLB to decrease the number of memory accesses.

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.

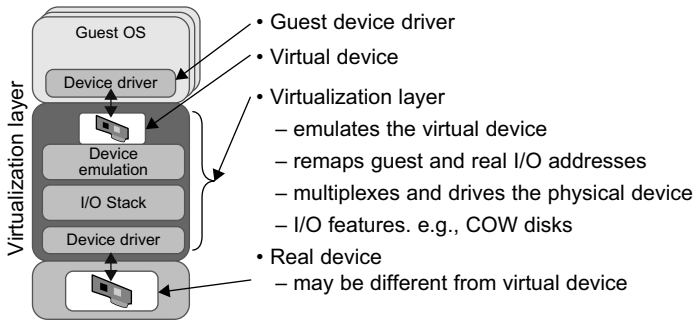


FIGURE 3.14

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

(*Courtesy of V. Chadha, et al. [10] and Y. Dong, et al. [15]*)

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 full device emulation approach is shown in Figure 3.14.

A single hardware device can be shared by multiple VMs that run concurrently. However, software emulation runs much slower than the hardware it emulates [10,15]. The para-virtualization method of I/O virtualization is typically used in Xen. It is also known as the split driver model consisting of a frontend driver and a backend driver. The frontend driver is running in Domain U and the backend driver is running in Domain 0. They interact with each other via a block of shared memory. The frontend driver manages the I/O requests of the guest OSes and the backend driver is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs. Although para-I/O-virtualization achieves better device performance than full device emulation, it comes with a higher CPU overhead.

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. For example, when a physical device is reclaimed (required by workload migration) for later reassignment, it may have been set to an arbitrary state (e.g., DMA to some arbitrary memory locations) that can function incorrectly or even crash the whole system. Since software-based I/O virtualization requires a very high overhead of device emulation, hardware-assisted I/O virtualization is critical. Intel VT-d supports the remapping of I/O DMA transfers and device-generated interrupts. The architecture of VT-d provides the flexibility to support multiple usage models that may run unmodified, special-purpose, or "virtualization-aware" guest OSes.

Another way to help I/O virtualization is via self-virtualized I/O (SV-IO) [47]. The key idea of SV-IO is to harness the rich resources of a multicore processor. All tasks associated with virtualizing an I/O device are encapsulated in SV-IO. It provides virtual devices and an associated access API to VMs and a management API to the VMM. SV-IO defines one virtual interface (VIF) for every kind of virtualized I/O device, such as virtual network interfaces, virtual block devices (disk), virtual camera devices,

and others. The guest OS interacts with the VIFs via VIF device drivers. Each VIF consists of two message queues. One is for outgoing messages to the devices and the other is for incoming messages from the devices. In addition, each VIF has a unique ID for identifying it in SV-IO.

Example 3.7 VMware Workstation for I/O Virtualization

The VMware Workstation runs as an application. It leverages the I/O device support in guest OSes, host OSes, and VMM to implement I/O virtualization. The application portion (VMApp) uses a driver loaded into the host operating system (VMDriver) to establish the privileged VMM, which runs directly on the hardware. A given physical processor is executed in either the host world or the VMM world, with the VMDriver facilitating the transfer of control between the two worlds. The VMware Workstation employs full device emulation to implement I/O virtualization. Figure 3.15 shows the functional blocks used in sending and receiving packets via the emulated virtual NIC.

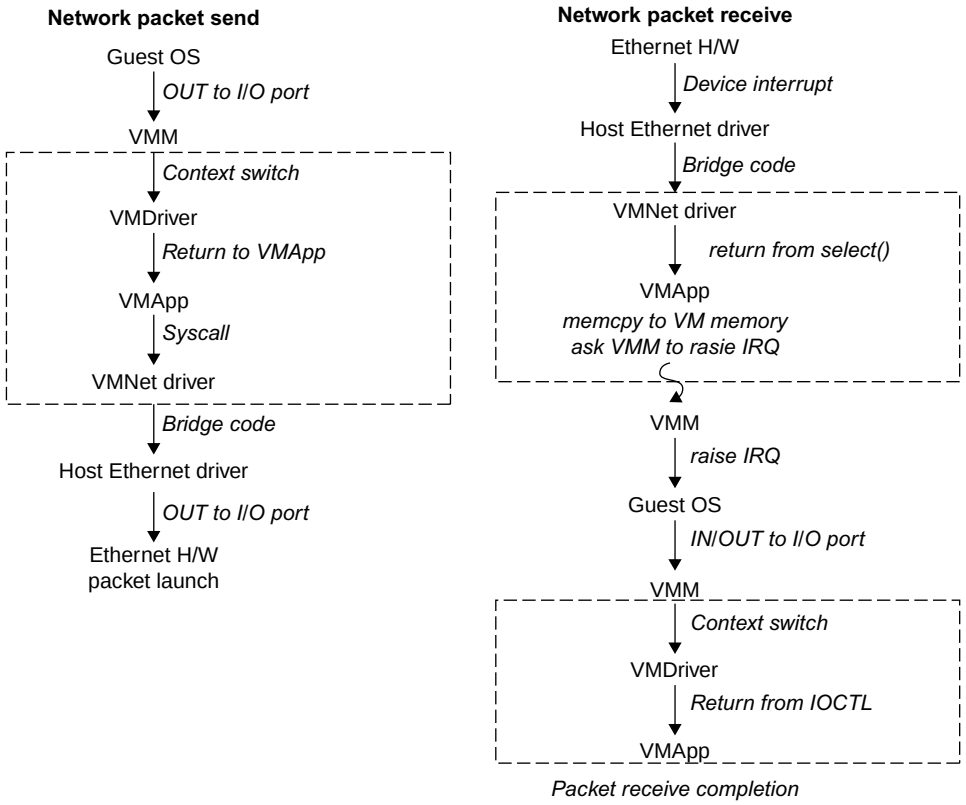


FIGURE 3.15

Functional blocks involved in sending and receiving network packets.

(*Courtesy of VMWare [71]*)

The virtual NIC models an AMD Lance Am79C970A controller. The device driver for a Lance controller in the guest OS initiates packet transmissions by reading and writing a sequence of virtual I/O ports; each read or write switches back to the VMApp to emulate the Lance port accesses. When the last OUT instruction of the sequence is encountered, the Lance emulator calls a normal *write()* to the VMNet driver. The VMNet driver then passes the packet onto the network via a host NIC and then the VMApp switches back to the VMM. The switch raises a virtual interrupt to notify the guest device driver that the packet was sent. Packet receives occur in reverse.

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. Yet these efforts cannot balance well among performance, complexity, and other issues. What is worse, as technology scales, a new 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 [33,66].

3.3.5.1 Physical versus Virtual Processor Cores

Wells, et al. [74] proposed a multicore virtualization method 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). Figure 3.16 illustrates the technique of a software-visible VCPU moving from one core to another and temporarily suspending execution of a VCPU when there are no appropriate cores on which it can run.

3.3.5.2 Virtual Hierarchy

The emerging many-core *chip multiprocessors* (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. This idea was originally suggested by Marty and Hill [39]. 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.

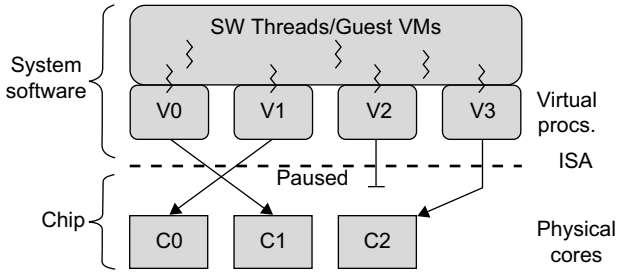


FIGURE 3.16

Multicore virtualization method that exposes four VCPUs to the software, when only three cores are actually present.

(*Courtesy of Wells, et al. [74]*)

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 [39]. 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.

The idea is illustrated in Figure 3.17(a). Space sharing is applied to assign three workloads to three clusters of virtual cores: namely VM0 and VM3 for database workload, VM1 and VM2 for web server workload, and VM4–VM7 for middleware workload. The basic assumption is that each workload runs in its own VM. However, space sharing applies equally within a single operating system. Statically distributing the directory among tiles can do much better, provided operating systems or hypervisors carefully map virtual pages to physical frames. Marty and Hill suggested a two-level virtual coherence and caching hierarchy that harmonizes with the assignment of tiles to the virtual clusters of VMs.

Figure 3.17(b) illustrates a logical view of such a virtual cluster hierarchy in two levels. Each VM operates in a isolated fashion at the first level. This will minimize both miss access time and performance interference with other workloads or VMs. Moreover, the shared resources of cache capacity, inter-connect links, and miss handling are mostly isolated between VMs. The second level maintains a globally shared memory. This facilitates dynamically repartitioning resources without costly cache flushes. Furthermore, maintaining globally shared memory minimizes changes to existing system software and allows virtualization features such as content-based page sharing. A virtual hierarchy adapts to space-shared workloads like multiprogramming and server consolidation. Figure 3.17 shows a case study focused on consolidated server workloads in a tiled architecture. This many-core mapping scheme can also optimize for space-shared multiprogrammed workloads in a single-OS environment.

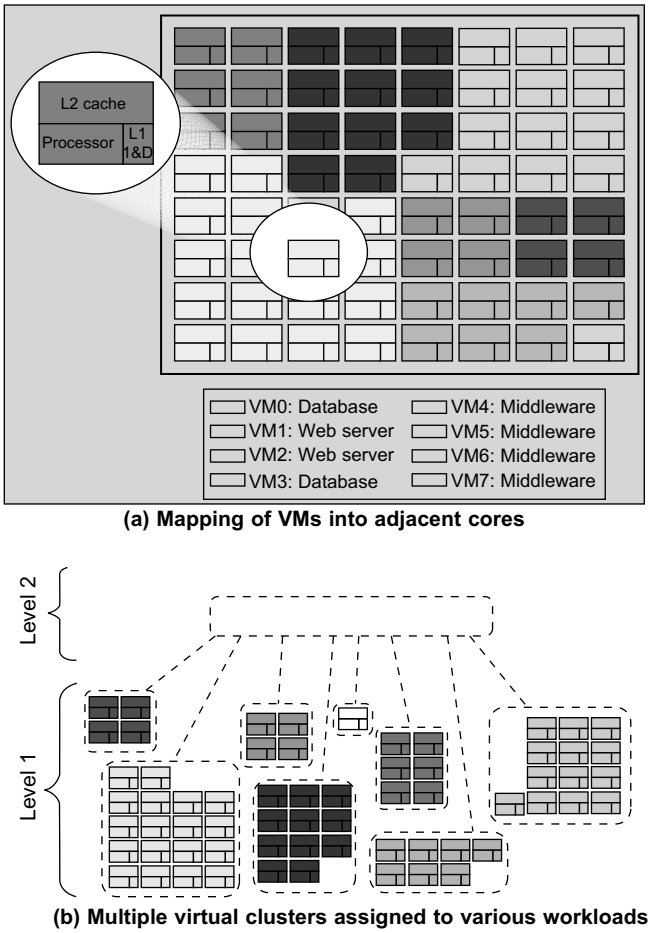


FIGURE 3.17

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.

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. Amazon's *Elastic Compute Cloud (EC2)* is a good example of a web service that provides elastic computing power in a cloud. EC2 permits customers to create VMs and to manage user accounts over the time of their use. Most virtualization platforms, including XenServer 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 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 OSes 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.

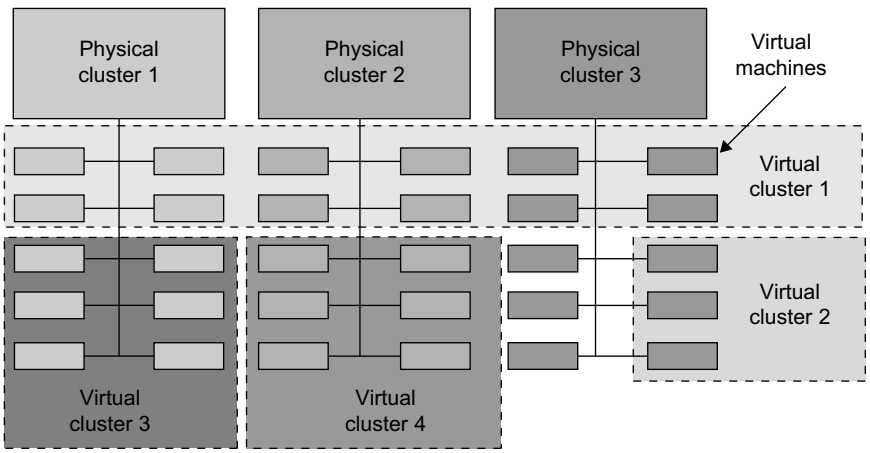


FIGURE 3.18

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

(*Courtesy of Fan Zhang, Tsinghua University*)

- VMs can be colonized (replicated) in multiple servers for the purpose of promoting distributed parallelism, 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.

Figure 3.19 shows the concept of a virtual cluster based on application partitioning or customization. The different colors in the figure represent the nodes in different virtual clusters. As a large number of VM images might be present, the most important thing is to determine how to store those images in the system efficiently. There are common installations for most users or applications, such as operating systems or user-level programming libraries. These software packages can be preinstalled as templates (called template VMs). With these templates, users can build their own software stacks. New OS instances can be copied from the template VM. User-specific components such as programming libraries and applications can be installed to those instances.

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

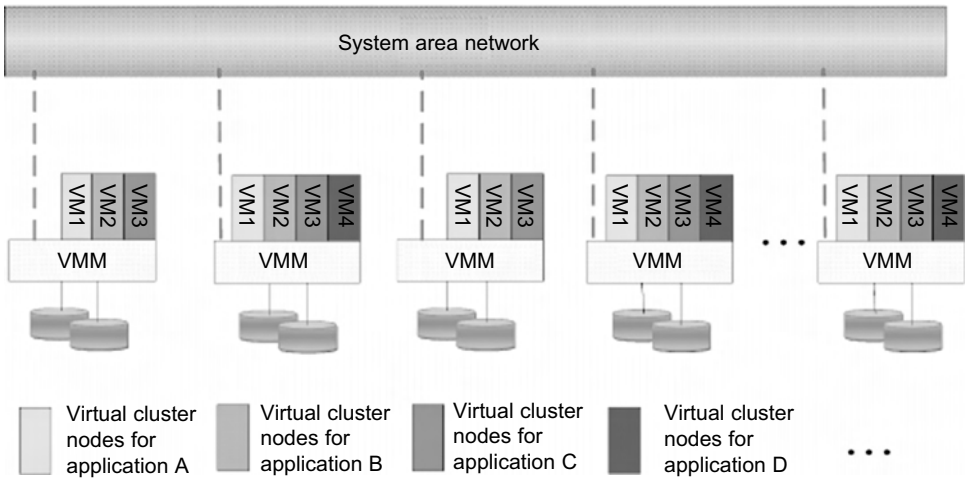


FIGURE 3.19

The concept of a virtual cluster based on application partitioning.

(*Courtesy of Kang Chen, Tsinghua University 2008*)

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 concept of "green computing" has attracted much attention recently. However, previous approaches have focused on saving the energy cost of components in a single workstation without a global vision. Consequently, they do not necessarily reduce the power consumption of the whole cluster. Other cluster-wide energy-efficient techniques can only be applied to homogeneous workstations and specific applications. 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. Therefore, the challenge is to determine how to design migration strategies to implement green computing without influencing the performance of clusters. 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. The automatic scale-up and scale-down mechanism of a virtual cluster can be implemented based on this model. Consequently, we can increase the resource utilization of nodes and shorten the response time of systems. Mapping VMs onto the most appropriate physical node should promote performance. 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 is 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 preedited 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 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. For example, openMosix is an open source Linux cluster running different guest systems on top of the Xen hypervisor. Another example is Sun's cluster Oasis, an experimental Solaris cluster of VMs supported by a VMware VMM. 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 good example is the VMware HA system that can restart a guest system after failure.

These two cluster management systems are either guest-only or host-only, but they do not mix. 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. Various cluster management schemes can be greatly enhanced when VM life migration is enabled with minimal overhead.

VMs can be live-migrated from one physical machine to another; in case of failure, one VM can be replaced by another VM. Virtual clusters can be applied in computational grids, cloud platforms, and high-performance computing (HPC) systems. The major attraction of this scenario is that virtual clustering provides dynamic resources that can be quickly put together upon user demand or after a node failure. In particular, virtual clustering plays a key role in cloud computing. When a VM runs a live service, it is necessary to make a trade-off to ensure that the migration occurs in a manner that minimizes all three metrics. The motivation is to design a live VM migration scheme with negligible downtime, the lowest network bandwidth consumption possible, and a reasonable total migration time.

Furthermore, we should ensure that the migration will not disrupt other active services residing in the same host through resource contention (e.g., CPU, network bandwidth). A VM can be in one of the following four states. An *inactive state* is defined by the virtualization platform, under which

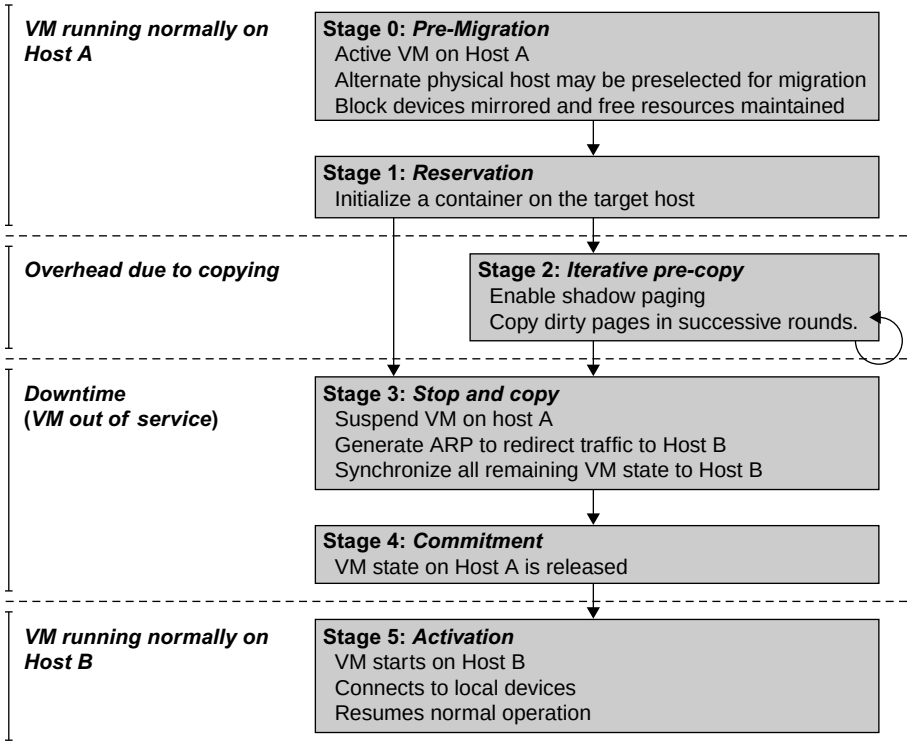


FIGURE 3.20

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

(*Courtesy of C. Clark, et al. [14]*)

the VM is not enabled. An *active state* refers to a VM that has been instantiated at the virtualization platform to perform a real task. A *paused state* corresponds to a VM that has been instantiated but disabled to process a task or paused in a waiting state. A VM enters the *suspended state* if its machine file and virtual resources are stored back to the disk. 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 precopying 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 nonmemory 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.

Figure 3.21 shows the effect on the data transmission rate (Mbit/second) of live migration of a VM from one host to another. Before copying the VM with 512 KB files for 100 clients, the data throughput was 870 MB/second. The first precopy takes 63 seconds, during which the rate is reduced to 765 MB/second. Then the data rate reduces to 694 MB/second in 9.8 seconds for more iterations of the copying process. The system experiences only 165 ms of downtime, before the VM is restored at the destination host. This experimental result shows a very small migration overhead in live transfer of a VM between host nodes. This is critical to achieve dynamic cluster reconfiguration and disaster recovery as needed in cloud computing. We will study these techniques in more detail in Chapter 4.

With the emergence of widespread cluster computing more than a decade ago, many cluster configuration and management systems have been developed to achieve a range of goals. These goals naturally influence individual approaches to cluster management. VM technology has become a popular method for simplifying management and sharing of physical computing resources. Platforms

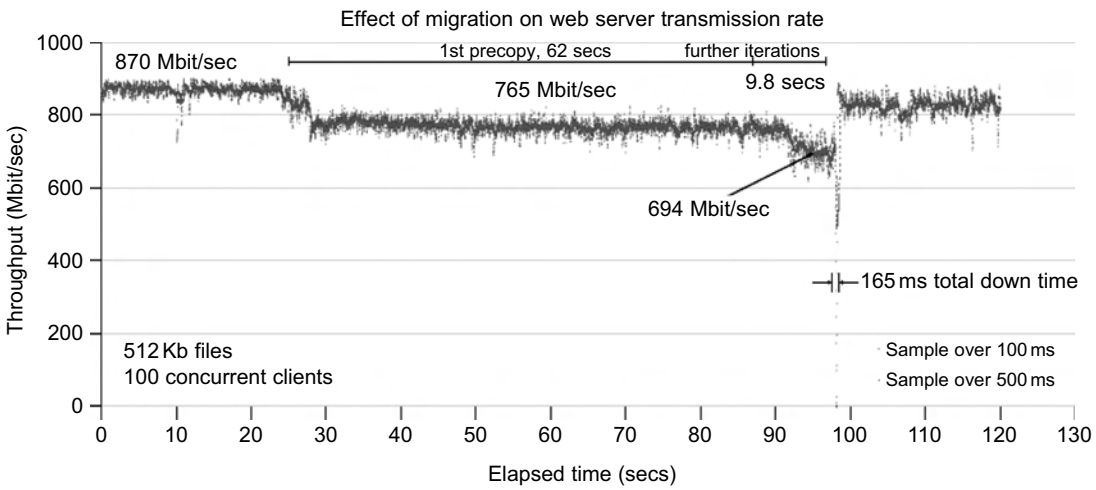


FIGURE 3.21

Effect on data transmission rate of a VM migrated from one failing web server to another.

(*Courtesy of C. Clark, et al. [14]*)

such as VMware and Xen allow multiple VMs with different operating systems and configurations to coexist on the same physical host in mutual isolation. Clustering inexpensive computers is an effective way to obtain reliable, scalable computing power for network services and compute-intensive applications

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. Shared clusters offer economies of scale and more effective utilization of resources by multiplexing. 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

This is one of the most important aspects of VM migration. Moving the memory instance of a VM from one physical host to another can be approached in any number of ways. But traditionally, the concepts behind the techniques tend to share common implementation paradigms. The techniques employed for this purpose depend upon the characteristics of application/workloads supported by the guest OS.

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 subfiles. 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.

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 tolerance. 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 migration 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. When applications' writable working set becomes small, the VM is suspended and only the CPU state and dirty pages in the last round are sent out to the destination.

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. An adaptive rate limiting approach is employed to mitigate this issue, but total migration time is prolonged by nearly 10 times. 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 precopy approach are caused by the large amount of transferred data during the whole migration process. A checkpointing/recovery and trace/replay approach (CR/TR-Motion) is proposed to provide fast VM migration. This approach transfers the execution trace file in iterations rather than dirty pages, which is logged by a trace daemon. Apparently, the total size of all log files is much less than that of dirty pages. So, total migration time and downtime of migration are drastically reduced. However, CR/TR-Motion is valid only when the log replay rate is larger than the log growth rate. The inequality between source and target nodes limits the application scope of live migration in clusters.

Another strategy of postcopy 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 precopy due to the latency of fetching pages from the source node before the VM can be resumed on the target. With the advent of multicore or many-core machines, abundant CPU resources are available. Even if several VMs reside on a same multicore 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.

3.4.3.4 Live Migration of VM Using Xen

In Section 3.2.1, we studied Xen as a VMM or hypervisor, which allows multiple commodity OSes to share x86 hardware in a safe and orderly fashion. The following example explains how to perform live migration of a VM between two Xen-enabled host machines. Domain 0 (or Dom0) performs tasks to create, terminate, or migrate to another host. Xen uses a send/recv model to transfer states across VMs.

Example 3.8 Live Migration of VMs between Two Xen-Enabled Hosts

Xen supports live migration. It is a useful feature and natural extension to virtualization platforms that allows for the transfer of a VM from one physical machine to another with little or no downtime of the services hosted by the VM. Live migration transfers the working state and memory of a VM across a network when it is running. Xen also supports VM migration by using a mechanism called *Remote Direct Memory Access (RDMA)*.

RDMA speeds up VM migration by avoiding TCP/IP stack processing overhead. RDMA implements a different transfer protocol whose origin and destination VM buffers must be registered before any transfer

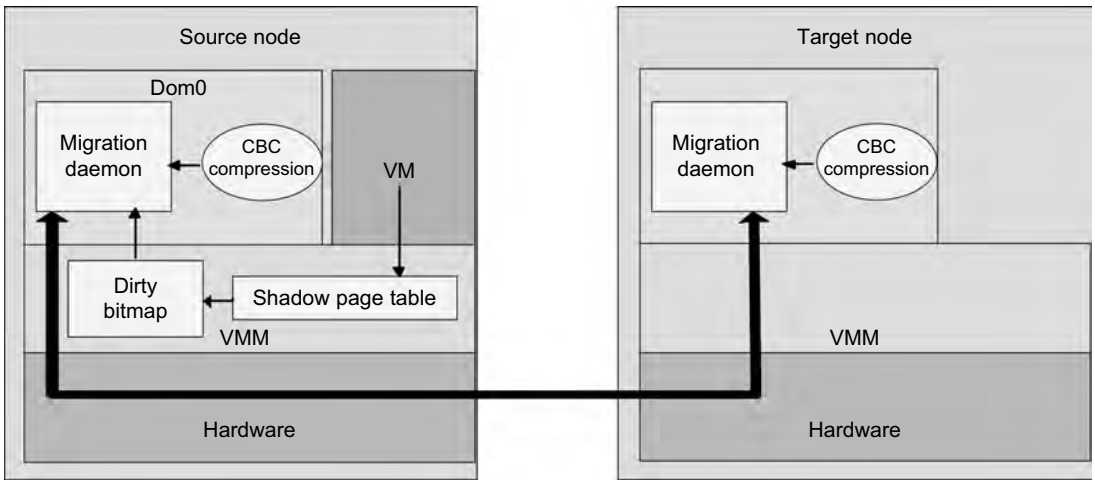


FIGURE 3.22

Live migration of VM from the Dom0 domain to a Xen-enabled target host.

operations occur, reducing it to a "one-sided" interface. Data communication over RDMA does not need to involve the CPU, caches, or context switches. This allows migration to be carried out with minimal impact on guest operating systems and hosted applications. Figure 3.22 shows the a compression scheme for VM migration.

This design requires that we make trade-offs between two factors. If an algorithm embodies expectations about the kinds of regularities in the memory footprint, it must be very fast and effective. A single compression algorithm for all memory data is difficult to achieve the win-win status that we expect. Therefore, it is necessary to provide compression algorithms to pages with different kinds of regularities. The structure of this live migration system is presented in Dom0.

Migration daemons running in the management VMs are responsible for performing migration. Shadow page tables in the VMM layer trace modifications to the memory page in migrated VMs during the precopy phase. Corresponding flags are set in a dirty bitmap. At the start of each precopy round, the bitmap is sent to the migration daemon. Then, the bitmap is cleared and the shadow page tables are destroyed and re-created in the next round. The system resides in Xen's management VM. Memory pages denoted by bitmap are extracted and compressed before they are sent to the destination. The compressed data is then decompressed on the target.

3.4.4 Dynamic Deployment of Virtual Clusters

Table 3.5 summarizes four virtual cluster research projects. We briefly introduce them here just to identify their design objectives and reported results. The Cellular Disco at Stanford is a virtual cluster built in a shared-memory multiprocessor system. The INRIA virtual cluster was built to test parallel algorithm performance. The COD and VIOLIN clusters are studied in forthcoming examples.

Table 3.5 Experimental Results on Four Research Virtual Clusters

Project Name	Design Objectives	Reported Results and References
Cluster-on-Demand at Duke Univ.	Dynamic resource allocation with a virtual cluster management system	Sharing of VMs by multiple virtual clusters using Sun GridEngine [12]
Cellular Disco at Stanford Univ.	To deploy a virtual cluster on a shared-memory multiprocessor	VMs deployed on multiple processors under a VMM called Cellular Disco [8]
VIOLIN at Purdue Univ.	Multiple VM clustering to prove the advantage of dynamic adaptation	Reduce execution time of applications running VIOLIN with adaptation [25,55]
GRAAL Project at INRIA in France	Performance of parallel algorithms in Xen-enabled virtual clusters	75% of max. performance achieved with 30% resource slacks over VM clusters

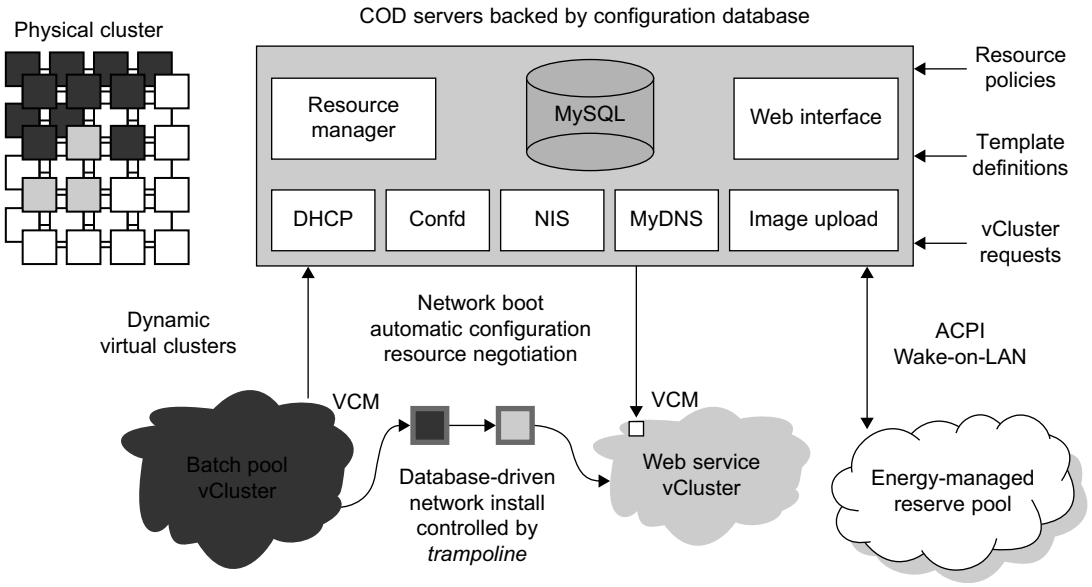


FIGURE 3.23

COD partitioning a physical cluster into multiple virtual clusters.

(*Courtesy of Jeff Chase, et al., HPDC-2003 [12]*)

Example 3.9 The Cluster-on-Demand (COD) Project at Duke University

Developed by researchers at Duke University, the COD (*Cluster-on-Demand*) project is a virtual cluster management system for dynamic allocation of servers from a computing pool to multiple virtual clusters [12]. The idea is illustrated by the prototype implementation of the COD shown in Figure 3.23. The COD

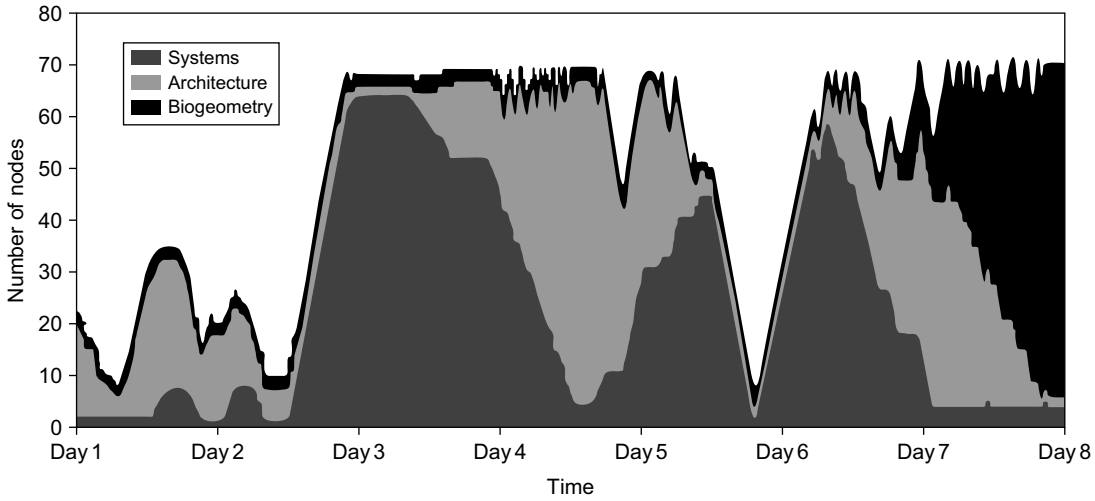


FIGURE 3.24

Cluster size variations in COD over eight days at Duke University.

(*Courtesy of J. Chase, et al. [12]*)

partitions a physical cluster into multiple virtual clusters (*vClusters*). vCluster owners specify the operating systems and software for their clusters through an XML-RPC interface. The vClusters run a batch schedule from Sun's GridEngine on a web server cluster. The COD system can respond to load changes in restructuring the virtual clusters dynamically.

The Duke researchers used the Sun GridEngine scheduler to demonstrate that dynamic virtual clusters are an enabling abstraction for advanced resource management in computing utilities such as grids. The system supports dynamic, policy-based cluster sharing between local users and hosted grid services. Attractive features include resource reservation, adaptive provisioning, scavenging of idle resources, and dynamic instantiation of grid services. The COD servers are backed by a configuration database. This system provides resource policies and template definition in response to user requests.

Figure 3.24 shows the variation in the number of nodes in each of three virtual clusters during eight days of a live deployment. Three application workloads requested by three user groups are labeled "Systems," "Architecture," and "BioGeometry" in the trace plot. The experiments were performed with multiple SGE batch pools on a test bed of 80 rack-mounted IBM xSeries-335 servers within the Duke cluster. This trace plot clearly shows the sharp variation in cluster size (number of nodes) over the eight days. Dynamic provisioning and deprovisioning of virtual clusters are needed in real-life cluster applications.

Example 3.10 The VIOLIN Project at Purdue University

The Purdue VIOLIN Project applies live VM migration to reconfigure a virtual cluster environment. Its purpose is to achieve better resource utilization in executing multiple cluster jobs on multiple cluster

domains. The project leverages the maturity of VM migration and environment adaptation technology. The approach is to enable mutually isolated virtual environments for executing parallel applications on top of a shared physical infrastructure consisting of multiple domains. Figure 3.25 illustrates the idea with five concurrent virtual environments, labeled as VIOLIN 1–5, sharing two physical clusters.

The squares of various shadings represent the VMs deployed in the physical server nodes. The major contribution by the Purdue group is to achieve autonomic adaptation of the virtual computation environments as active, integrated entities. A virtual execution environment is able to relocate itself across the infrastructure, and can scale its share of infrastructural resources. The adaptation is transparent to both users of virtual environments and administrations of infrastructures. The adaptation overhead is maintained at 20 sec out of 1,200 sec in solving a large NEMO3D problem of 1 million particles.

The message being conveyed here is that the virtual environment adaptation can enhance resource utilization significantly at the expense of less than 1 percent of an increase in total execution time. The

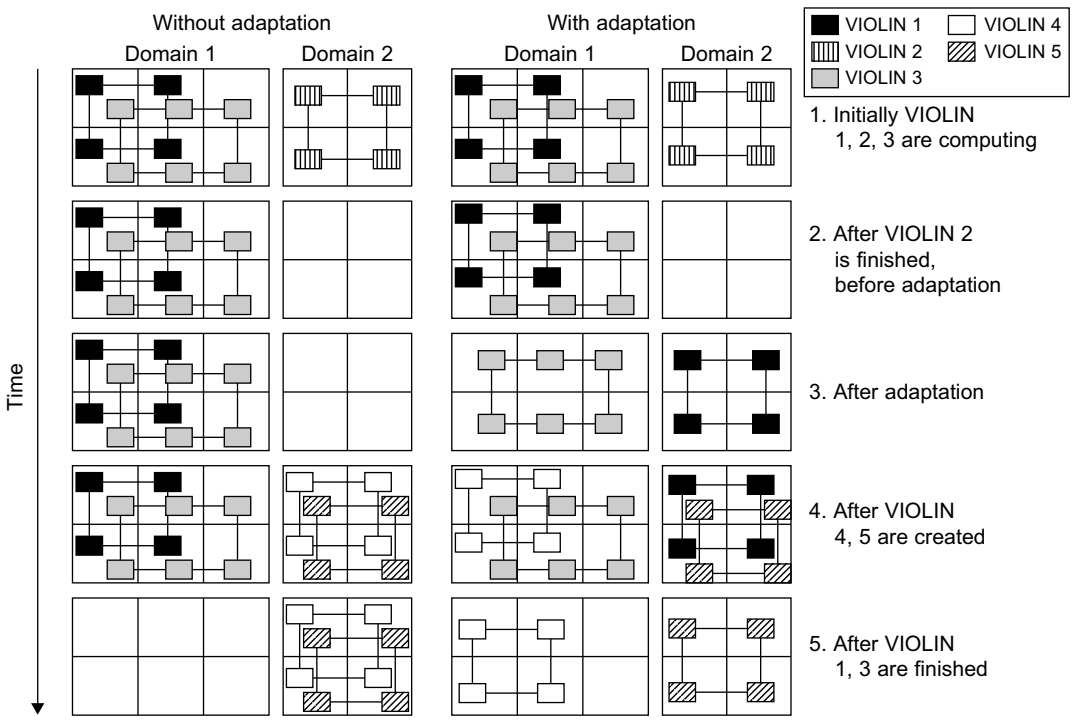


FIGURE 3.25

VIOLIN adaptation scenario of five virtual environments sharing two hosted clusters. Note that there are more idle squares (blank nodes) before and after the adaptation.

(*Courtesy of P. Ruth, et al. [55]*)

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.

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.

Dynamic CPU allocation is based on VM utilization and application-level QoS metrics. One method considers both CPU and memory flowing as well as automatically adjusting resource overhead based on varying workloads in hosted services. Another scheme uses a two-level resource management system to handle the complexity involved. A local controller at the VM level and a global controller at the server level are designed. They implement autonomic resource allocation via the interaction of the local and global controllers. Multicore and virtualization are two cutting techniques that can enhance each other.

However, the use of CMP is far from well optimized. The memory system of CMP is a typical example. One can design a virtual hierarchy on a CMP in data centers. One can consider protocols that minimize the memory access time, inter-VM interferences, facilitating VM reassignment, and supporting inter-VM sharing. One can also consider a VM-aware power budgeting scheme using multiple managers integrated to achieve better power management. The power budgeting policies cannot ignore the heterogeneity problems. Consequently, one must address the trade-off of power saving and data-center performance.

3.5.2 Virtual Storage Management

The term "storage virtualization" was widely used before the renaissance 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 a real hard disk while the guest OSes 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).

In addition, the storage primitives used by VMs are not nimble. Hence, operations such as remapping volumes across hosts and checkpointing disks are frequently clumsy and esoteric, and sometimes simply unavailable. In data centers, there are often thousands of VMs, which cause the VM images to become flooded. Many researchers tried to solve these problems in virtual storage management. The main purposes of their research are to make management easy while enhancing performance and reducing the amount of storage occupied by the VM images. Parallax is a distributed storage system customized for virtualization environments. Content Addressable Storage (CAS) is a solution to reduce the total size of VM images, and therefore supports a large set of VM-based systems in data centers.

Since traditional storage management techniques do not consider the features of storage in virtualization environments, Parallax designs a novel architecture in which storage features that have traditionally been implemented directly on high-end storage arrays and switchers are relocated into a federation of storage VMs. These storage VMs share the same physical hosts as the VMs that they serve. Figure 3.30 provides an overview of the Parallax system architecture. It supports all popular system virtualization techniques, such as paravirtualization and full virtualization. For each physical machine, Parallax customizes a special storage appliance VM. The storage appliance VM acts as a block virtualization layer between individual VMs and the physical storage device. It provides a virtual disk for each VM on the same physical machine.

Example 3.11 Parallax Providing Virtual Disks to Client VMs from a Large Common Shared Physical Disk

The architecture of Parallax is scalable and especially suitable for use in cluster-based environments. Figure 3.26 shows a high-level view of the structure of a Parallax-based cluster. A cluster-wide administrative domain manages all storage appliance VMs, which makes storage management easy. The storage appliance

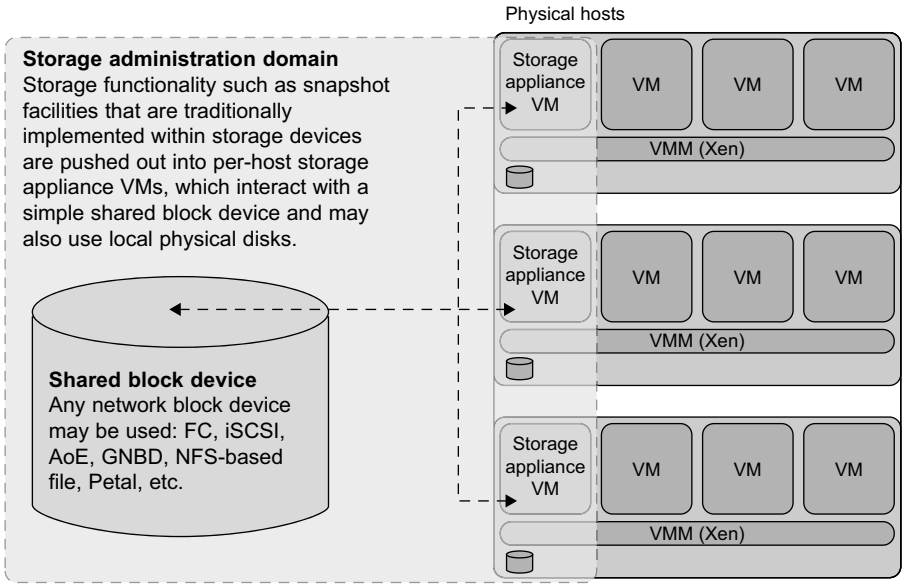


FIGURE 3.26

Parallax is a set of per-host storage appliances that share access to a common block device and presents virtual disks to client VMs.

(*Courtesy of D. Meyer, et al. [43]*)

VM also allows functionality that is currently implemented within data-center hardware to be pushed out and implemented on individual hosts. This mechanism enables advanced storage features such as snapshot facilities to be implemented in software and delivered above commodity network storage targets.

Parallax itself runs as a user-level application in the storage appliance VM. It provides *virtual disk images (VDIs)* to VMs. A VDI is a single-writer virtual disk which may be accessed in a location-transparent manner from any of the physical hosts in the Parallax cluster. The VDIs are the core abstraction provided by Parallax. Parallax uses Xen's block tap driver to handle block requests and it is implemented as a tapdisk library. This library acts as a single block virtualization service for all client VMs on the same physical host. In the Parallax system, it is the storage appliance VM that connects the physical hardware device for block and network access. As shown in Figure 3.30, physical device drivers are included in the storage appliance VM. This implementation enables a storage administrator to live-upgrade the block device drivers in an active cluster.

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,

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

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.

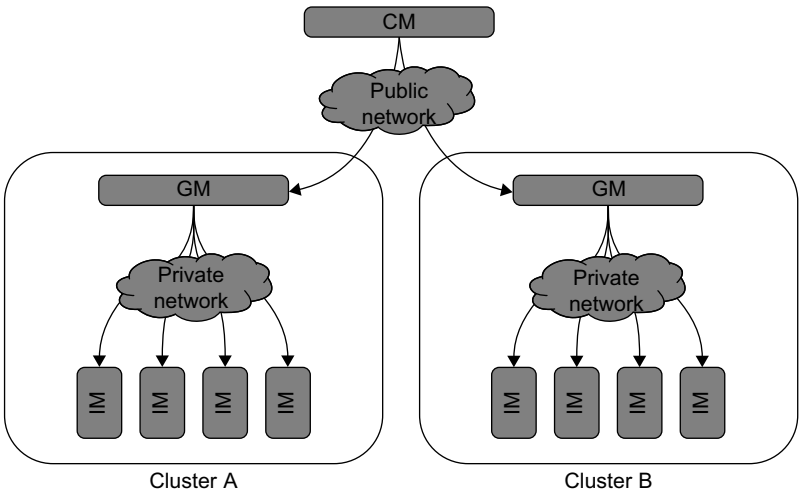


FIGURE 3.27

Eucalyptus for building private clouds by establishing virtual networks over the VMs linking through Ethernet and the Internet.

(*Courtesy of D. Nurmi, et al. [45]*)

Furthermore, the designers leverage existing web-service features such as WS-Security policies for secure communication between components. The three resource managers in Figure 3.27 are specified below:

- **Instance Manager** controls the execution, inspection, and terminating of VM instances on the host where it runs.
- **Group Manager** gathers information about and schedules VM execution on specific instance managers, as well as manages virtual instance network.
- **Cloud Manager** is the entry-point into the cloud for users and administrators. It queries node managers for information about resources, makes scheduling decisions, and implements them by making requests to group managers.

In terms of functionality, Eucalyptus works like AWS APIs. Therefore, it can interact with EC2. It does provide a storage API to emulate the Amazon S3 API for storing user data and VM images. It is installed on Linux-based platforms, is compatible with EC2 with SOAP and Query, and is S3-compatible with SOAP and REST. CLI and web portal services can be applied with Eucalyptus.

Example 3.13 VMware vSphere 4 as a Commercial Cloud OS [66]

The vSphere 4 offers a hardware and software ecosystem developed by VMware and released in April 2009. vSphere extends earlier virtualization software products by VMware, namely the VMware Workstation, ESX for server virtualization, and Virtual Infrastructure for server clusters. Figure 3.28 shows vSphere's

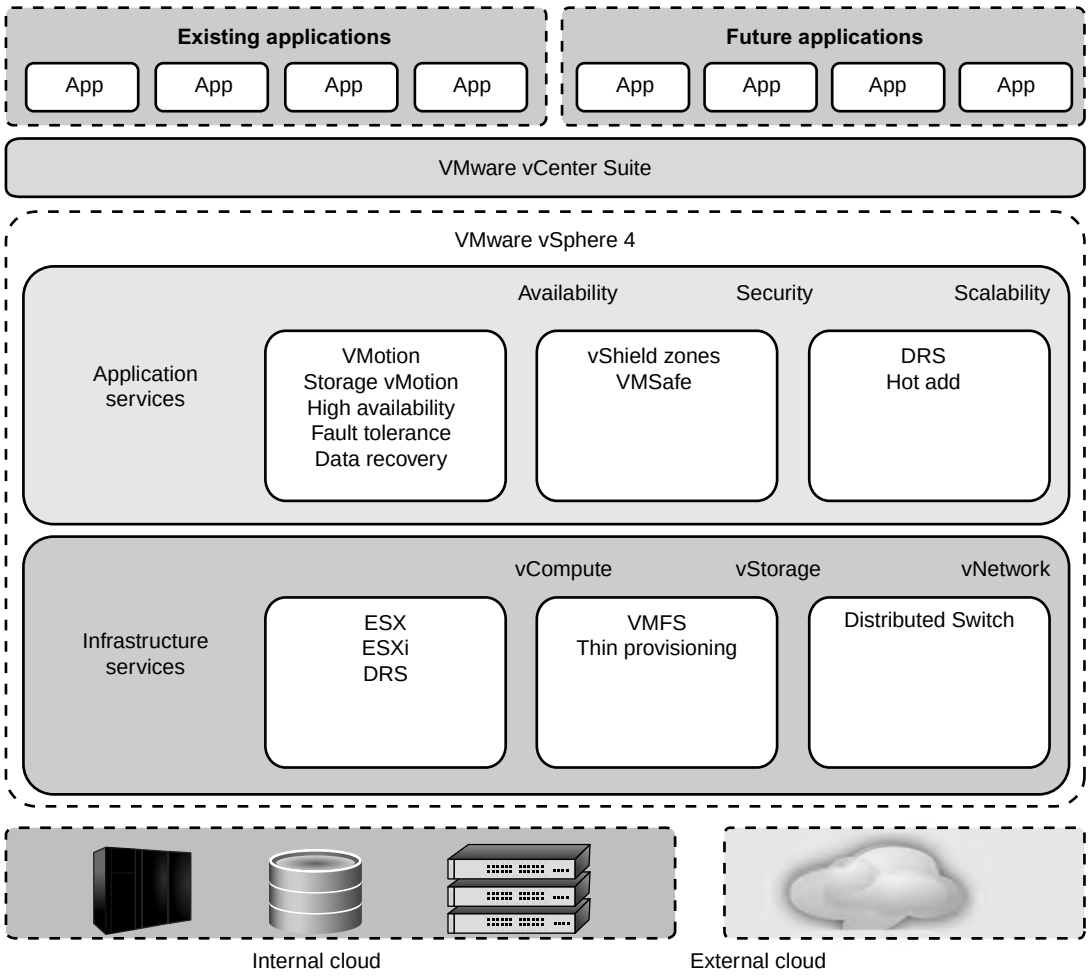


FIGURE 3.28

vSphere/4, a cloud operating system that manages compute, storage, and network resources over virtualized data centers.

(*Courtesy of VMware, April 2010 [72]*)

overall architecture. The system interacts with user applications via an interface layer, called *vCenter*. vSphere is primarily intended to offer virtualization support and resource management of data-center resources in building private clouds. VMware claims the system is the first cloud OS that supports availability, security, and scalability in providing cloud computing services.

The vSphere 4 is built with two functional software suites: *infrastructure services* and *application services*. It also has three component packages intended mainly for virtualization purposes: *vCompute* is supported by ESX, ESXi, and DRS virtualization libraries from VMware; *vStorage* is supported by VMS and

thin provisioning libraries; and *vNetwork* offers distributed switching and networking functions. These packages interact with the hardware servers, disks, and networks in the data center. These infrastructure functions also communicate with other external clouds.

The application services are also divided into three groups: *availability*, *security*, and *scalability*. Availability support includes VMotion, Storage VMotion, HA, Fault Tolerance, and Data Recovery from VMware. The security package supports vShield Zones and VMsafe. The scalability package was built with DRS and Hot Add. Interested readers should refer to the vSphere 4 web site for more details regarding these component software functions. To fully understand the use of vSphere 4, users must also learn how to use the vCenter interfaces in order to link with existing applications or to develop new applications.

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. Noncryptographic 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

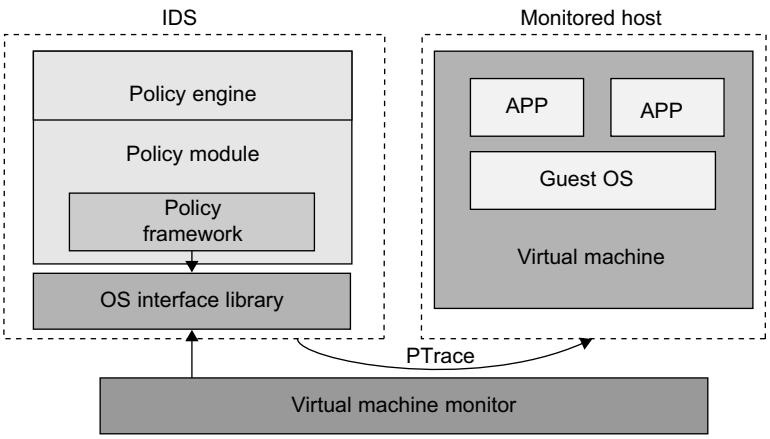


FIGURE 3.29

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

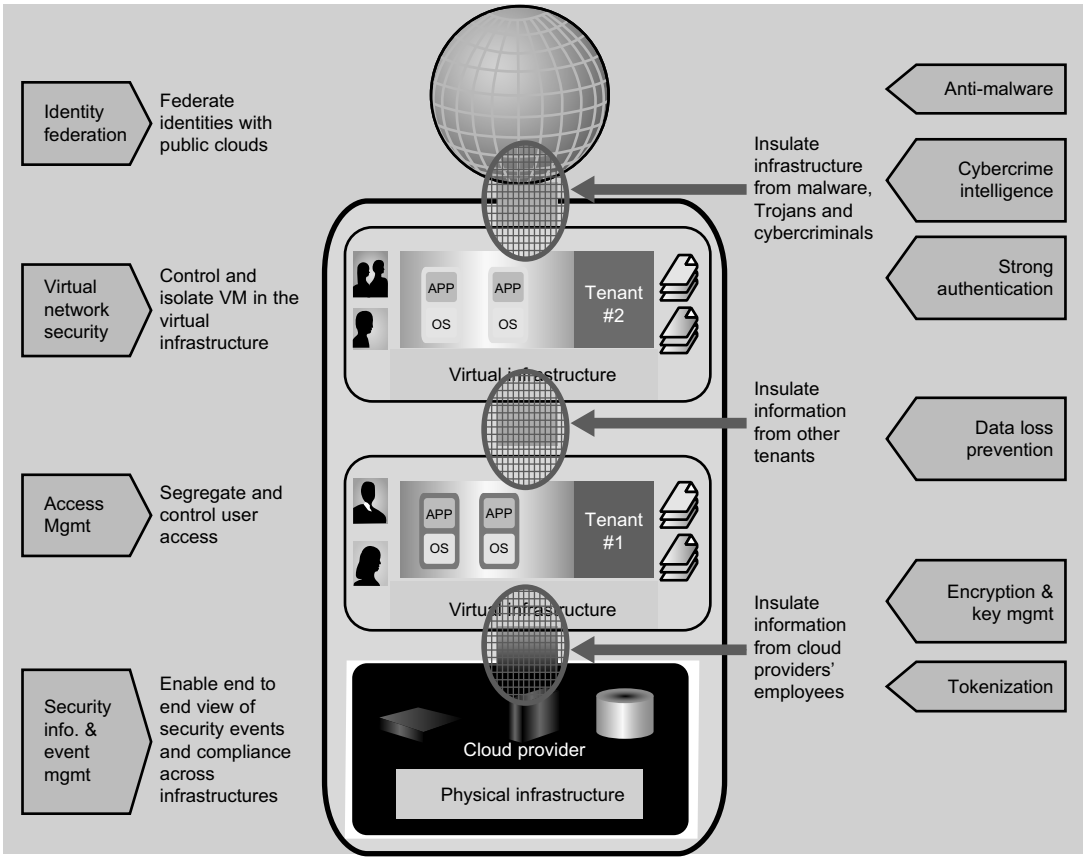


FIGURE 3.30

Techniques for establishing trusted zones for virtual cluster insulation and VM isolation.

(*Courtesy of L. Nick, EMC [40]*)

applications and OSes for each tenant) provisioned in separate virtual environments. The physical infrastructure is shown at the bottom, and marked as a cloud provider. The virtual clusters or infrastructures are shown in the upper boxes for two tenants. The public cloud is associated with the global user communities at the top.

The arrowed boxes on the left and the brief description between the arrows and the zoning boxes are security functions and actions taken at the four levels from the users to the providers. The small circles between the four boxes refer to interactions between users and providers and among the users themselves. The arrowed boxes on the right are those functions and actions applied between the tenant environments, the provider, and the global communities.

Almost all available countermeasures, such as anti-virus, worm containment, intrusion detection, encryption and decryption mechanisms, are applied here to insulate the trusted zones and isolate the VMs for private tenants. The main innovation here is to establish the trust zones among the virtual clusters.

The end result is to enable an end-to-end view of security events and compliance across the virtual clusters dedicated to different tenants. We will discuss security and trust issues in Chapter 7 when we study clouds in more detail.

3.6 BIBLIOGRAPHIC NOTES AND HOMEWORK PROBLEMS

Good reviews on virtualization technology can be found in Rosenblum, et al. [53,54] and in Smith and Nair [58,59]. White papers at [71,72] report on virtualization products at VMware, including the vSphere 4 cloud operating system. The Xen hypervisor was introduced in [7,13,42] and KVM in [31]. Qian, et al. [50] have given some theoretical treatment of the subject. ISA-level virtualization and binary translation techniques are treated in [3] and in Smith and Nair [58]. Some comparison of virtualizing software environments can be found in Buyya, et al. [9]. Hardware-level virtualization is treated in [1,7,8,13,37,43,54,67]. Intel's support of hardware-level virtualization is treated in [62]. Some entries in Table 3.5 are taken from Buyya, et al. [9].

For GPU computing on VMs, the readers are referred to [57]. The x86 host virtualization is treated in [2]. Pentium virtualization is treated in [53]. treated in [11,24,26,52]. I/O virtualization is

Sun Microsystems reports on OS-level virtualization in [64]. OpenVZ is introduced in its user's guide [65]. Virtualization in Windows NT machines is described in [77,78]. For GPU computing on VMs, readers are referred to [53]. The x86 host virtualization is treated in [2]. Pentium virtualization is treated in [53]. The book [24] have a good coverage of hardware support for virtualization. Virtual clusters are treated in [8,12,20,25,49,55,56]. In particular, Duke's COD is reported in [12] and Purdue's Violin in [25,55]. Memory virtualization is treated in [1,10,13,51,58,68].

Hardware-level virtualization is treated in [1,7,8,13,41,47,58,73]. Intel's support of hardware-level virtualization is treated in [68]. Wells, et al. [74] have studied multi-core virtualization. The integration of multi-core and virtualization on future CPU/GPU chips posts a very hot research area, called asymmetric CMP as reported in [33,39,63,66,74]. Architectural supports for virtualized chip multiprocessors have been also studied in [17,28,30,39,66]. The maturity of this co-design CMP approach will greatly impact the future development of HPC and HTC systems.

Server consolidation in virtualized data centers is treated in [29,39,75]. Power consumption in virtualized data centers is treated in [44,46]. Literatures [46,60,61,62] discussed the virtual resource management in data centers. Kochut gives an analytical model for virtualized data centers [32]. Security protection and trust management for virtualized data centers are treated in [18,45]. For virtual storage, readers are referred to the literature [27,36,43,48,76,79]. Specific references for the examples are identified in the figure captions or in the text description of the example. The Eucalyptus was reported in [45], vSphere in [72], Parallax in [43]. The vCUDA was reported in [57] for CUDA programming on the VMs.

Acknowledgments

This chapter is coauthored by Zhibin Yu of Huazhong University of Science and Technology (HUST), China and by Kai Hwang of USC. Dr. Hai Jin and Dr. Xiaofei Liao of HUST have extended significant technical support to this work. The authors would like to thank Dr. Zhong-Yuan Qin

of Southeast University, China, Fan Zhang of Tsinghua University, and Lizhong Chen and Zhou Zhao of USC for their assistance in drawing several sketched figures and updating the references in this chapter.

References

[1] Advanced Micro Devices. AMD Secure Virtual Machine Architecture Reference Manual, 2008.
[2] K. Adams, O. Agesen, A comparison of software and hardware techniques for x86 virtualization, in: Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, October 2006, pp. 21–25.
[3] V. Adve, C. Lattner, et al., LLVA: A low-level virtual instruction set architecture, in: Proceedings of the 36th International Symposium on Micro-architecture (MICRO-36 '03), 2003.
[4] J. Alonso, L. Silva, A. Andrzejak, P. Silva, J. Torres, High-available grid services through the use of virtualized clustering, in: Proceedings of the 8th Grid Computing Conference, 2007.
[5] P. Anedda, M. Gaggero, et al., A general service-oriented approach for managing virtual machine allocation, in: Proceedings of the 24th Annual ACM Symposium on Applied Computing (SAC 2009), ACM Press, March 2009, pp. 9–12.
[6] H. Andre Lagar-Cavilla, J.A. Whitney, A. Scannell, et al., SnowFlock: rapid virtual machine cloning for cloud computing, in: Proceedings of EuroSystems, 2009.
[7] P. Barham, B. Dragovic, K. Fraser, et al., Xen and the art of virtualization, in: Proceedings of the 19th ACM Symposium on Operating System Principles (SOSP19), ACM Press, 2003, pp. 164–177.
[8] E. Bugnion, S. Devine, M. Rosenblum, Disco: running commodity OS on scalable multiprocessors, in: Proceedings of SOSP, 1997.
[9] R. Buyya, J. Broberg, A. Goscinski (Eds.), Cloud Computing: Principles and Paradigms, Wiley Press, New York, 2011.
[10] V. Chadha, R. Illikkal, R. Iyer, I/O Processing in a virtualized platform: a simulation-driven approach, in: Proceedings of the 3rd International Conference on Virtual Execution Environments (VEE), 2007.
[11] R. Chandra, N. Zeldovich, C. Sapuntzakis, M.S. Lam, The collective: a cache-based system management architecture, in: Proceedings of the Second Symposium on Networked Systems Design and Implementation (NSDI '05), USENIX, Boston, May 2005, pp. 259–272.
[12] J. Chase, L. Grit, D. Irwin, J. Moore, S. Sprenkle, Dynamic virtual cluster in a grid site manager, in: IEEE Int'l Symp. on High Performance Distributed Computing, (HPDC-12), 2003.
[13] D. Chisnall, The Definitive Guide to the Xen Hypervisor, Prentice Hall, International, 2007.
[14] C. Clark, K. Fraser, S. Hand, et al., Live migration of virtual machines, in: Proceedings of the Second Symposium on Networked Systems Design and Implementation (NSDI '05), 2005, pp. 273–286.
[15] Y. Dong, J. Dai, et al., Towards high-quality I/O virtualization, in: Proceedings of SYSTOR 2009, The Israeli Experimental Systems Conference, 2009.
[16] E. Elnozahy, M. Kistler, R. Rajamony, Energy-efficient server clusters, in: Proceedings of the 2nd Workshop on Power-Aware Computing Systems, February 2002.
[17] J. Frich, et al., On the potential of NoC virtualization for multicore chips, in: IEEE Int'l Conf. on Complex, Intelligent and Software-Intensive Systems, 2008, pp. 801–807.
[18] T. Garfinkel, M. Rosenblum, A virtual machine introspection-based architecture for intrusion detection, 2002.
[19] L. Grit, D. Irwin, A. Yumerefendi, J. Chase, Virtual machine hosting for networked clusters: building the foundations for autonomic orchestration, in: First International Workshop on Virtualization Technology in Distributed Computing (VTDC), November 2006.

[20] D. Gupta, S. Lee, M. Vrable, et al., Difference engine: Harnessing memory redundancy in virtual machines, in: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI '08), 2008, pp. 309–322.
[21] M. Hines, K. Gopalan, Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning, in: Proceedings of the ACM/USENIX International Conference on Virtual Execution Environments (VEE '09), 2009, pp. 51–60.
[22] T. Hirofuchi, H. Nakada, et al., A live storage migration mechanism over WAN and its performance evaluation, in: Proceedings of the 4th International Workshop on Virtualization Technologies in Distributed Computing, 15 June, ACM Press, Barcelona, Spain, 2009.
[23] K. Hwang, D. Li, Trusted cloud computing with secure resources and data coloring, IEEE Internet Comput., (September/October) (2010) 30–39.
[24] Intel Open Source Technology Center, System Virtualization—Principles and Implementation, Tsinghua University Press, Beijing, China, 2009.
[25] X. Jiang, D. Xu, VIOLIN: Virtual internetworking on overlay infrastructure, in: Proceedings of the International Symposium on Parallel and Distributed Processing and Applications, 2004, pp. 937–946.
[26] H. Jin, L. Deng, S. Wu, X. Shi, X. Pan, Live virtual machine migration with adaptive memory compression, in: Proceedings of the IEEE International Conference on Cluster Computing, 2009.
[27] K. Jin, E. Miller, The effectiveness of deduplication on virtual machine disk images, in: Proceedings of SYSTOR, 2009, The Israeli Experimental Systems Conference, 2009.
[28] S. Jones, A. Arpaci-Disseau, R. Arpaci-Disseau, Geiger: Monitoring the buffer cache in a virtual machine environment, in: ACM ASPLOS, San Jose, CA, October 2006, pp. 13–14.
[29] F. Kamoun, Virtualizing the datacenter without compromising server performance, ACM Ubiquity 2009, (9) (2009).
[30] D. Kim, H. Kim, J. Huh, Virtual snooping: Filtering snoops in virtualized multi-coures, in: 43rd Annual IEEE/ACM Int'l Symposium on Mcrosrchitecture (MICRO-43).
[31] A. Kivity, et al., KVM: The linux virtual machine monitor, in: Proceedings of the Linux Symposium, Ottawa, Canada, 2007, p. 225.
[32] A. Kochut, On impact of dynamic virtual machine reallocation on data center efficiency, in: Proceedings of the IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems (MASCOTS), 2008.
[33] R. Kumar, et al., Heterogeneous chip multiptiprocessors, IEEE Comput. Mag. 38 (November) (2005) 32–38.
[34] B. Kyrre, Managing large networks of virtual machines, in: Proceedings of the 20th Large Installation System Administration Conference, 2006, pp. 205–214.
[35] J. Lange, P. Dinda, Transparent network services via a virtual traffic layer for virtual machines, in: Proceedings of High Performance Distributed Computing, ACM Press, Monterey, CA, pp. 25–29, June 2007.
[36] A. Liguori, E. Hensbergen, Experiences with content addressable storage and virtual disks, in: Proceedings of the Workshop on I/O Virtualization (WIOV '08), 2008.
[37] H. Liu, H. Jin, X. Liao, L. Hu, C. Yu, Live migration of virtual machine based on full system trace and replay, in: Proceedings of the 18th International Symposium on High Performance Distributed Computing (HPDC '09), 2009, pp. 101–110.
[38] A. Mainwaring, D. Culler, Design challenges of virtual networks: Fast, general-purpose communication, in: Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, 1999.
[39] M. Marty, M. Hill, Virtual hierarchies to support server consolidation, in: Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA), 2007.

[40] M. McNett, D. Gupta, A. Vahdat, G.M. Voelker, Usher: An extensible framework for managing clusters of virtual machines, in: 21st Large Installation System Administration Conference (LISA) 2007.
[41] D. Menasce, Virtualization: Concepts, applications., performance modeling, in: Proceedings of the 31st International Computer Measurement Group Conference, 2005, pp. 407–414.
[42] A. Menon, J. Renato, Y. Turner, Diagnosing performance overheads in the Xen virtual machine environment, in: Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments, 2005.
[43] D. Meyer, et al., Parallax: Virtual disks for virtual machines, in: Proceedings of EuroSys, 2008.
[44] J. Nick, Journey to the private cloud: Security and compliance, in: Technical presentation by EMC Visiting Team, May 25, Tsinghua University, Beijing, 2010.
[45] D. Nurmi, et al., The eucalyptus open-source cloud computing system, in: Proceedings of the 9th IEEE ACM International Symposium on Cluster Computing and The Grid (CCGrid), Shanghai, China, September 2009, pp. 124–131.
[46] P. Padala, et al., Adaptive control of virtualized resources in utility computing environments, in: Proceedings of EuroSys 2007.
[47] L. Peterson, A. Bavier, M.E. Fiuczynski, S. Muir, Experiences Building PlanetLab, in: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI2006), 6–8 November 2006.
[48] B. Pfaff, T. Garfinkel, M. Rosenblum, Virtualization aware file systems: Getting beyond the limitations of virtual disks, in: Proceedings of USENIX Networked Systems Design and Implementation (NSDI 2006), May 2006, pp. 353–366.
[49] E. Pinheiro, R. Bianchini, E. Carrera, T. Heath, Dynamic cluster reconfiguration for power and performance, in: L. Benini (Ed.), Compilers and Operating Systems for Low Power, Kluwer Academic Publishers, 2003.
[50] H. Qian, E. Miller, et al., Agility in virtualized utility computing, in: Proceedings of the Third International Workshop on Virtualization Technology in Distributed Computing (VTDC 2007), 12 November 2007.
[51] H. Raj, I. Ganev, K. Schwan, Self-Virtualized I/O: High Performance, Scalable I/O Virtualization in Multi-core Systems, Technical Report GIT-CERCS-06-02, CERCS, Georgia Tech, 2006, www.cercs.gatech.edu/tech-reports/tr2006/git-cercs-06-02.pdf.
[52] J. Robin, C. Irvine, Analysis of the Intel pentium's ability to support a secure virtual machine monitor, in: Proceedings of the 9th USENIX Security Symposium Vol. 9, 2000.
[53] M. Rosenblum, The reincarnation of virtual machines, ACM QUEUE, (July/August) (2004).
[54] M. Rosenblum, T. Garfinkel, Virtual machine monitors: current technology and future trends, IEEE Comput 38 (5) (2005) 39–47.
[55] P. Ruth, et al., Automatic Live Migration of Virtual Computational Environments in a Multi-domain Infrastructure, Purdue University, 2006.
[56] C. Sapuntzakis, R. Chandra, B. Pfaff, et al., Optimizing the migration of virtual computers, in: Proceedings of the 5th Symposium on Operating Systems Design and Implementation, Boston, 9–11 December 2002.
[57] L. Shi, H. Chen, J. Sun, vCUDA: GPU accelerated high performance computing in virtual machines, in: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, 2009.
[58] J. Smith, R. Nair, Virtual Machines: Versatile Platforms for Systems and Processes, Morgan Kaufmann, 2005.
[59] J. Smith, R. Nair, The architecture of virtual machines, IEEE Comput., (May) (2005).
[60] Y. Song, H. Wang, et al., Multi-tiered on-demand resource scheduling for VM-based data center, in: Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009.
[61] B. Sotomayor, K. Keahey, I. Foster, Combining batch execution and leasing using virtual machines, in: Proceedings of the 17th International Symposium on High-Performance Distributed Computing, 2008.

[62] M. Steinder, I. Whalley, et al., Server virtualization in autonomic management of heterogeneous workloads, ACM SIGOPS Oper. Syst. Rev. 42 (1) (2008) 94–95.
[63] M. Suleman, Y. Patt, E. Sprangle, A. Rohillah, Asymmetric chip multiprocessors: balancing hardware efficiency and programming efficiency, (2007).
[64] Sun Microsystems. Solaris Containers: Server Virtualization and Manageability, Technical white paper, September 2004.
[65] SWsoft, Inc. OpenVZ User's Guide, http://ftp.openvz.org/doc/OpenVZ-Users-Guide.pdf, 2005.
[66] F. Trivino, et al., Virtualizing netwoirk on chip resources in chip multiprocessors, J. Microprocess. Microsyst. 35 (2010). 245–230 http://www.elsevier.com/locate/micro.
[67] J. Xu, M. Zhao, et al., On the use of fuzzy modeling in virtualized datacenter management, in: Proceedings of the 4th International Conference on Autonomic Computing (ICAC07), 2007.
[68] R. Ublig, et al., Intel virtualization technology, IEEE Comput., (May) (2005).
[69] H. Van, F. Tran, Autonomic virtual resource management for service hosting platforms, CLOUD (2009).
[70] A. Verma, P. Ahuja, A. Neogi, pMapper: Power and migration cost aware application placement in virtualized systems, in: Proceedings of the 9th International Middleware Conference, 2008, pp. 243–264.
[71] VMware (white paper). Understanding Full Virtualization, Paravirtualization, and Hardware Assist, www.vmware.com/files/pdf/VMware_paravirtualization.pdf.
[72] VMware (white paper). The vSphere 4 Operating System for Virtualizing Datacenters, News release, February 2009, www.vmware.com/products/vsphere/, April 2010.
[73] J. Walters, et al., A comparison of virtualization technologies for HPC, in: Proceedings of Advanced Information Networking and Applications (AINA), 2008.
[74] P. Wells, K. Chakraborty, G.S. Sohi, Dynamic heterogeneity and the need for multicore virtualization, ACM SIGOPS Operat. Syst. Rev. 43 (2) (2009) 5–14.
[75] T. Wood, G. Levin, P. Shenoy, Memory buddies: Exploiting page sharing for smart collocation in virtualized data centers, in: Proceedings of the 5th International Conference on Virtual Execution Environments (VEE), 2009.
[76] J. Xun, K. Chen, W. Zheng, Amigo file system: CAS based storage management for a virtual cluster system, in: Proceedings of IEEE 9th International Conference on Computer and Information Technology (CIT), 2009.
[77] Y. Yu, OS-level Virtualization and Its Applications, Ph.D. dissertation, Computer Science Department, SUNY, Stony Brook, New York, December 2007.
[78] Y. Yu, F. Guo, et al., A feather-weight virtual machine for windows applications, in: Proceedings of the 2nd International Conference on Virtual Execution Environments (VEE), Ottawa, Canada, 14–16 June 2006.
[79] M. Zhao, J. Zhang, et al., Distributed file system support for virtual machines in grid computing, in: Proceedings of High Performance Distributed Computing, 2004.

HOMEWORK PROBLEMS

Problem 3.1

Briefly answer the following questions on virtualization levels. Highlight the key points and identify the distinctions in different approaches. Discuss their relative advantages, shortcomings and limitations. Also identify example systems implemented at each level.

Problem 3.2

Explain the differences between hypervisor and para-virtualization and give one example VMM (virtual machine monitor), that was built in each of the two categories.

Problem 3.3

Install the VMware Workstation on a Windows XP or Vista personal computer or laptop, and then install Red Hat Linux and Windows XP in the VMware Workstation. Configure the network settings of Red Hat Linux and Windows XP to get on the Internet. Write an installation and configuration guide for the VMware Workstation, Red Hat Linux, and Windows XP systems. Include any trouble-shooting tips in the guide.

Problem 3.4

Download a new kernel package from www.kernel.org/. Compile it in Red Hat Linux in the VMware Workstation installed in Problem 3.3 with Red Hat Linux on a real computer. Compare the time required for the two compilations. Which one takes longer to compile? What are their major differences?

Problem 3.5

Install Xen on a Red Hat Linux machine in two methods from the binary code or from the source code. Compile installation guides for the two methods used. Describe the dependencies of utilities and packages along with troubleshooting tips.

Problem 3.6

Install Red Hat Linux on the Xen you installed in Problem 3.5. Download nbench from www.tux.org/~mayer/linux/bmark.html. Run the nbench on the VM using Xen and on a real machine. Compare the performance of the programs on the two platforms.

Problem 3.7

Use the utilities for easing deployment of Google enterprise applications in VMs. The Google-vm-deployment tool can be downloaded from http://code.google.com/p/google-vm-deployment/.

Problem 3.8

Describe the approaches used to exchange data among the domains of Xen and design experiments to compare the performance of data communication between the domains. This is designed to familiarize you with the Xen programming environment. It may require a longer period of time to port the Xen code, implement the application code, perform the experiments, collect the performance data, and interpret the results.

Problem 3.9

Build your own LAN by using the VMware Workstation. The topological structure of the LAN is specified in Figure 3.31. Machine A is required to install Red Hat Linux while machine B is required to install Windows XP.

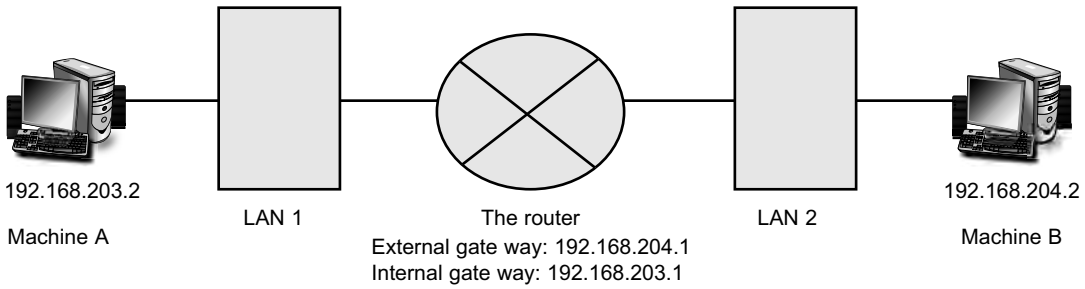


FIGURE 3.31

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 for multi-core CMP design and applications. Repeat Problem 3.10 with a survey report after the research study.

Problem 3.12

Hardware and software resource deployment are 4 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.

Problem 3.15

Design and implement a VM execution environment for grid computing based on VMware Server. The environment should enable grid users and resource providers to use services that are unique to a VM-based approach to distributed computing. Users can define customized execution environments which can then be archived, copied, shared, and instantiated as multiple runtime clones.

Problem 3.16

Design a large-scale virtual cluster system. This problem may require three students to work together for a semester. Assume that users can create multiple VMs at one time. Users can also manipulate and configure multiple VMs at the same time. Common software such as OS or libraries are preinstalled as templates. These templates enable users to create a new execution environment rapidly. Finally, you can assume that users have their own profiles which store the identification of data blocks.

Problem 3.17

Figure 3.32 shows another VIOLIN adaptation scenario for changes in virtual environments. There are four VIOLIN applications running in two cluster domains. Trace the three steps of VIOLIN job execution and discuss the gains in resource utilization after live migration of the virtual execution

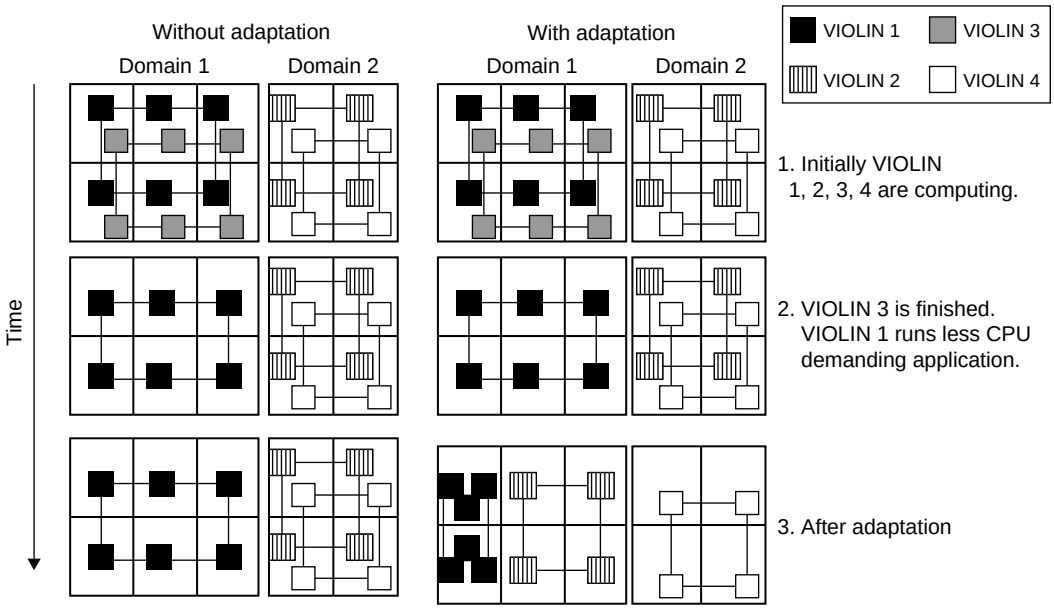


FIGURE 3.32

An adaptation scenario with four VIOLINs running in two cluster domains in the VIOLIN Virtual clustering experiments.

(*Courtesy of P. Ruth, et al. [55]*)

environment in the two cluster domains. You can check your results against the cited paper to compare your observations.

Problem 3.18

After studying the material presented in Section 3.3.5, plus reading the papers by Wells, et al. [74] and by Marty and Hill in [39] answer the following two questions:

a. Distinguish virtual cores from physical cores and discuss the mapping technique in Wells's paper on upgrading resource utilization and fault tolerance in using virtualized multicore processors.

b. Study the cache coherence protocol presented in the Marty and Hill paper and discuss its feasibility and advantages to implement on many-core CMP in the future.

CHAPTER 4

Cloud Platform Architecture over Virtualized Data Centers

CHAPTER OUTLINE

This chapter covers design principles, architectures, and enabling technologies of cloud platforms. We begin with a discussion of data-center design and management. Next, we present design choices for building compute and storage cloud platforms. We cover layered platform structure, virtualization support, resource provisioning, and infrastructure management. Several public cloud platforms are also studied, including Amazon Web Services, the Google App Engine, and Microsoft Azure. Subsequent chapters are devoted to service-oriented architectures, cloud computing paradigms, programming environments, and future cloud extensions.

cloud computing, Infrastructure-as-a-Service (Iaas), Platform-as-a-Service (PaaS), Software-as-a-Service (SaaS), data centers, cloud architecture, public clouds, resource management, cloud security

4.1 Cloud Computing and Service Models

Over the past two decades, the world economy has rapidly moved from manufacturing to more service-oriented. In 2010, 80 percent of the U.S. economy was driven by the service industry, leaving only 15 percent in manufacturing and 5 percent in agriculture and other areas. Cloud computing benefits the service industry most and advances business computing with a new paradigm. In 2009, the global cloud service marketplace reached $17.4 billion. IDC predicted in 2010 that the cloud-based economy may increase to $44.2 billion by 2013. Developers of innovative cloud applications no longer acquire large capital equipment in advance. They just rent the resources from some large data centers that have been automated for this purpose.

In this and the next two chapters, we will study the cloud platform architecture, service models, and programming environments. Users can access and deploy cloud applications from anywhere in the world at very competitive costs. Virtualized cloud platforms are often built on top of large data centers. With that in mind, we examine first the server cluster in a data center and its interconnection issues. In other words, clouds aim to power the next generation of data centers by architecting them as virtual resources over automated

hardware, databases, user interfaces, and application environments. In this sense, clouds grow out of the desire to build better data centers through automated resource provisioning.

4.1.1 Public, Private, and Hybrid Clouds

The concept of cloud computing has evolved from cluster, grid, and utility computing. Cluster and grid computing leverage the use of many computers in parallel to solve problems of any size. Utility and Software as a Service (SaaS) provide computing resources as a service with the notion of pay per use. Cloud computing leverages dynamic resources to deliver large numbers of services to end users. Cloud computing is a high-throughput computing (HTC) paradigm whereby the infrastructure provides the services through a large data center or server farms. The cloud computing model enables users to share access to resources from anywhere at any time through their connected devices.

Recall the introduction in Chapter 1 in which we said that the cloud will free users to focus on user application development and create business value by outsourcing job execution to cloud providers. In this scenario, the computations (programs) are sent to where the data is located, rather than copying the data to millions of desktops as in the traditional approach. Cloud computing avoids large data movement, resulting in much better network bandwidth utilization. Furthermore, machine virtualization has enhanced resource utilization, increased application flexibility, and reduced the total cost of using virtualized data-center resources.

The cloud offers significant benefit to IT companies by freeing them from the low-level task of setting up the hardware (servers) and managing the system software. Cloud computing applies a virtual platform with elastic resources put together by on-demand provisioning of hardware, software, and data sets, dynamically. The main idea is to move desktop computing to a service-oriented platform using server clusters and huge databases at data centers. Cloud computing leverages its low cost and simplicity to both providers and users. According to Ian Foster [25], cloud computing intends to leverage multitasking to achieve higher throughput by serving many heterogeneous applications, large or small, simultaneously.

4.1.1.1 Centralized versus Distributed Computing

Some people argue that cloud computing is centralized computing at data centers. Others claim that cloud computing is the practice of distributed parallel computing over data-center resources. These represent two opposite views of cloud computing. All computations in cloud applications are distributed to servers in a data center. These are mainly *virtual machines* (VMs) in virtual clusters created out of data-center resources. In this sense, cloud platforms are systems distributed through virtualization.

As Figure 4.1 shows, both *public clouds* and *private clouds* are developed in the Internet. As many clouds are generated by commmercial providers or by enterprises in a distributed manner, they will be interconnected over the Internet to achieve scalable and efficient computing services. Commercial cloud providers such as Amazon, Google, and Microsoft created their platforms to be distributed geographically. This distribution is partially attributed to fault tolerance, response latency reduction, and even legal reasons. Intranet-based private clouds are linked to public clouds to get additional resources. Nevertheless, users in Europe may not feel comfortable using clouds in the United States, and vice versa, until extensive *service-level agreements* (SLAs) are developed between the two user communities.

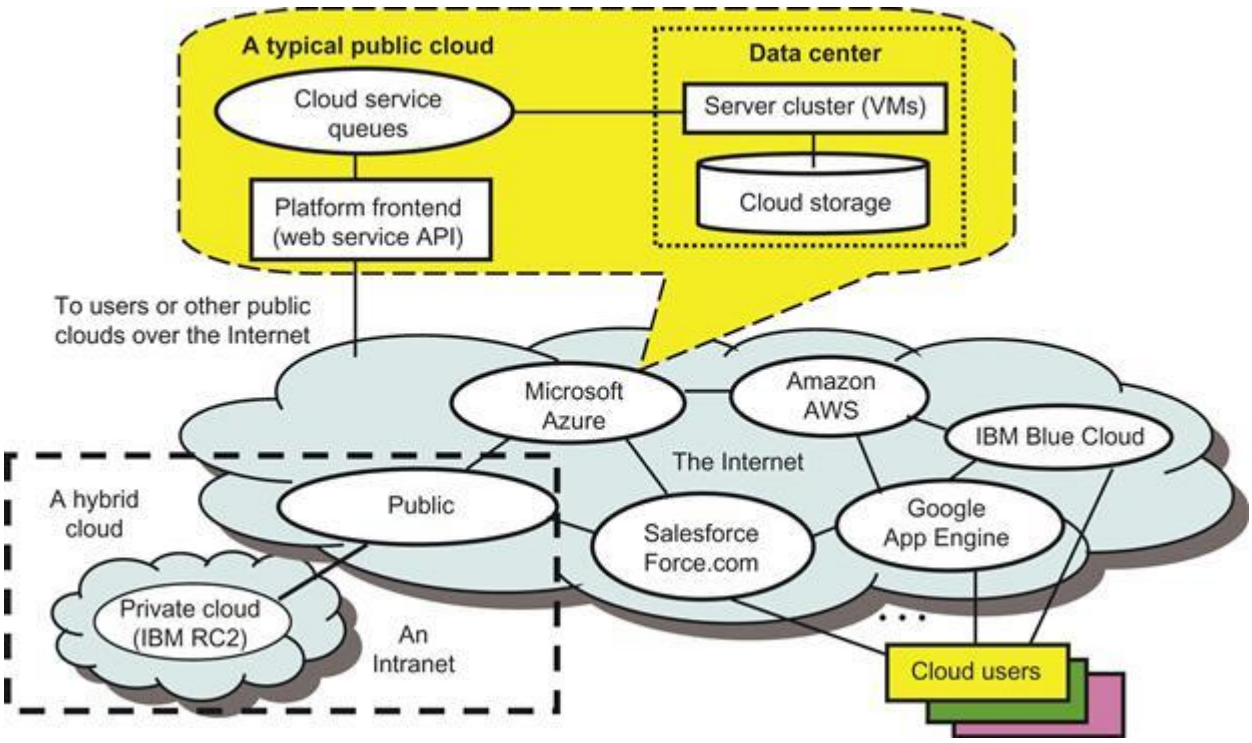


FIGURE 4.1 Public, private, and hybrid clouds illustrated by functional architecture and connectivity of representative clouds available by 2011.

4.1.1.2 Public Clouds

A *public cloud* is built over the Internet and can be accessed by any user who has paid for the service. Public clouds are owned by service providers and are accessible through a subscription. The callout box in top of Figure 4.1 shows the architecture of a typical public cloud. Many public clouds are available, including Google App Engine (GAE), Amazon Web Services (AWS), Microsoft Azure, IBM Blue Cloud, and Salesforce.com's Force.com. The providers of the aforementioned clouds are commercial providers that offer a publicly accessible remote interface for creating and managing VM instances within their proprietary infrastructure. A public cloud delivers a selected set of business processes. The application and infrastructure services are offered on a flexible price-per-use basis.

4.1.1.3 Private Clouds

A *private cloud* is built within the domain of an intranet owned by a single organization. Therefore, it is client owned and managed, and its access is limited to the owning clients and their partners. Its deployment was not meant to sell capacity over the Internet through publicly accessible interfaces. Private clouds give local users a flexible and agile private infrastructure to run service workloads within their administrative domains. A private cloud is supposed to deliver more efficient and convenient cloud services. It may impact the cloud standardization, while retaining greater customization and organizational control.

4.1.1.4 Hybrid Clouds

A *hybrid cloud* is built with both public and private clouds, as shown at the lower-left corner of Figure 4.1. Private clouds can also support a hybrid cloud model by supplementing local infrastructure with computing capacity from an external public cloud. For example, the *Research Compute Cloud* (RC2) is a private cloud, built by IBM, that interconnects the computing and IT resources at eight IBM Research Centers scattered throughout the United States, Europe, and Asia. A hybrid cloud provides access to clients, the partner network, and

third parties. In summary, public clouds promote standardization, preserve capital investment, and offer application flexibility. Private clouds attempt to achieve customization and offer higher efficiency, resiliency, security, and privacy. Hybrid clouds operate in the middle, with many compromises in terms of resource sharing.

4.1.1.5 Data-Center Networking Structure

The core of a cloud is the server cluster (or VM cluster). Cluster nodes are used as compute nodes. A few control nodes are used to manage and monitor cloud activities. The scheduling of user jobs requires that you assign work to virtual clusters created for users. The gateway nodes provide the access points of the service from the outside world. These gateway nodes can be also used for security control of the entire cloud platform. In physical clusters and traditional grids, users expect static demand of resources. Clouds are designed to handle fluctuating workloads, and thus demand variable resources dynamically. Private clouds will satisfy this demand if properly designed and managed.

Data centers and supercomputers have some similarities as well as fundamental differences. We discussed supercomputers in Chapter 2. In the case of data centers, scaling is a fundamental requirement. Data-center server clusters are typically built with large number of servers, ranging from thousands to millions of servers (nodes). For example, Microsoft has a data center in the Chicago area that has 100,000 eight-core servers, housed in 50 containers. In supercomputers, a separate data farm is used, while a data center uses disks on server nodes plus memory cache and databases.

Data centers and supercomputers also differ in networking requirements, as illustrated in Figure 4.2. Super computers use custom-designed high-bandwidth networks such as fat trees or 3D torus networks (which we discussed in Chapter 2). Data-center networks are mostly IP-based commodity networks, such as the 10 Gbps Ethernet network, which is optimized for Internet access. Figure 4.2 shows a multilayer structure for accessing the Internet. The server racks are at the bottom Layer 2, and they are connected through fast switches (S) as the hardware core. The data center is connected to the Internet at Layer 3 with many *access routers* (ARs) and *border routers* (BRs).

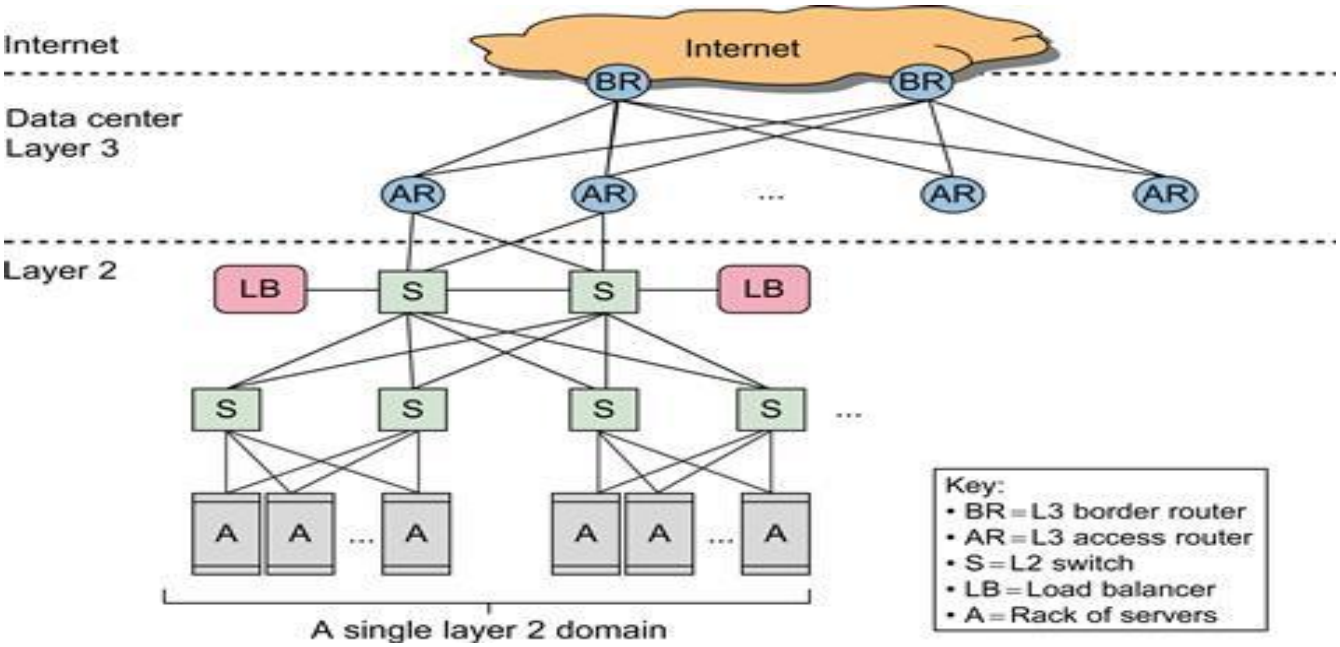


FIGURE 4.2 Standard data-center networking for the cloud to access the Internet Courtesy of Dennis Gannon, 2010 [26]

An example of a private cloud is the one the U.S. National Aeronautics and Space Administration (NASA) is building to enable researchers to run climate models on remote systems it provides. This can save users the capital expense of HPC machines at local sites. Furthermore, NASA can build the complex weather models around its data centers, which is more cost-effective. Another good example is the cloud built by the European Council for Nuclear Research (CERN). This is a very big private cloud designed to distribute data, applications, and computing resources to thousands of scientists around the world.

These cloud models demand different levels of performance, data protection, and security enforcement. In this case, different SLAs may be applied to satisfy both providers and paid users. Cloud computing exploits many existing technologies. For example, grid computing is the backbone of cloud computing in that the grid has the same goals of resource sharing with better utilization of research facilities. Grids are more focused on delivering storage and computing resources while cloud computing aims to achieve economies of scale with abstracted services and resources.

4.1.1.6 Cloud Development Trends

Although most clouds built in 2010 are large public clouds, the authors believe private clouds will grow much faster than public clouds in the future. Private clouds are easier to secure and more trustworthy within a company or organization. Once private clouds become mature and better secured, they could be open or converted to public clouds. Therefore, the boundary between public and private clouds could be blurred in the future. Most likely, most future clouds will be hybrid in nature.

For example, an e-mail application can run in the service-access nodes and provide the user interface for outside users; the application can get the service from the internal cloud computing services (e.g., the e-mail storage service). There are also some service nodes designed to support the proper functioning of cloud computing clusters. These nodes are called runtime supporting service nodes. For example, there might be distributed locking services for supporting specific applications. Finally, it is possible that there will be some independent service nodes. Those nodes would provide independent services for other nodes in the cluster. For example, a news service need geographical information under service-access nodes.

With cost-effective performance as the key concept of clouds, we will consider the public cloud in this chapter, unless otherwise specified. Many executable application codes are much smaller than the web-scale data sets they process. Cloud computing avoids large data movement during execution. This will result in less traffic on the Internet and better network utilization. Clouds also alleviate the petascale I/O problem. Cloud performance and its *Quality of Service* (QoS) are yet to be proven in more real-life applications. We will model the performance of cloud computing in Chapter 9, along with data protection, security measures, service availability, fault tolerance, and operating cost.

4.1.2 Cloud Ecosystem and Enabling Technologies

Cloud computing platforms differ from conventional computing platforms in many aspects. In this section, we will identify their differences in computing paradigms and cost models

applied. The traditional computing model is specified below by the process on the left, which involves buying the hardware, acquiring the necessary system software, installing the system, testing the configuration, and executing the application code and management of resources. What is even worse is that this cycle repeats itself in about every 18 months, meaning the machine we bought becomes obsolete every 18 months.

The cloud computing paradigm is shown on the right. This computing model follows a pay-as-you-go model. Therefore the cost is significantly reduced, because we simply rent computer resources without buying the computer in advance. All hardware and software resources are leased from the cloud provider without capital investment on the part of the users. Only the execution phase costs some money. The experts at IBM have estimated that an 80 percent to 95 percent saving results from cloud computing, compared with the conventional computing paradigm. This is very much desired, especially for small businesses, which requires limited computing power and thus avoid the purchase of expensive computers or servers repeatedly every few years.

Classical Computing	**Cloud Computing**
(Repeat the following cycle every 18 months)	(Pay as you go per each service provided)
Buy and own	**Subscribe**
Hardware, system software, applications to meet peak needs	- - - -
Install, configure, test, verify, evaluate, manage	**Use** (Save about 80-95% of the total cost)
- - - -	- -
Use	(Finally)

- - - - **$ - Pay for what you use**

Pay $$$$$ (High cost) based on the QoS

For example, IBM has estimated that the worldwide cloud service market may reach $126 billion by 2012, including components, infrastructure services, and business services. Internet clouds work as service factories built around multiple data centers. To formalize the above cloud computing model, we characterize the cloud cost model, the cloud ecosystems, and enabling technologies. These topics help our readers understand the motivations behind cloud computing. The intention is to remove the barriers of cloud computing

4.1.2.1 Cloud Design Objectives

Despite the controversy surrounding the replacement of desktop or deskside computing by centralized computing and storage services at data centers or big IT companies, the cloud computing community has reached some consensus on what has to be done to make cloud computing universally acceptable. The following list highlights six design objectives for cloud computing:

• **Shifting computing from desktops to data centers** Computer processing, storage, and software delivery is shifted away from desktops and local servers and toward data centers over the Internet.
• **Service provisioning and cloud economics** Providers supply cloud services by signing SLAs with consumers and end users. The services must be efficient in terms of computing, storage, and power consumption. Pricing is based on a pay-as-you-go policy.
• **Scalability in performance** The cloud platforms and software and infrastructure services must be able to scale in performance as the number of users increases.
• **Data privacy protection** Can you trust data centers to handle your private data and records? This concern must be addressed to make clouds successful as trusted services.
• **High quality of cloud services** The QoS of cloud computing must be standardized to make clouds interoperable among multiple providers.
• **New standards and interfaces** This refers to solving the data lock-in problem associated with data centers or cloud providers. Universally accepted APIs and access protocols are needed to provide high portability and flexibility of virtualized applications.

4.1.2.2 Cost Model

In traditional IT computing, users must acquire their own computer and peripheral equipment as capital expenses. In addition, they have to face operational expenditures in operating and maintaining the computer systems, including personnel and service costs. Figure 4.3(a) shows the addition of variable operational costs on top of fixed capital investments in traditional IT. Note that the fixed cost is the main cost, and that it could be reduced slightly as the number of users increases. However, the operational costs may increase sharply with a larger number of users. Therefore, the total cost escalates quickly with massive numbers of users. On the other hand, cloud computing applies a pay-per-use business model, in which user jobs are outsourced to data centers. To use the cloud, one has

no up-front cost in hardware acquisitions. Only variable costs are experienced by cloud users, as demonstrated in Figure 4.3(b).

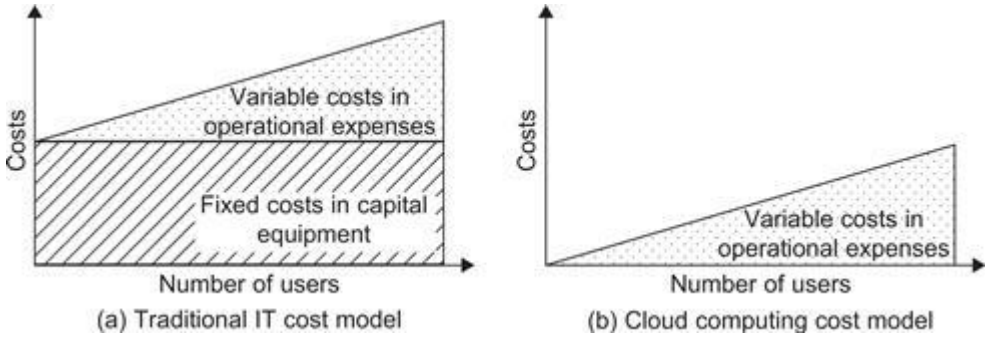


FIGURE 4.3 Computing economics between traditional IT users and cloud users.

Overall, cloud computing will reduce computing costs significantly for both small users and large enterprises. Computing economics does show a big gap between traditional IT users and cloud users. The savings in acquiring expensive computers up front releases a lot of burden for startup companies. The fact that cloud users only pay for operational expenses and do not have to invest in permanent equipment is especially attractive to massive numbers of small users. This is a major driving force for cloud computing to become appealing to most enterprises and heavy computer users. In fact, any IT users whose capital expenses are under more pressure than their operational expenses should consider sending their overflow work to utility computing or cloud service providers.

4.1.2.3 Cloud Ecosystems

With the emergence of various Internet clouds, an ecosystem of providers, users, and technologies has appeared. This ecosystem has evolved around public clouds. Strong interest is growing in open source cloud computing tools that let organizations build their own IaaS clouds using their internal infrastructures. Private and hybrid clouds are not exclusive, since public clouds are involved in both cloud types. A private/hybrid cloud allows remote access to its resources over the Internet using remote web service interfaces such as that used in Amazon EC2.

An ecosystem was suggested by Sotomayor, et al. [39] (Figure 4.4) for building private clouds. They suggested four levels of ecosystem development in a private cloud. At the user end, consumers demand a flexible platform. At the cloud management level, the cloud manager provides virtualized resources over an IaaS platform. At the virtual infrastructure (VI) management level, the manager allocates VMs over multiple server clusters. Finally, at the VM management level, the VM managers handle VMs installed on individual host machines. An ecosystem of cloud tools attempts to span both cloud management and VI management. Integrating these two layers is complicated by the lack of open and standard interfaces between them.

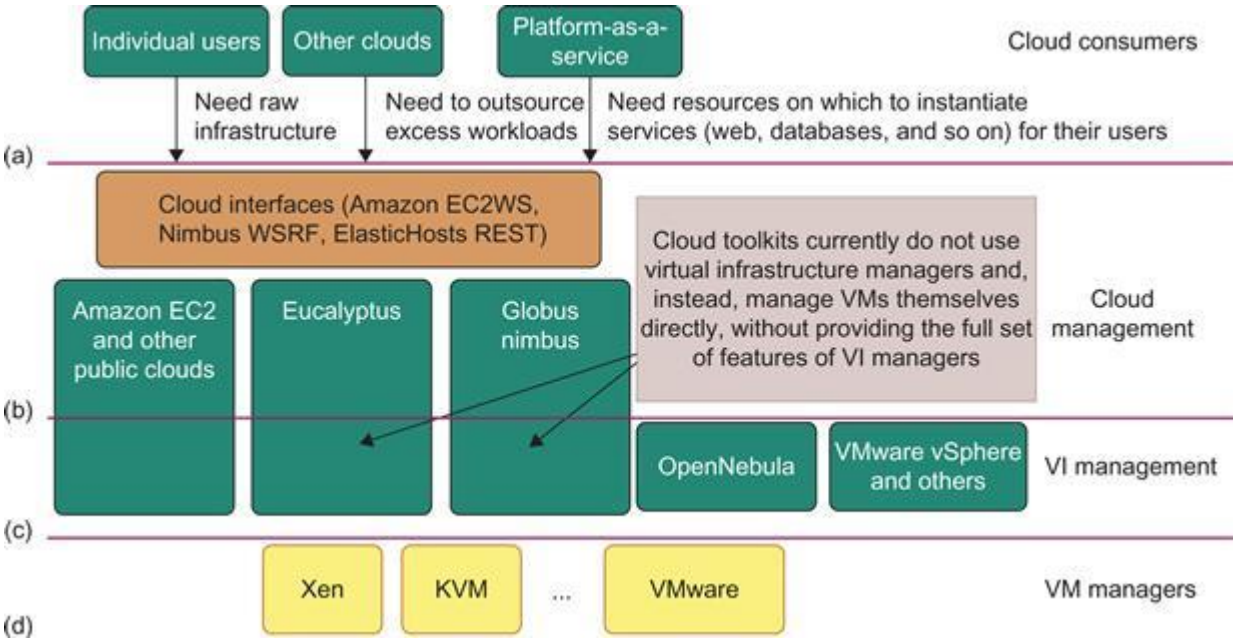


FIGURE 4.4 Cloud ecosystem for building private clouds: (a) Consumers demand a flexible platform; (b) Cloud manager provides virtualized resources over an IaaS platform; (c) VI manager allocates VMs; (d) VM managers handle VMs installed on servers Courtesy of Sotomayor, et al. © IEEE [68]

An increasing number of startup companies are now basing their IT strategies on cloud resources, spending little or no capital to manage their own IT infrastructures. We desire a flexible and open architecture that enables organizations to build private/hybrid clouds. VI management is aimed at this goal. Example VI tools include oVirt (https://fedorahosted.org/ovirt/), vSphere/4 (www.vmware.com/products/vsphere/) from VMWare, and VM Orchestrator (www.platform.com/Products/platform-vm-orchestrator) from Platfom Computing.

These tools support dynamic placement and VM management on a pool of physical resources, automatic load balancing, server consolidation, and dynamic infrastructure resizing and partitioning. In addition to public clouds such as Amazon EC2, Eucalyptus and Globus Nimbus are open source tools for virtualization of cloud infrastructure. To access these cloud management tools, one can use the Amazon EC2WS, Nimbus WSRF, and ElasticHost REST cloud interfaces. For VI management, OpenNebula and VMware vSphere can be used to manage all VM generation including Xen, KVM, and VMware tools.

4.1.2.4 Surge of Private Clouds

In general, private clouds leverage existing IT infrastructure and personnel within an enterprise or government organization. Both public and private clouds handle workloads dynamically. However, public clouds should be designed to handle workloads without communication dependency. Both types of clouds distribute data and VM resources. However, private clouds can balance workloads to exploit IT resources more efficiently within the same intranet. Private clouds can also provide preproduction testing and enforce data privacy and security policies more effectively. In a public cloud, the surge workload is often offloaded. The major advantage of public clouds lies in the avoidance of capital expenses by users in IT investments in hardware, software, and personnel.

Most companies start with virtualization of their computing machines to lower the operating costs. Companies such as Microsoft, Oracle, and SAP may want to establish policy-driven

management of their computing resources, mainly to improve QoS to their employees and customers. By integrating virtualized data centers and company IT resources, they offer *IT as a service* to improve the agility of their company operations. This approach avoids replacement of a large number of servers every 18 months. As a result, these companies can upgrade their IT efficiency significantly.

4.1.3 Infrastructure-as-a-Service (IaaS)

Cloud computing delivers infrastructure, platform, and software (application) as services, which are made available as subscription-based services in a pay-as-you-go model to consumers. The services provided over the cloud can be generally categorized into three different service models: namely IaaS, Platform as a Service (PaaS), and Software as a Service (SaaS). These form the three pillars on top of which cloud computing solutions are delivered to end users. All three models allow users to access services over the Internet, relying entirely on the infrastructures of cloud service providers.

These models are offered based on various SLAs between providers and users. In a broad sense, the SLA for cloud computing is addressed in terms of service availability, performance, and data protection and security. Figure 4.5 illustrates three cloud models at different service levels of the cloud. SaaS is applied at the application end using special interfaces by users or clients. At the PaaS layer, the cloud platform must perform billing services and handle job queuing, launching, and monitoring services. At the bottom layer of the IaaS services, databases, compute instances, the file system, and storage must be provisioned to satisfy user demands.

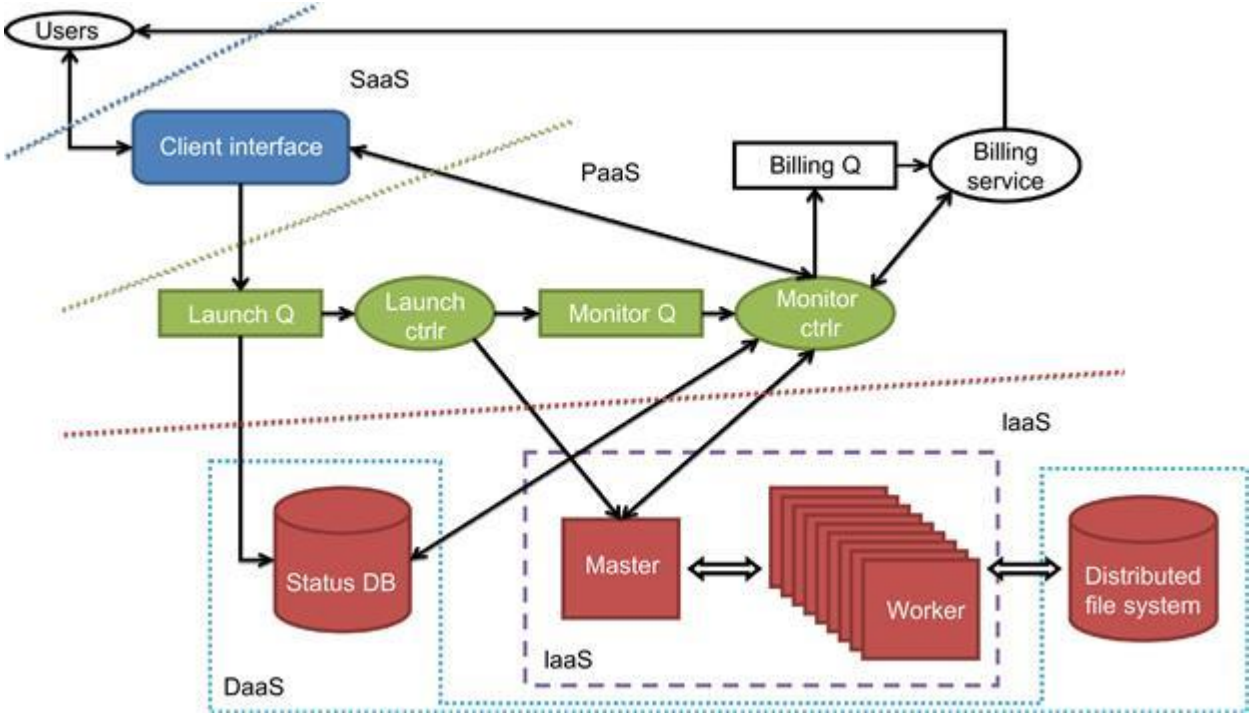


FIGURE 4.5 The IaaS, PaaS, and SaaS cloud service models at different service levels. Courtesy of J. Suh and S. Kang, USC

4.1.3.1 Infrastructure as a Service

This model allows users to use virtualized IT resources for computing, storage, and networking. In short, the service is performed by rented cloud infrastructure. The user can deploy and run his applications over his chosen OS environment. The user does not manage

or control the underlying cloud infrastructure, but has control over the OS, storage, deployed applications, and possibly select networking components. This IaaS model encompasses *storage as a service*, compute *instances as a service*, and *communication as a service*.

Table 4.1 : Public Cloud Offerings of IaaS [10,18]

Cloud Name	VM Instance Capacity	API and Access Tools	Hypervisor, Guest OS
Amazon EC2	Each instance has 1–20 EC2 processors, 1.7–15 GB of memory, and 160–1.69 TB of storage.	CLI or web Service (WS) portal	Xen, Linux, Windows
GoGrid	Each instance has 1–6 CPUs, 0.5–8 GB of memory, and 30–480 GB of storage.	REST, Java, PHP, Python, Ruby	Xen, Linux, Windows
Rackspace Cloud	Each instance has a four-core CPU, 0.25–16 GB of memory, and 10–620 GB of storage.	REST, Python, PHP, Java, C#, .NET	Xen, Linux
FlexiScale in the UK	Each instance has 1–4 CPUs, 0.5–16 GB of memory, and 20–270 GB of storage.	web console	Xen, Linux, Windows
Joyent Cloud	Each instance has up to eight CPUs, 0.25–32 GB of memory, and 30–480 GB of storage.	No specific API, SSH, Virtual/Min	OS-level virtualization, OpenSolaris

The *Virtual Private Cloud (VPC)* in Example 4.1 shows how to provide Amazon EC2 clusters and S3 storage to multiple users. Many startup cloud providers have appeared in recent years. GoGrid, FlexiScale, and Aneka are good examples. Table 4.1 summarizes the IaaS offerings by five public cloud providers. Interested readers can visit the companies' web sites for updated information. More examples can be also found in two recent cloud books [10,18].

Example 4.1 : Amazon VPC for Multiple Tenants

A user can use a private facility for basic computations. When he must meet a specific workload requirement, he can use the Amazon VPC to provide additional EC2 instances or more storage (S3) to handle urgent applications. Figure 4.6 shows VPC which is essentially a private cloud designed to address the privacy concerns of public clouds that hamper their application when sensitive data and software are involved.

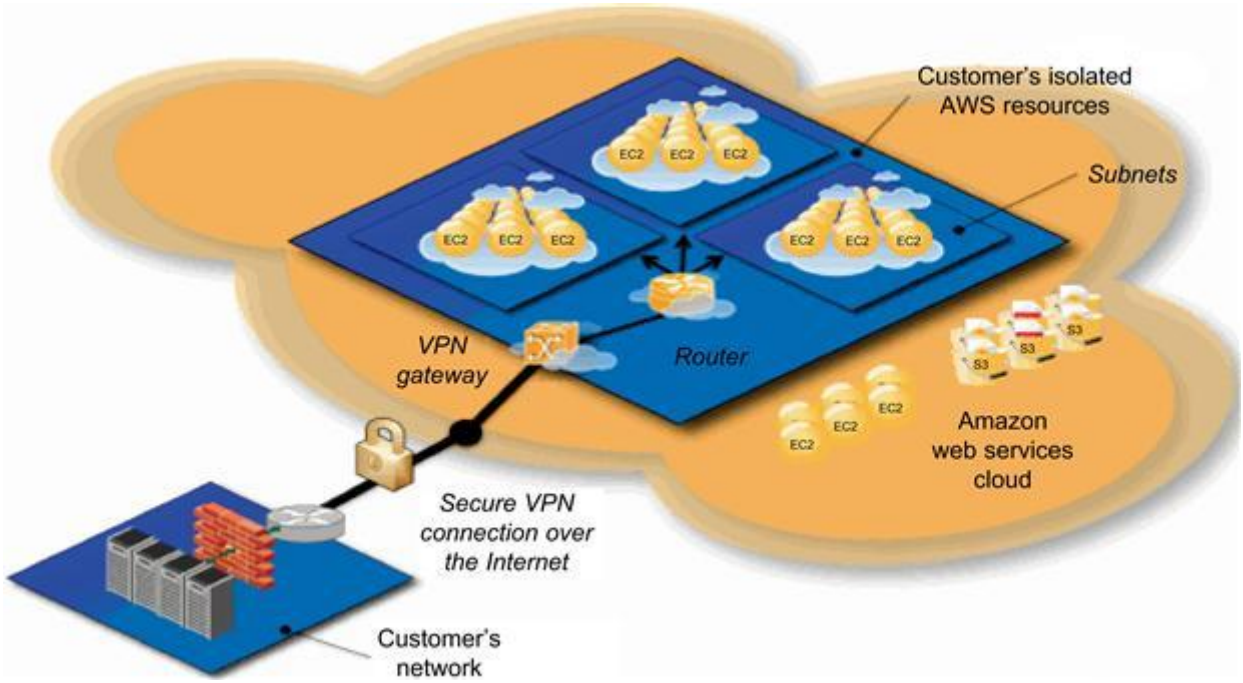


FIGURE 4.6 Amazon VPC (virtual private cloud) Courtesy of VMWare,http://aws.amazon.com/vpc/

Amazon EC2 provides the following services: resources from multiple data centers globally distributed, CL1, web services (SOAP and Query), web-based console user interfaces, access to VM instances via SSH and Windows, 99.5 percent available agreements, per-hour pricing, Linux and Windows OSes, and automatic scaling and load balancing. We will illustrate the use of EC2 in more detail in Chapter 6. VPC allows the user to isolate provisioned AWS processors, memory, and storage from interference by other users. Both *auto-scaling* and *elastic load balancing* services can support related demands. Auto-scaling enables users to automatically scale their VM instance capacity up or down. With auto-scaling, one can ensure that a sufficient number of Amazon EC2 instances are provisioned to meet desired performance. Or one can scale down the VM instance capacity to reduce costs, when the workload is reduced.

4.1.4 Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS)

In this section, we will introduce the PaaS and SaaS models for cloud computing. SaaS is often built on top of the PaaS, which is in turn built on top of the IaaS.

4.1.4.1 Platform as a Service (PaaS)

To be able to develop, deploy, and manage the execution of applications using provisioned resources demands a cloud platform with the proper software environment. Such a platform includes operating system and runtime library support. This has triggered the creation of the PaaS model to enable users to develop and deploy their user applications. Table 4.2 highlights cloud platform services offered by five PaaS services. Further details of some of these PaaS offerings are provided in Section 4.4 and inChapter 6. Some illustrative examples and case studies can be found in [10,18].

Table 4.2 : Five Public Cloud Offerings of PaaS [10,18]

Cloud Name	Languages and Developer Tools	Programming Models Supported by Provider	Target Applications and Storage Option
Google App Engine	Python, Java, and Eclipse-based IDE	MapReduce, web programming on demand	Web applications and BigTable storage
Salesforce.com's Force.com	Apex, Eclipse-based IDE, web-based Wizard	Workflow, Excel-like formula, Web programming on demand	Business applications such as CRM
Microsoft Azure	.NET, Azure tools for MS Visual Studio	Unrestricted model	Enterprise and web applications
Amazon Elastic MapReduce	Hive, Pig, Cascading, Java, Ruby, Perl, Python, PHP, R, C++	MapReduce	Data processing and e-commerce
Aneka	.NET, stand-alone SDK	Threads, task, MapReduce	.NET enterprise applications, HPC

The platform cloud is an integrated computer system consisting of both hardware and software infrastructure. The user application can be developed on this virtualized cloud platform using some programming languages and software tools supported by the provider (e.g., Java, Python, .NET). The user does not manage the underlying cloud infrastructure. The cloud provider supports user application development and testing on a well-defined service platform. This PaaS model enables a collaborated software development platform for users from different parts of the world. This model also encourages third parties to provide software management, integration, and service monitoring solutions.

Example 4.2

Google App Engine for PaaS Applications

As web applications are running on Google's server clusters, they share the same capability with many other users. The applications have features such as automatic scaling and load balancing which are very convenient while building web applications. The distributed scheduler mechanism can also schedule tasks for triggering events at specified times and regular intervals. Figure 4.7shows the operational model for GAE. To develop applications using GAE, a development environment must be provided.

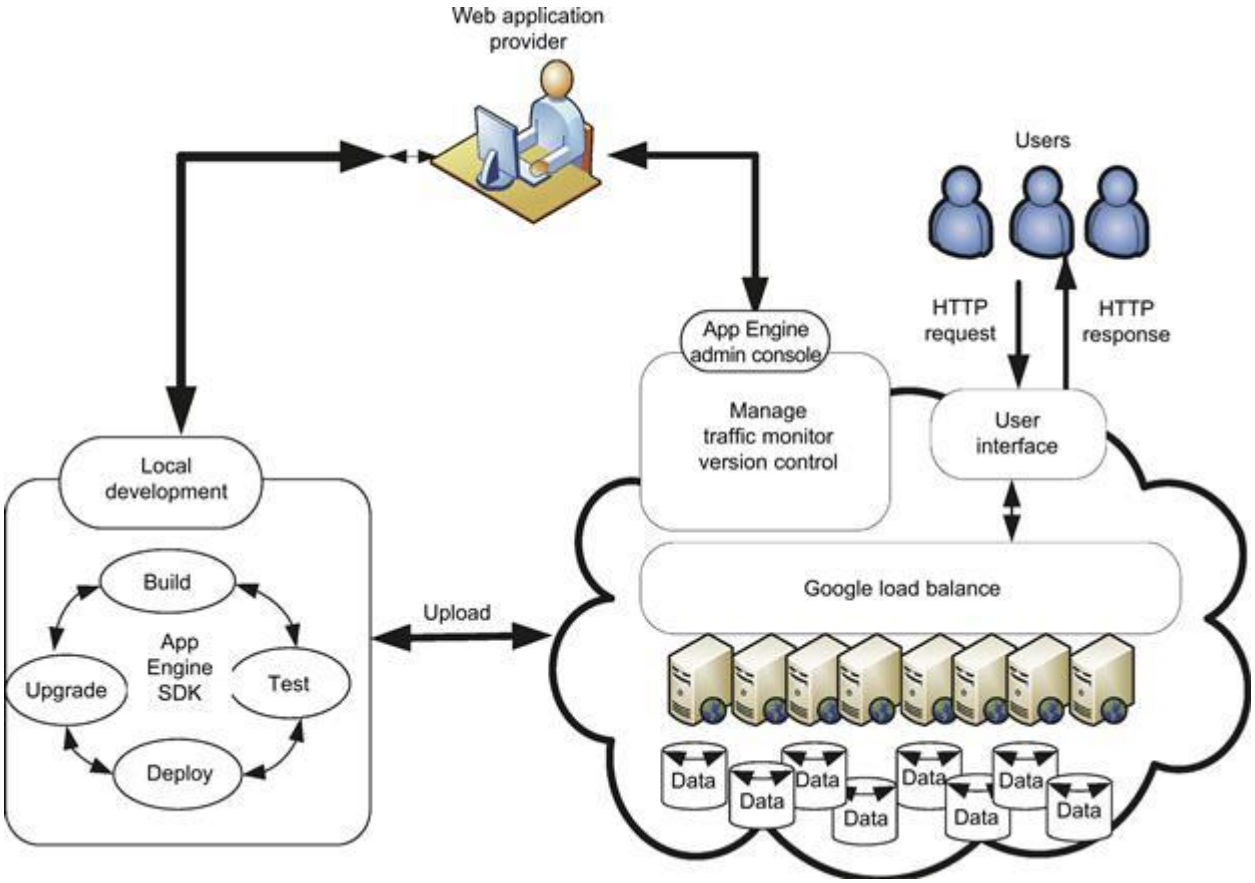


FIGURE 4.7 Google App Engine platform for PaaS operations Courtesy of Yangting Wu, USC

Google provides a fully featured local development environment that simulates GAE on the developer's computer. All the functions and application logic can be implemented locally which is quite similar to traditional software development. The coding and debugging stages can be performed locally as well. After these steps are finished, the SDK provided provides a tool for uploading the user's application to Google's infrastructure where the applications are actually deployed. Many additional third-party capabilities, including software management, integration, and service monitoring solutions, are also provided.
Here are some useful links when logging on to the GAE system:

- Google App Engine home page: http://code.google.com/appengine/
- Sign up for an account or use your Gmail account name: https://appengine.google.com/
- Download GAE SDK: http://code.google.com/appengine/downloads.html
- Python Getting Started Guide: http://code.google.com/appengine/docs/python/gettingstarted/
- Java Getting Started Guide: http://code.google.com/appengine/docs/java/gettingstarted/
- Quota page for free service: http://code.google.com/appengine/docs/quotas.html#Resources
- Billing page if you go over the quota:http://code.google.com/appengine/docs/billing.html#Billable_Quota_Unit_Cost

4.1.4.2 Software as a Service (SaaS)

This refers to browser-initiated application software over thousands of cloud customers. Services and tools offered by PaaS are utilized in construction of applications and management of their deployment on resources offered by IaaS providers. The SaaS model provides software applications as a service. As a result, on the customer side, there is no upfront investment in servers or software licensing. On the provider side, costs are kept

rather low, compared with conventional hosting of user applications. Customer data is stored in the cloud that is either vendor proprietary or publicly hosted to support PaaS and IaaS.

The best examples of SaaS services include Google Gmail and docs, Microsoft SharePoint, and the CRM software from Salesforce.com. They are all very successful in promoting their own business or are used by thousands of small businesses in their day-to-day operations. Providers such as Google and Microsoft offer integrated IaaS and PaaS services, whereas others such as Amazon and GoGrid offer pure IaaS services and expect third-party PaaS providers such as Manjrasoft to offer application development and deployment services on top of their infrastructure services. To identify important cloud applications in enterprises, the success stories of three real-life cloud applications are presented in Example 4.3 for HTC, news media, and business transactions. The benefits of using cloud services are evident in these SaaS applications.

Example 4.3

Three Success Stories on SaaS Applications

1. To discover new drugs through DNA sequence analysis, Eli Lily Company has used Amazon's AWS platform with provisioned server and storage clusters to conduct high-performance biological sequence analysis without using an expensive supercomputer. The benefit of this IaaS application is reduced drug deployment time with much lower costs.
2. The *New York Times* has applied Amazon's EC2 and S3 services to retrieve useful pictorial information quickly from millions of archival articles and newspapers. The *New York Times* has significantly reduced the time and cost in getting the job done.
3. Pitney Bowes, an e-commerce company, offers clients the opportunity to perform B2B transactions using the Microsoft Azure platform, along with .NET and SQL services. These offerings have significantly increased the company's client base.

4.1.4.3 Mashup of Cloud Services

At the time of this writing, public clouds are in use by a growing number of users. Due to the lack of trust in leaking sensitive data in the business world, more and more enterprises, organizations, and communities are developing private clouds that demand deep customization. An enterprise cloud is used by multiple users within an organization. Each user may build some strategic applications on the cloud, and demands customized partitioning of the data, logic, and database in the metadata representation. More private clouds may appear in the future.

Based on a 2010 Google search survey, interest in grid computing is declining rapidly. *Cloud mashups*have resulted from the need to use multiple clouds simultaneously or in sequence. For example, an industrial supply chain may involve the use of different cloud resources or services at different stages of the chain. Some public repository provides thousands of service APIs and mashups for web commerce services. Popular APIs are provided by Google Maps, Twitter, YouTube, Amazon eCommerce, Salesforce.com, etc.

4.3 Architectural Design of Compute and Storage Clouds

This section presents basic cloud design principles. We start with basic cloud architecture to process massive amounts of data with a high degree of parallelism. Then we study

virtualization support, resource provisioning, infrastructure management, and performance modeling.

4.3.1 A Generic Cloud Architecture Design

An Internet cloud is envisioned as a public cluster of servers provisioned on demand to perform collective web services or distributed applications using data-center resources. In this section, we will discuss cloud design objectives and then present a basic cloud architecture design.

4.3.1.1 Cloud Platform Design Goals

Scalability, virtualization, efficiency, and reliability are four major design goals of a cloud computing platform. Clouds support Web 2.0 applications. Cloud management receives the user request, finds the correct resources, and then calls the provisioning services which invoke the resources in the cloud. The cloud management software needs to support both physical and virtual machines. Security in shared resources and shared access of data centers also pose another design challenge.

The platform needs to establish a very large-scale HPC infrastructure. The hardware and software systems are combined to make it easy and efficient to operate. System scalability can benefit from cluster architecture. If one service takes a lot of processing power, storage capacity, or network traffic, it is simple to add more servers and bandwidth. System reliability can benefit from this architecture. Data can be put into multiple locations. For example, user e-mail can be put in three disks which expand to different geographically separate data centers. In such a situation, even if one of the data centers crashes, the user data is still accessible. The scale of the cloud architecture can be easily expanded by adding more servers and enlarging the network connectivity accordingly.

4.3.1.2 Enabling Technologies for Clouds

The key driving forces behind cloud computing are the ubiquity of broadband and wireless networking, falling storage costs, and progressive improvements in Internet computing software. Cloud users are able to demand more capacity at peak demand, reduce costs, experiment with new services, and remove unneeded capacity, whereas service providers can increase system utilization via multiplexing, virtualization, and dynamic resource provisioning. Clouds are enabled by the progress in hardware, software, and networking technologies summarized in Table 4.3.

Table 4.3

Cloud-Enabling Technologies in Hardware, Software, and Networking

Technology	Requirements and Benefits
Fast platform deployment	Fast, efficient, and flexible deployment of cloud resources to provide dynamic computing environment to users
Virtual clusters on demand	Virtualized cluster of VMs provisioned to satisfy user demand and virtual cluster reconfigured as workload changes

Technology	Requirements and Benefits
Multitenant techniques	SaaS for distributing software to a large number of users for their simultaneous use and resource sharing if so desired
Massive data processing	Internet search and web services which often require massive data processing, especially to support personalized services
Web-scale communication	Support for e-commerce, distance education, telemedicine, social networking, digital government, and digital entertainment applications
Distributed storage	Large-scale storage of personal records and public archive information which demands distributed storage over the clouds
Licensing and billing services	License management and billing services which greatly benefit all types of cloud services in utility computing

These technologies play instrumental roles in making cloud computing a reality. Most of these technologies are mature today to meet increasing demand. In the hardware area, the rapid progress in multicore CPUs, memory chips, and disk arrays has made it possible to build faster data centers with huge amounts of storage space. Resource virtualization enables rapid cloud deployment and disaster recovery. *Service-oriented architecture* (SOA) also plays a vital role.

Progress in providing SaaS, Web 2.0 standards, and Internet performance have all contributed to the emergence of cloud services. Today's clouds are designed to serve a large number of tenants over massive volumes of data. The availability of large-scale, distributed storage systems is the foundation of today's data centers. Of course, cloud computing is greatly benefitted by the progress made in license management and automatic billing techniques in recent years.

4.3.1.3 A Generic Cloud Architecture

Figure 4.14 shows a security-aware cloud architecture. The Internet cloud is envisioned as a massive cluster of servers. These servers are provisioned on demand to perform collective web services or distributed applications using data-center resources. The cloud platform is formed dynamically by provisioning or deprovisioning servers, software, and database resources. Servers in the cloud can be physical machines or VMs. User interfaces are applied to request services. The provisioning tool carves out the cloud system to deliver the requested service.

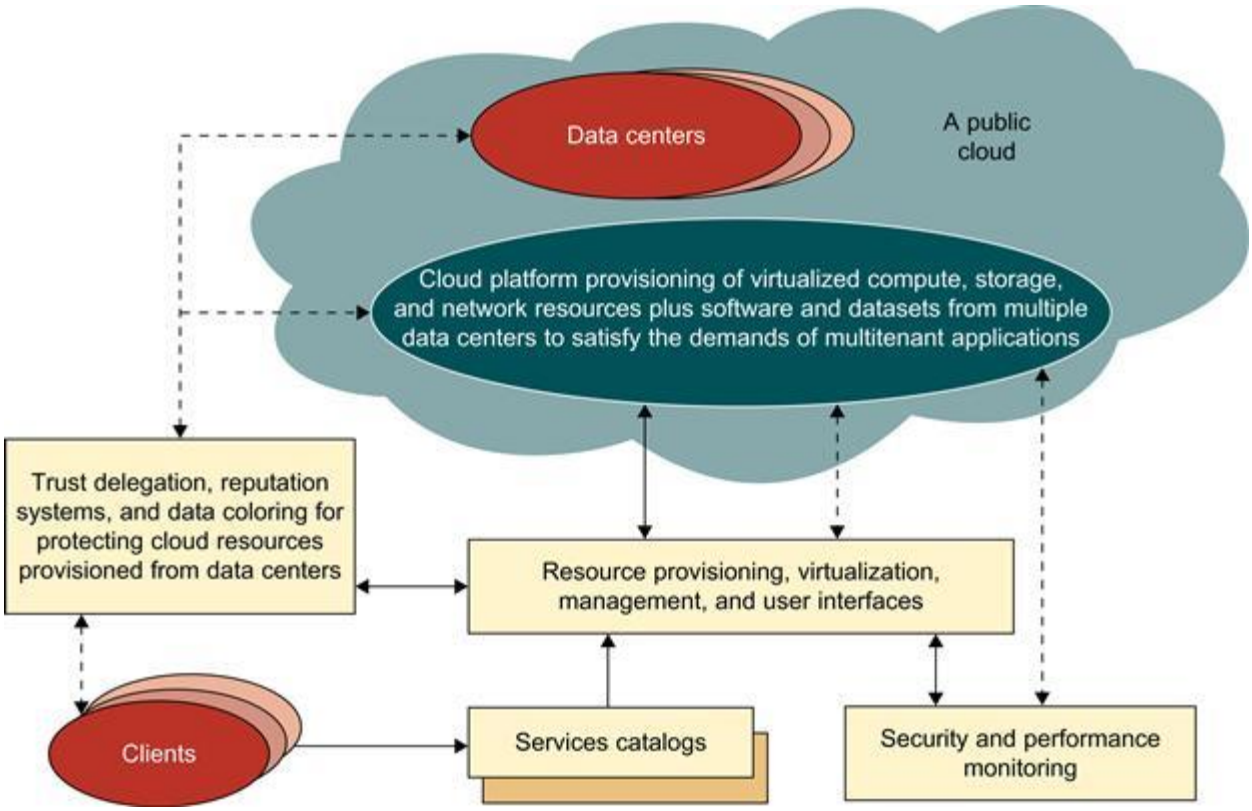


FIGURE 4.14 A security-aware cloud platform built with a virtual cluster of VMs, storage, and networking resources over the data-center servers operated by providers Courtesy of K. Hwang and D. Li, 2010 [36]

In addition to building the server cluster, the cloud platform demands distributed storage and accompanying services. The cloud computing resources are built into the data centers, which are typically owned and operated by a third-party provider. Consumers do not need to know the underlying technologies. In a cloud, software becomes a service. The cloud demands a high degree of trust of massive amounts of data retrieved from large data centers. We need to build a framework to process large-scale data stored in the storage system. This demands a distributed file system over the database system. Other cloud resources are added into a cloud platform, including storage area networks (SANs), database systems, firewalls, and security devices. Web service providers offer special APIs that enable developers to exploit Internet clouds. Monitoring and metering units are used to track the usage and performance of provisioned resources.

The software infrastructure of a cloud platform must handle all resource management and do most of the maintenance automatically. Software must detect the status of each node server joining and leaving, and perform relevant tasks accordingly. Cloud computing providers, such as Google and Microsoft, have built a large number of data centers all over the world. Each data center may have thousands of servers. The location of the data center is chosen to reduce power and cooling costs. Thus, the data centers are often built around hydroelectric power. The cloud physical platform builder is more concerned about the performance/price ratio and reliability issues than shear speed performance.

In general, private clouds are easier to manage, and public clouds are easier to access. The trends in cloud development are that more and more clouds will be hybrid. This is because many cloud applications must go beyond the boundary of an intranet. One must learn how to create a private cloud and how to interact with public clouds in the open Internet. Security

becomes a critical issue in safeguarding the operation of all cloud types. We will study cloud security and privacy issues at the end of this chapter.

4.3.2 Layered Cloud Architectural Development

The architecture of a cloud is developed at three layers: infrastructure, platform, and application, as demonstrated in Figure 4.15. These three development layers are implemented with virtualization and standardization of hardware and software resources provisioned in the cloud. The services to public, private, and hybrid clouds are conveyed to users through networking support over the Internet and intranets involved. It is clear that the infrastructure layer is deployed first to support IaaS services. This infrastructure layer serves as the foundation for building the platform layer of the cloud for supporting PaaS services. In turn, the platform layer is a foundation for implementing the application layer for SaaS applications. Different types of cloud services demand application of these resources separately.

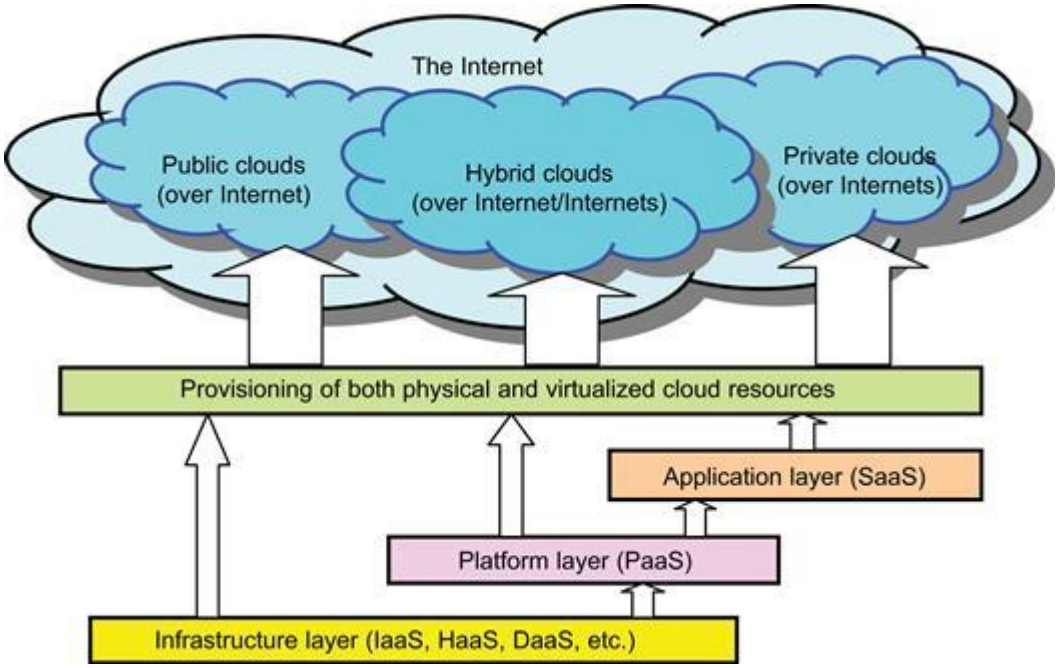


FIGURE 4.15 Layered architectural development of the cloud platform for IaaS, PaaS, and SaaS applications over the Internet.

The infrastructure layer is built with virtualized compute, storage, and network resources. The abstraction of these hardware resources is meant to provide the flexibility demanded by users. Internally, virtualization realizes automated provisioning of resources and optimizes the infrastructure management process. The platform layer is for general-purpose and repeated usage of the collection of software resources. This layer provides users with an environment to develop their applications, to test operation flows, and to monitor execution results and performance. The platform should be able to assure users that they have scalability, dependability, and security protection. In a way, the virtualized cloud platform serves as a "system middleware" between the infrastructure and application layers of the cloud.

The application layer is formed with a collection of all needed software modules for SaaS applications. Service applications in this layer include daily office management work, such as information retrieval, document processing, and calendar and authentication services. The

application layer is also heavily used by enterprises in business marketing and sales, consumer relationship management (CRM), financial transactions, and supply chain management. It should be noted that not all cloud services are restricted to a single layer. Many applications may apply resources at mixed layers. After all, the three layers are built from the bottom up with a dependence relationship.

From the provider's perspective, the services at various layers demand different amounts of functionality support and resource management by providers. In general, SaaS demands the most work from the provider, PaaS is in the middle, and IaaS demands the least. For example, Amazon EC2 provides not only virtualized CPU resources to users, but also management of these provisioned resources. Services at the application layer demand more work from providers. The best example of this is the Salesforce.com CRM service, in which the provider supplies not only the hardware at the bottom layer and the software at the top layer, but also the platform and software tools for user application development and monitoring.

4.3.2.1 Market-Oriented Cloud Architecture

As consumers rely on cloud providers to meet more of their computing needs, they will require a specific level of QoS to be maintained by their providers, in order to meet their objectives and sustain their operations. Cloud providers consider and meet the different QoS parameters of each individual consumer as negotiated in specific SLAs. To achieve this, the providers cannot deploy traditional system-centric resource management architecture. Instead, market-oriented resource management is necessary to regulate the supply and demand of cloud resources to achieve market equilibrium between supply and demand.

The designer needs to provide feedback on economic incentives for both consumers and providers. The purpose is to promote QoS-based resource allocation mechanisms. In addition, clients can benefit from the potential cost reduction of providers, which could lead to a more competitive market, and thus lower prices. Figure 4.16 shows the high-level architecture for supporting market-oriented resource allocation in a cloud computing environment. This cloud is basically built with the following entities:

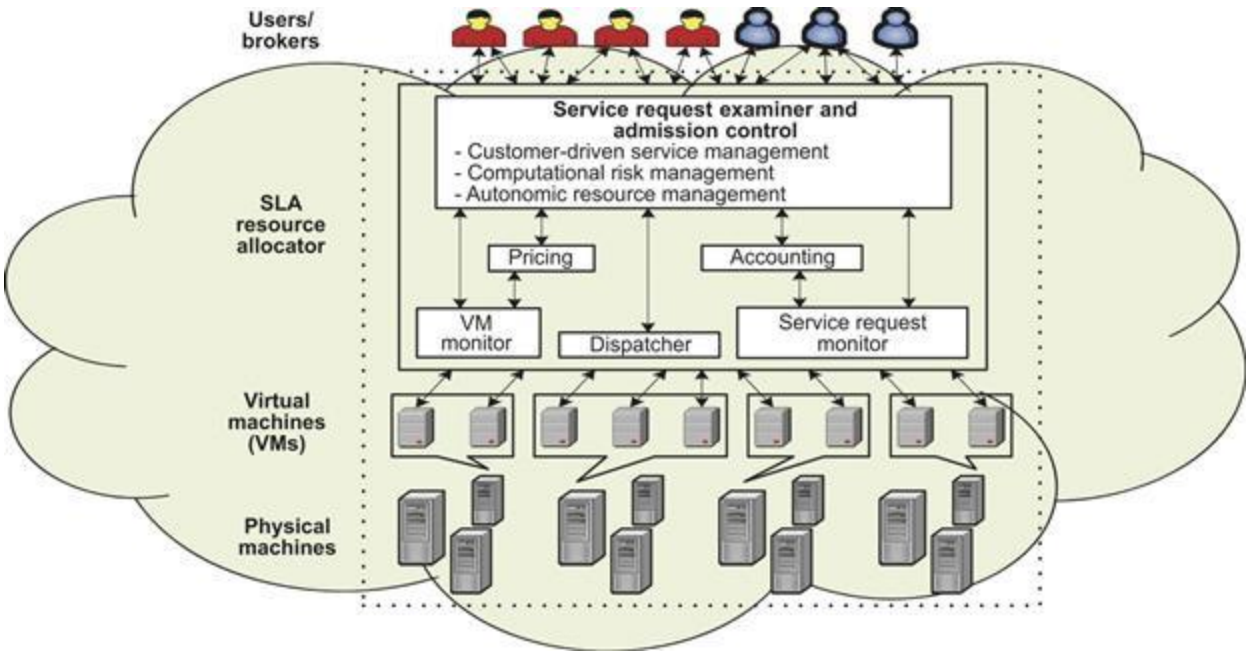


FIGURE 4.16 Market-oriented cloud architecture to expand/shrink leasing of resources with variation in QoS/demand from users Courtesy of Raj Buyya, et al. [11]

Users or brokers acting on user's behalf submit service requests from anywhere in the world to the data center and cloud to be processed. The SLA resource allocator acts as the interface between the data center/cloud service provider and external users/brokers. It requires the interaction of the following mechanisms to support SLA-oriented resource management. When a service request is first submitted the service request examiner interprets the submitted request for QoS requirements before determining whether to accept or reject the request.

The request examiner ensures that there is no overloading of resources whereby many service requests cannot be fulfilled successfully due to limited resources. It also needs the latest status information regarding resource availability (from the VM Monitor mechanism) and workload processing (from the Service Request Monitor mechanism) in order to make resource allocation decisions effectively. Then it assigns requests to VMs and determines resource entitlements for allocated Vms.

The Pricing mechanism decides how service requests are charged. For instance, requests can be charged based on submission time (peak/off-peak), pricing rates (fixed/changing), or availability of resources (supply/demand). Pricing serves as a basis for managing the supply and demand of computing resources within the data center and facilitates in prioritizing resource allocations effectively. The Accounting mechanism maintains the actual usage of resources by requests so that the final cost can be computed and charged to users. In addition, the maintained historical usage information can be utilized by the Service Request Examiner and Admission Control mechanism to improve resource allocation decisions.

The VM Monitor mechanism keeps track of the availability of VMs and their resource entitlements. The Dispatcher mechanism starts the execution of accepted service requests on allocated VMs. The Service Request Monitor mechanism keeps track of the execution progress of service requests. Multiple VMs can be started and stopped on demand on a single physical machine to meet accepted service requests, hence providing maximum flexibility to configure various partitions of resources on the same physical machine to different specific requirements of service requests. In addition, multiple VMs can concurrently run applications based on different operating system environments on a single physical machine since the VMs are isolated from one another on the same physical machine.

4.3.2.2 Quality of Service Factors

The data center comprises multiple computing servers that provide resources to meet service demands. In the case of a cloud as a commercial offering to enable crucial business operations of companies, there are critical QoS parameters to consider in a service request, such as time, cost, reliability, and trust/security. In particular, QoS requirements cannot be static and may change over time due to continuing changes in business operations and operating environments. In short, there should be greater importance on customers since they pay to access services in clouds. In addition, the state of the art in cloud computing has no or limited support for dynamic negotiation of SLAs between participants and mechanisms for automatic allocation of resources to multiple competing requests. Negotiation mechanisms are needed to respond to alternate offers protocol for establishing SLAs [72].

Commercial cloud offerings must be able to support customer-driven service management based on customer profiles and requested service requirements. Commercial clouds define computational risk management tactics to identify, assess, and manage risks involved in the execution of applications with regard to service requirements and customer needs. The cloud also derives appropriate market-based resource management strategies that encompass both customer-driven service management and computational risk management to sustain SLA-oriented resource allocation. The system incorporates autonomic resource management models that effectively self-manage changes in service requirements to satisfy both new service demands and existing service obligations, and leverage VM technology to dynamically assign resource shares according to service requirements.

4.3.3 Virtualization Support and Disaster Recovery

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

In addition, in cloud computing, virtualization also means the resources and fundamental infrastructure are virtualized. The user will not care about the computing resources that are used for providing the services. Cloud users do not need to know and have no way to discover physical resources that are involved while processing a service request. Also, application developers do not care about some infrastructure issues such as scalability and fault tolerance (i.e., they are virtualized). Application developers focus on service logic. Figure 4.17 shows the infrastructure needed to virtualize the servers in a data center for implementing specific cloud applications.

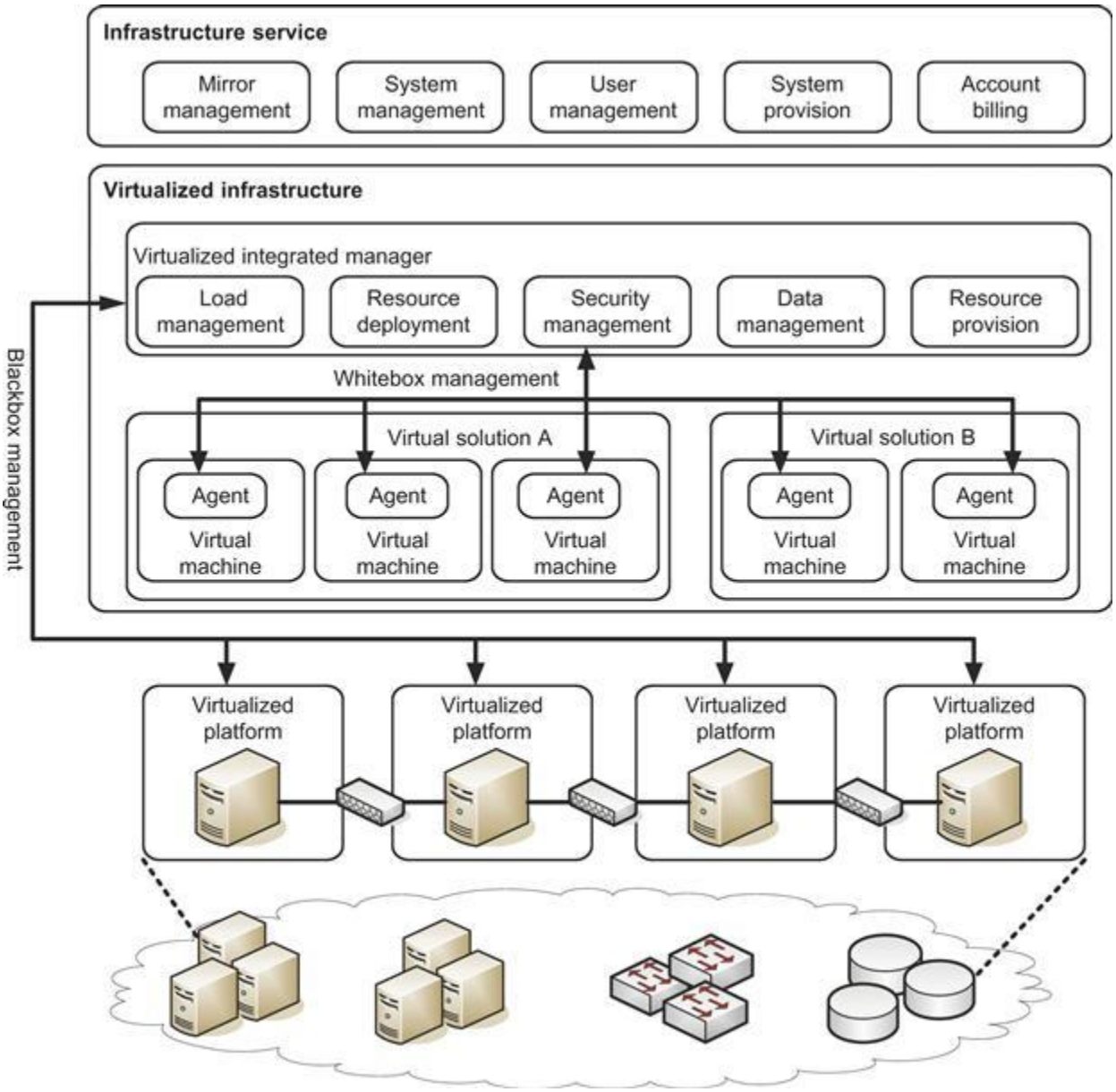


FIGURE 4.17 Virtualized servers, storage, and network for cloud platform construction Courtesy of Zhong-Yuan Qin, SouthEast University, China

4.3.3.1 Hardware Virtualization

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

Some cloud computing providers have used virtualization technology to provide this service for developers. As mentioned before, system virtualization software is considered the hardware analog mechanism to run an unmodified operating system, usually on bare hardware directly, on top of software. Table 4.4 lists some of the system virtualization software in wide use at the time of this writing. Currently, the VMs installed on a cloud

computing platform are mainly used for hosting third-party programs. VMs provide flexible runtime services to free users from worrying about the system environment.
Using VMs in a cloud computing platform ensures extreme flexibility for users. As the computing resources are shared by many users, a method is required to maximize the users' privileges and still keep them separated safely. Traditional sharing of cluster resources depends on the user and group mechanism on a system. Such sharing is not flexible. Users cannot customize the system for their special purposes. Operating systems cannot be changed. The separation is not complete. An environment that meets one user's requirements often cannot satisfy another user. Virtualization allows users to have full privileges while keeping them separate.

Users have full access to their own VMs, which are completely separate from other users' VMs. Multiple VMs can be mounted on the same physical server. Different VMs may run with different OSes. We also need to establish the virtual disk storage and virtual networks needed by the VMs. The virtualized resources form a resource pool. The virtualization is carried out by special servers dedicated to generating the virtualized resource pool. The virtualized infrastructure (black box in the middle) is built with many *virtualizing integration managers*. These managers handle loads, resources, security, data, and provisioning functions. Figure 4.18 shows two VM platforms. Each platform carries out a virtual solution to a user job. All cloud services are managed in the boxes at the top.

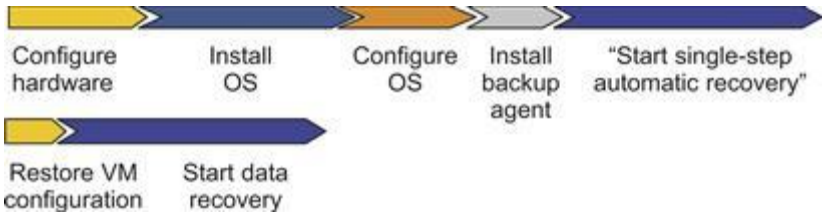


FIGURE 4.18 Recovery overhead of a conventional disaster recovery scheme, compared with that required to recover from live migration of VMs.

4.3.3.2 Virtualization Support in Public Clouds

Armbrust, et al. [4] have assessed in Table 4.4 three public clouds in the context of virtualization support: AWS, Microsoft Azure, and GAE. AWS provides extreme flexibility (VMs) for users to execute their own applications. GAE provides limited application-level virtualization for users to build applications only based on the services that are created by Google. Microsoft provides programming-level virtualization (.NET virtualization) for users to build their applications.

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

Provider	AWS	Microsoft Azure	GAE
Compute cloud with virtual cluster of servers	x86 instruction set, Xen VMs, resource elasticity allows scalability through virtual cluster, or a third party such as RightScale must provide the cluster	Common language runtime VMs provisioned by declarative descriptions	Predefined application framework handlers written in Python, automatic scaling up and down, server failover inconsistent with the web applications
Storage cloud with virtual storage	Models for block store (EBS) and augmented key/blob store (SimpleDB), automatic scaling varies from EBS to fully automatic (SimpleDB, S3)	SQL Data Services (restricted view of SQL Server), Azure storage service	MegaStore/BigTable
Network cloud services	Declarative IP-level topology; placement details hidden, security groups restricting communication, availability zones isolate network failure, elastic IP applied	Automatic with user's declarative descriptions or roles of app. components	Fixed topology to accommodate three-tier web app. structure, scaling up and down is automatic and programmer-invisible

The VMware tools apply to workstations, servers, and virtual infrastructure. The Microsoft tools are used on PCs and some special servers. The XenEnterprise tool applies only to Xen-based servers. Everyone is interested in the cloud; the entire IT industry is moving toward the vision of the cloud. Virtualization leads to HA, disaster recovery, dynamic load leveling, and rich provisioning support. Both cloud computing and utility computing leverage the benefits of virtualization to provide a scalable and autonomous computing environment.

4.3.3.3 Storage Virtualization for Green Data Centers

IT power consumption in the United States has more than doubled to 3 percent of the total energy consumed in the country. The large number of data centers in the country has contributed to this energy crisis to a great extent. More than half of the companies in the Fortune 500 are actively implementing new corporate energy policies. Recent surveys from both IDC and Gartner confirm the fact that virtualization had a great impact on cost reduction from reduced power consumption in physical computing systems. This alarming situation has made the IT industry become more energy-aware. With little evolution of alternate energy resources, there is an imminent need to conserve power in all computers. Virtualization and server consolidation have already proven handy in this aspect. Green data centers and benefits of storage virtualization are considered to further strengthen the synergy of green computing.

4.3.3.4 Virtualization for IaaS

VM technology has increased in ubiquity. This has enabled users to create customized environments atop physical infrastructure for cloud computing. Use of VMs in clouds has the following distinct benefits: (1) System administrators consolidate workloads of underutilized servers in fewer servers; (2) VMs have the ability to run legacy code without interfering with other APIs; (3) VMs can be used to improve security through creation of sandboxes for running applications with questionable reliability; And (4) virtualized cloud platforms can apply performance isolation, letting providers offer some guarantees and better QoS to customer applications.

4.3.3.5 VM Cloning for Disaster Recovery

VM technology requires an advanced disaster recovery scheme. One scheme is to recover one physical machine by another physical machine. The second scheme is to recover one VM by another VM. As shown in the top timeline of Figure 4.18, traditional disaster recovery from one physical machine to another is rather slow, complex, and expensive. Total recovery time is attributed to the hardware configuration, installing and configuring the OS, installing the backup agents, and the long time to restart the physical machine. To recover a VM platform, the installation and configuration times for the OS and backup agents are eliminated. Therefore, we end up with a much shorter disaster recovery time, about 40 percent of that to recover the physical machines. Virtualization aids in fast disaster recovery by VM encapsulation.

We discussed disaster recovery in Chapters 2 and 3. The cloning of VMs offers an effective solution. The idea is to make a clone VM on a remote server for every running VM on a local server. Among all the clone VMs, only one needs to be active. The remote VM should be in a suspended mode. A cloud control center should be able to activate this clone VM in case of failure of the original VM, taking a snapshot of the VM to enable live migration in a minimal amount of time. The migrated VM can run on a shared Internet connection. Only updated data and modified states are sent to the suspended VM to update its state. The *Recovery Property Objective (RPO)* and *Recovery Time Objective (RTO*) are affected by the number of snapshots taken. Security of the VMs should be enforced during live migration of VMs.

4.3.4 Architectural Design Challenges

In this section, we will identify six open challenges in cloud architecture development. Armbrust, et al.[4] have observed some of these topics as both obstacles and opportunities. Plausible solutions to meet these challenges are discussed shortly.

4.3.4.1 Challenge 1—Service Availability and Data Lock-in Problem

The management of a cloud service by a single company is often the source of single points of failure. To achieve HA, one can consider using multiple cloud providers. Even if a company has multiple data centers located in different geographic regions, it may have common software infrastructure and accounting systems. Therefore, using multiple cloud providers may provide more protection from failures. Another availability obstacle is distributed *denial of service* (DDoS) attacks. Criminals threaten to cut off the incomes of SaaS providers by making their services unavailable. Some utility computing services offer SaaS providers the opportunity to defend against DDoS attacks by using quick scale-ups.

Software stacks have improved interoperability among different cloud platforms, but the APIs itself are still proprietary. Thus, customers cannot easily extract their data and programs from one site to run on another. The obvious solution is to standardize the APIs so that a SaaS developer can deploy services and data across multiple cloud providers. This will rescue the loss of all data due to the failure of a single company. In addition to mitigating data lock-in concerns, standardization of APIs enables a new usage model in which the same software infrastructure can be used in both public and private clouds. Such an option could enable "surge computing," in which the public cloud is used to capture the extra tasks that cannot be easily run in the data center of a private cloud.

4.3.4.2 Challenge 2—Data Privacy and Security Concerns

Current cloud offerings are essentially public (rather than private) networks, exposing the system to more attacks. Many obstacles can be overcome immediately with well-understood technologies such as encrypted storage, virtual LANs, and network middleboxes (e.g., firewalls, packet filters). For example, you could encrypt your data before placing it in a cloud. Many nations have laws requiring SaaS providers to keep customer data and copyrighted material within national boundaries.

Traditional network attacks include buffer overflows, DoS attacks, spyware, malware, rootkits, Trojan horses, and worms. In a cloud environment, newer attacks may result from hypervisor malware, guest hopping and hijacking, or VM rootkits. Another type of attack is the man-in-the-middle attack for VM migrations. In general, passive attacks steal sensitive data or passwords. Active attacks may manipulate kernel data structures which will cause major damage to cloud servers. We will study all of these security and privacy problems on clouds in Section 4.5.

4.3.4.3 Challenge 3—Unpredictable Performance and Bottlenecks

Multiple VMs can share CPUs and main memory in cloud computing, but I/O sharing is problematic. For example, to run 75 EC2 instances with the STREAM benchmark requires a mean bandwidth of 1,355 MB/second. However, for each of the 75 EC2 instances to write 1 GB files to the local disk requires a mean disk write bandwidth of only 55 MB/second. This demonstrates the problem of I/O interference between VMs. One solution is to improve I/O architectures and operating systems to efficiently virtualize interrupts and I/O channels.

Internet applications continue to become more data-intensive. If we assume applications to be "pulled apart" across the boundaries of clouds, this may complicate data placement and transport. Cloud users and providers have to think about the implications of placement and traffic at every level of the system, if they want to minimize costs. This kind of reasoning can be seen in Amazon's development of its new CloudFront service. Therefore, data transfer bottlenecks must be removed, bottleneck links must be widened, and weak servers should be removed. We will study performance issues in Chapter 8.

4.3.4.4 Challenge 4—Distributed Storage and Widespread Software Bugs

The database is always growing in cloud applications. The opportunity is to create a storage system that will not only meet this growth, but also combine it with the cloud advantage of scaling arbitrarily up and down on demand. This demands the design of efficient distributed SANs. Data centers must meet programmers' expectations in terms of scalability, data durability, and HA. Data consistence checking in SAN-connected data centers is a major challenge in cloud computing.

Large-scale distributed bugs cannot be reproduced, so the debugging must occur at a scale in the production data centers. No data center will provide such a convenience. One solution may be a reliance on using VMs in cloud computing. The level of virtualization may make it possible to capture valuable information in ways that are impossible without using VMs. Debugging over simulators is another approach to attacking the problem, if the simulator is well designed.

4.3.4.5 Challenge 5—Cloud Scalability, Interoperability, and Standardization

The pay-as-you-go model applies to storage and network bandwidth; both are counted in terms of the number of bytes used. Computation is different depending on virtualization level. GAE automatically scales in response to load increases and decreases; users are charged by the cycles used. AWS charges by the hour for the number of VM instances used, even if the machine is idle. The opportunity here is to scale quickly up and down in response to load variation, in order to save money, but without violating SLAs.
Open Virtualization Format (OVF) describes an open, secure, portable, efficient, and extensible format for the packaging and distribution of VMs. It also defines a format for distributing software to be deployed in VMs. This VM format does not rely on the use of a specific host platform, virtualization platform, or guest operating system. The approach is to address virtual platform-agnostic packaging with certification and integrity of packaged software. The package supports virtual appliances to span more than one VM.

OVF also defines a transport mechanism for VM templates, and can apply to different virtualization platforms with different levels of virtualization. In terms of cloud standardization, we suggest the ability for virtual appliances to run on any virtual platform. We also need to enable VMs to run on heterogeneous hardware platform hypervisors. This requires hypervisor-agnostic VMs. We also need to realize cross-platform live migration between x86 Intel and AMD technologies and support legacy hardware for load balancing. All these issue are wide open for further research.

4.3.4.6 Challenge 6—Software Licensing and Reputation Sharing

Many cloud computing providers originally relied on open source software because the licensing model for commercial software is not ideal for utility computing. The primary opportunity is either for open source to remain popular or simply for commercial software companies to change their licensing structure to better fit cloud computing. One can consider using both pay-for-use and bulk-use licensing schemes to widen the business coverage.

One customer's bad behavior can affect the reputation of the entire cloud. For instance, blacklisting of EC2 IP addresses by spam-prevention services may limit smooth VM installation. An opportunity would be to create reputation-guarding services similar to the "trusted e-mail" services currently offered (for a fee) to services hosted on smaller ISPs. Another legal issue concerns the transfer of legal liability. Cloud providers want legal liability to remain with the customer, and vice versa. This problem must be solved at the SLA level. We will study reputation systems for protecting data centers in the next section.

CHAPTER 4

Cloud Platform Architecture over Virtualized Data Centers

CHAPTER OUTLINE

4.4 Public Cloud Platforms: GAE, AWS, and Azure

In this section, we will review the system architectures of four commercially available cloud platforms. These case studies will prepare readers for subsequent sections and chapters.

4.4.1 Public Clouds and Service Offerings

Cloud services are demanded by computing and IT administrators, software vendors, and end users. Figure 4.19 introduces five levels of cloud players. At the top level, individual users and organizational users demand very different services. The application providers at the SaaS level serve mainly individual users. Most business organizations are serviced by IaaS and PaaS providers. The infrastructure services (IaaS) provide compute, storage, and communication resources to both applications and organizational users. The cloud environment is defined by the PaaS or platform providers. Note that the platform providers support both infrastructure services and organizational users directly.

Cloud services rely on new advances in machine virtualization, SOA, grid infrastructure management, and power efficiency. Consumers purchase such services in the form of IaaS, PaaS, or SaaS as described earlier. Also, many cloud entrepreneurs are selling value-added utility services to massive numbers of users. The cloud industry leverages the growing demand by many enterprises and business users to outsource their computing and storage jobs to professional providers.

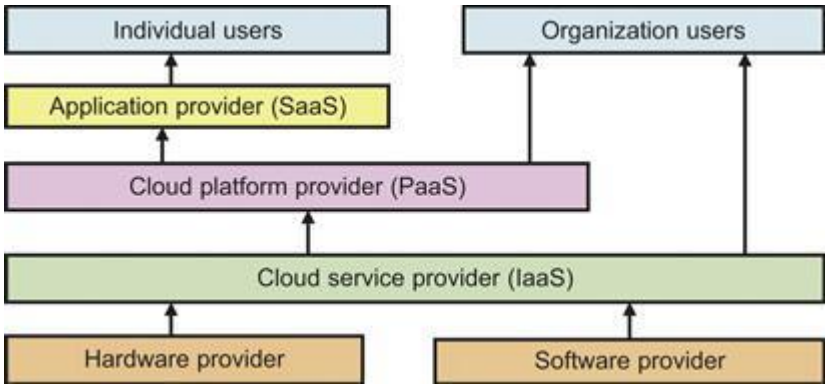


FIGURE 4.19 Roles of individual and organizational users and their interaction with cloud providers under various cloud service models.

The provider service charges are often much lower than the cost for users to replace their obsolete servers frequently. Table 4.5 summarizes the profiles of five major cloud providers by 2010 standards.

Table 4.5 : Five Major Cloud Platforms and Their Service Offerings [36]

Model	IBM	Amazon	Google	Microsoft	Salesforce
PaaS	BlueCloud, WCA, RC2		App Engine (GAE)	Windows Azure	Force.com
IaaS	Ensembles	AWS		Windows Azure	
SaaS	Lotus Live		Gmail, Docs	.NET service, Dynamic CRM	Online CRM, Gifttag
Virtualization		OS and Xen	Application Container	OS level/ Hypel-V	
Service Offerings	SOA, B2, TSAM, RAD, Web 2.0	EC2, S3, SQS, SimpleDB	GFS, Chubby, BigTable, MapReduce	Live, SQL Hotmail	Apex, visual force, record security
Security Features	WebSphere2 and PowerVM tuned for protection	PKI, VPN, EBS to recover from failure	Chubby locks for security enforcement	Replicated data, rule-based access control	Admin./record security, uses metadata API
User Interfaces		EC2 command-line tools	Web-based admin. console	Windows Azure portal	
Web API	Yes	Yes	Yes	Yes	Yes
Programming Support	AMI		Python	.NET Framework	

Note: WCA: WebSphere CloudBurst Appliance; RC2: Research Compute Cloud; RAD: Rational Application Developer; SOA: Service-Oriented Architecture; TSAM: Tivoli Service Automation Manager; EC2: Elastic Compute Cloud; S3: Simple Storage Service; SQS: Simple Queue Service; GAE: Google App Engine; AWS: Amazon Web Services; SQL: Structured Query Language; EBS: Elastic Block Store; CRM: Consumer Relationship Management.

Amazon pioneered the IaaS business in supporting e-commerce and cloud applications by millions of customers simultaneously. The elasticity in the Amazon cloud comes from the flexibility provided by the hardware and software services. EC2 provides an environment for running virtual servers on demand. S3 provides unlimited online storage space. Both EC2 and S3 are supported in the AWS platform. Microsoft offers the Azure platform for cloud applications. It has also supported the .NET service, dynamic CRM, Hotmail, and SQL applications. Salsforce.com offers extensive SaaS applications for online CRM applications using its Force.com platforms.

As Table 4.5 shows, all IaaS, PaaS, and SaaS models allow users to access services over the Internet, relying entirely on the infrastructures of the cloud service providers. These models are offered based on various SLAs between the providers and the users. SLAs are more common in network services as they account for the QoS characteristics of network services. For cloud computing services, it is difficult to find a reasonable precedent for negotiating an SLA. In a broader sense, the SLAs for cloud computing address service availability, data integrity, privacy, and security protection. Blank spaces in the table refer to unknown or underdeveloped features.

4.4.2 Google App Engine (GAE)

Google has the world's largest search engine facilities. The company has extensive experience in massive data processing that has led to new insights into data-center design (see Chapter 3) and novel programming models that scale to incredible sizes. The Google platform is based on its search engine expertise, but as discussed

earlier with MapReduce, this infrastructure is applicable to many other areas. Google has hundreds of data centers and has installed more than 460,000 servers worldwide. For example, 200 Google data centers are used at one time for a number of cloud applications. Data items are stored in text, images, and video and are replicated to tolerate faults or failures. Here we discuss Google's App Engine (GAE) which offers a PaaS platform supporting various cloud and web applications.

4.4.2.1 Google Cloud Infrastructure

Google has pioneered cloud development by leveraging the large number of data centers it operates. For example, Google pioneered cloud services in Gmail, Google Docs, and Google Earth, among other applications. These applications can support a large number of users simultaneously with HA. Notable technology achievements include the Google File System (GFS), MapReduce, BigTable, and Chubby. In 2008, Google announced the GAE web application platform which is becoming a common platform for many small cloud service providers. This platform specializes in supporting scalable (elastic) web applications. GAE enables users to run their applications on a large number of data centers associated with Google's search engine operations.

4.4.2.2 GAE Architecture

Figure 4.20 shows the major building blocks of the Google cloud platform which has been used to deliver the cloud services highlighted earlier. GFS is used for storing large amounts of data. MapReduce is for use in application program development. Chubby is used for distributed application lock services. BigTable offers a storage service for accessing structured data. These technologies are described in more detail in Chapter 8. Users can interact with Google applications via the web interface provided by each application. Third-party application providers can use GAE to build cloud applications for providing services. The applications all run in data centers under tight management by Google engineers. Inside each data center, there are thousands of servers forming different clusters.

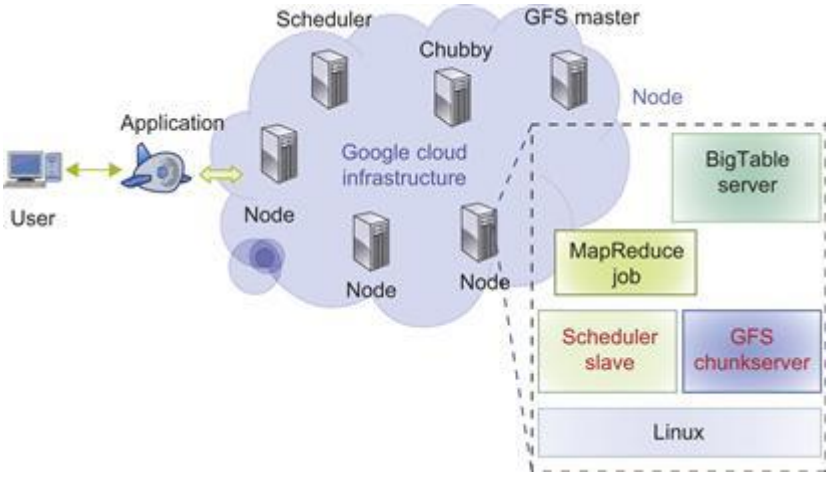


FIGURE 4.20 Google cloud platform and major building blocks, the blocks shown are large clusters of low-cost servers Courtesy of Kang Chen, Tsinghua University, China

Google is one of the larger cloud application providers, although its fundamental service program is private and outside people cannot use the Google infrastructure to build their own service. The building blocks of Google's cloud computing application include the Google File System for storing large amounts of data, the MapReduce programming framework for application developers, Chubby for distributed application lock services, and BigTable as a storage service for accessing structural or semistructural data. With these building blocks, Google has built many cloud applications. Figure 4.20shows the overall architecture of the Google cloud infrastructure. A typical cluster configuration can run the Google File System, MapReduce jobs, and BigTable servers for structure data. Extra services such as Chubby for distributed locks can also run in the clusters.

GAE runs the user program on Google's infrastructure. As it is a platform running third-party programs, application developers now do not need to worry about the maintenance of servers. GAE can be thought of as the combination of several software components. The frontend is an application framework which is similar to other web application frameworks such as ASP, J2EE, and JSP. At the time of this writing, GAE supports Python and Java programming environments. The applications can run similar to web application containers.

The frontend can be used as the dynamic web serving infrastructure which can provide the full support of common technologies.

4.4.2.3 Functional Modules of GAE

The GAE platform comprises the following five major components. The GAE is not an infrastructure platform, but rather an application development platform for users. We describe the component functionalities separately.

a. The *datastore* offers object-oriented, distributed, structured data storage services based on BigTable techniques. The datastore secures data management operations.

b. The *application runtime environment* offers a platform for scalable web programming and execution. It supports two development languages: Python and Java.

c. The *software development kit* (SDK) is used for local application development. The SDK allows users to execute test runs of local applications and upload application code.

d. The *administration console* is used for easy management of user application development cycles, instead of for physical resource management.

e. The *GAE web service infrastructure* provides special interfaces to guarantee flexible use and management of storage and network resources by GAE.

Google offers essentially free GAE services to all Gmail account owners. You can register for a GAE account or use your Gmail account name to sign up for the service. The service is free within a quota. If you exceed the quota, the page instructs you on how to pay for the service. Then you download the SDK and read the Python or Java guide to get started. Note that GAE only accepts Python, Ruby, and Java programming languages. The platform does not provide any IaaS services, unlike Amazon, which offers Iaas and PaaS. This model allows the user to deploy user-built applications on top of the cloud infrastructure that are built using the programming languages and software tools supported by the provider (e.g., Java, Python). Azure does this similarly for .NET. The user does not manage the underlying cloud infrastructure. The cloud provider facilitates support of application development, testing, and operation support on a well-defined service platform.

4.4.2.4 GAE Applications

Well-known GAE applications include the Google Search Engine, Google Docs, Google Earth, and Gmail. These applications can support large numbers of users simultaneously. Users can interact with Google applications via the web interface provided by each application. Third-party application providers can use GAE to build cloud applications for providing services. The applications are all run in the Google data centers. Inside each data center, there might be thousands of server nodes to form different clusters. (See the previous section.) Each cluster can run multipurpose servers.

GAE supports many web applications. One is a storage service to store application-specific data in the Google infrastructure. The data can be persistently stored in the backend storage server while still providing the facility for queries, sorting, and even transactions similar to traditional database systems. GAE also provides Google-specific services, such as the Gmail account service (which is the login service, that is, applications can use the Gmail account directly). This can eliminate the tedious work of building customized user management components in web applications. Thus, web applications built on top of GAE can use the APIs authenticating users and sending e-mail using Google accounts.

4.4.3 Amazon Web Services (AWS)

VMs can be used to share computing resources both flexibly and safely. Amazon has been a leader in providing public cloud services (http://aws.amazon.com/). Amazon applies the IaaS model in providing its services. Figure 4.21 shows the AWS architecture. EC2 provides the virtualized platforms to the host VMs where the cloud application can run. S3 (Simple Storage Service) provides the object-oriented storage service for users. EBS (Elastic Block Service) provides the block storage interface which can be used to support traditional applications. SQS stands for Simple Queue Service, and its job is to ensure a reliable message service between

two processes. The message can be kept reliably even when the receiver processes are not running. Users can access their objects through SOAP with either browsers or other client programs which support the SOAP standard.

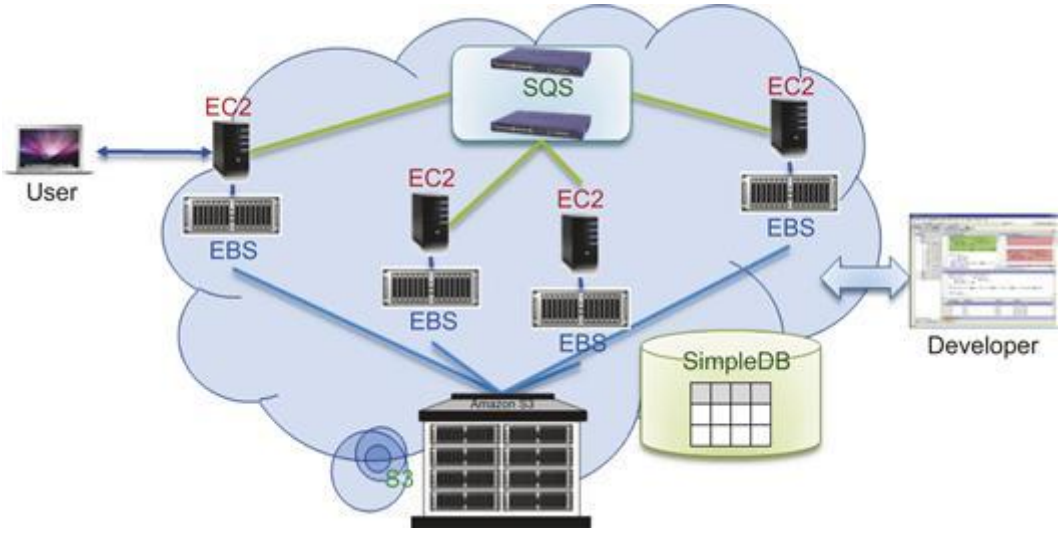


FIGURE 4.21 Amazon cloud computing infrastructure (Key services are identified here; many more are listed in Table 4.6) Courtesy of Kang Chen, Tsinghua University, China

Table 4.6 summarizes the service offerings by AWS in 12 application tracks. Details of EC2, S3, and EBS are available in Chapter 6 where we discuss programming examples. Amazon offers queuing and notification services (SQS and SNS), which are implemented in the AWS cloud. Note brokering systems run very efficiently in clouds and offer a striking model for controlling sensors and providing office support of smartphones and tablets. Different from Google, Amazon provides a more flexible cloud computing platform for developers to build cloud applications. Small and medium-size companies can put their business on the Amazon cloud platform. Using the AWS platform, they can service large numbers of Internet users and make profits through those paid services.

Table 4.6 AWS Offerings in 2011

Service Area	Service Modules and Abbreviated Names
Compute	Elastic Compute Cloud (EC2), Elastic MapReduce, Auto Scaling
Messaging	Simple Queue Service (SQS), Simple Notification Service (SNS)
Storage	Simple Storage Service (S3), Elastic Block Storage (EBS), AWS Import/Export
Content Delivery	Amazon CloudFront
Monitoring	Amazon CloudWatch
Support	AWS Premium Support
Database	Amazon SimpleDB, Relational Database Service (RDS)
Networking	Virtual Private Cloud (VPC) (Example 4.1, Figure 4.6), Elastic Load Balancing
Web Traffic	Alexa Web Information Service, Alexa Web Sites
E-Commerce	Fulfillment Web Service (FWS)
Payments and Billing	Flexible Payments Service (FPS), Amazon DevPay
Workforce	Amazon Mechanical Turk

Courtesy of Amazon, http://aws.amazon.com [3]

ELB automatically distributes incoming application traffic across multiple Amazon EC2 instances and allows user to avoid nonoperating nodes and to equalize load on functioning images. Both auto-scaling and ELB are enabled by CloudWatch which monitors running instances. CloudWatch is a web service that provides monitoring for AWS cloud resources, starting with Amazon EC2. It provides customers with visibility into resource utilization, operational performance, and overall demand patterns, including metrics such as CPU utilization, disk reads and writes, and network traffic.

Amazon (like Azure) offers a *Relational Database Service (RDS)* with a messaging interface to be covered in Section 4.1. The Elastic MapReduce capability is equivalent to Hadoop running on the basic EC2 offering. AWS Import/Export allows one to ship large volumes of data to and from EC2 by shipping physical disks; it is well known that this is often the highest bandwidth connection between geographically distant systems. Amazon CloudFront implements a content distribution network. Amazon DevPay is a simple-to-use online billing and account management service that makes it easy for businesses to sell applications that are built into or run on top of AWS.

FPS provides developers of commercial systems on AWS with a convenient way to charge Amazon's customers that use such services built on AWS. Customers can pay using the same login credentials, shipping address, and payment information they already have on file with Amazon. The FWS allows merchants to access Amazon's fulfillment capabilities through a simple web service interface. Merchants can send order information to Amazon to fulfill customer orders on their behalf. In July 2010, Amazon offered MPI clusters and cluster compute instances. The AWS cluster compute instances use hardware-assisted virtualization instead of the para-virtualization used by other instance types and requires booting from the EBS. Users are freed to create a new AMI as needed.

4.4.4 Microsoft Windows Azure

In 2008, Microsoft launched a Windows Azure platform to meet the challenges in cloud computing. This platform is built over Microsoft data centers. Figure 4.22 shows the overall architecture of Microsoft's cloud platform. The platform is divided into three major component platforms. Windows Azure offers a cloud platform built on Windows OS and based on Microsoft virtualization technology. Applications are installed on VMs deployed on the data-center servers. Azure manages all servers, storage, and network resources of the data center. On top of the infrastructure are the various services for building different cloud applications. Cloud-level services provided by the Azure platform are introduced below. More details on Azure services are given in Chapter 6.

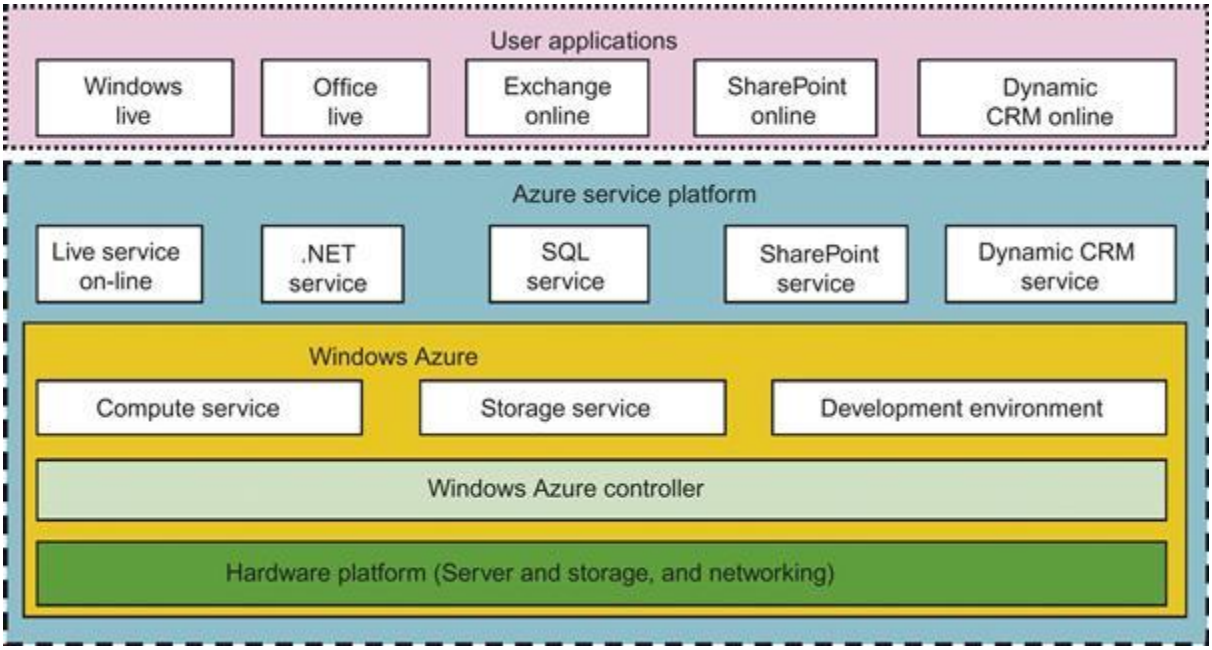


FIGURE 4.22 Microsoft Windows Azure platform for cloud computingCourtesy of Microsoft, 2010, http://www.microsoft.com/windowsazure

• **Live service** Users can visit Microsoft Live applications and apply the data involved across multiple machines concurrently.

• **.NET service** This package supports application development on local hosts and execution on cloud machines.

• **SQL Azure** This function makes it easier for users to visit and use the relational database associated with the SQL server in the cloud.

• **SharePoint service** This provides a scalable and manageable platform for users to develop their special business applications in upgraded web services.

• **Dynamic CRM service** This provides software developers a business platform in managing CRM applications in financing, marketing, and sales and promotions.

All these cloud services in Azure can interact with traditional Microsoft software applications, such as Windows Live, Office Live, Exchange online, SharePoint online, and dynamic CRM online. The Azure platform applies the standard web communication protocols SOAP and REST. The Azure service applications allow users to integrate the cloud application with other platforms or third-party clouds. You can download the Azure development kit to run a local version of Azure. The powerful SDK allows Azure applications to be developed and debugged on the Windows hosts.

4.5 Inter-cloud Resource Management

This section characterizes the various cloud service models and their extensions. The cloud service trends are outlined. Cloud resource management and intercloud resource exchange schemes are reviewed. We will discuss the defense of cloud resources against network threats in Section 4.6.

4.5.1 Extended Cloud Computing Services

Figure 4.23 shows six layers of cloud services, ranging from hardware, network, and collocation to infrastructure, platform, and software applications. We already introduced the top three service layers as SaaS, PaaS, and IaaS, respectively. The cloud platform provides PaaS, which sits on top of the IaaS infrastructure. The top layer offers SaaS. These must be implemented on the cloud platforms provided. Although the three basic models are dissimilar in usage, as shown in Table 4.7, they are built one on top of another. The implication is that one cannot launch SaaS applications with a cloud platform. The cloud platform cannot be built if compute and storage infrastructures are not there.

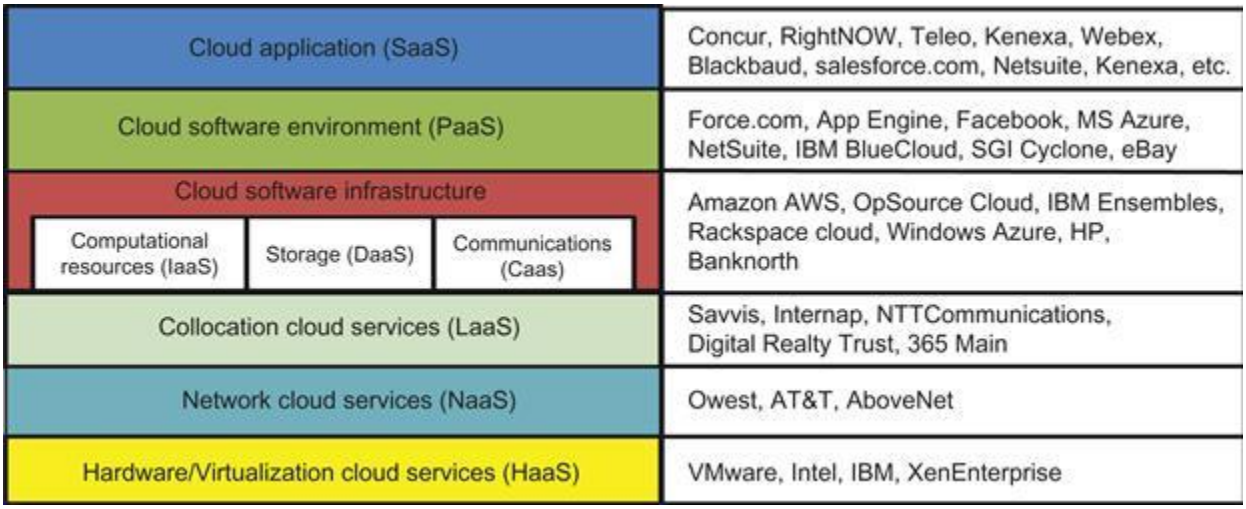


FIGURE 4.23 A stack of six layers of cloud services and their providersCourtesy of T. Chou, Active Book Express, 2010 [16]

The bottom three layers are more related to physical requirements. The bottommost layer provides *Hardware as a Service (HaaS)*. The next layer is for interconnecting all the hardware components, and is simply called *Network as a Service (NaaS)*. *Virtual LANs* fall within the scope of NaaS. The next layer up offers *Location as a Service*

(LaaS), which provides a collocation service to house, power, and secure all the physical hardware and network resources. Some authors say this layer provides *Security as a Service* ("SaaS"). The cloud infrastructure layer can be further subdivided as *Data as a Service (DaaS)*and *Communication as a Service (CaaS)* in addition to compute and storage in IaaS.

We will examine commercial trends in cloud services in subsequent sections. Here we will mainly cover the top three layers with some success stories of cloud computing. As shown in Table 4.7, cloud players are divided into three classes: (1) cloud service providers and IT administrators, (2) software developers or vendors, and (3) end users or business users. These cloud players vary in their roles under the IaaS, PaaS, and SaaS models. The table entries distinguish the three cloud models as viewed by different players. From the software vendors' perspective, application performance on a given cloud platform is most important. From the providers' perspective, cloud infrastructure performance is the primary concern. From the end users' perspective, the quality of services, including security, is the most important.

Table 4.7 Cloud Differences in Perspectives of Providers, Vendors, and Users

Cloud Players	IaaS	PaaS	SaaS
IT administrators/cloud providers	Monitor SLAs	Monitor SLAs and enable service platforms	Monitor SLAs and deploy software
Software developers (vendors)	To deploy and store data	Enabling platforms via configurators and APIs	Develop and deploy software
End users or business users	To deploy and store data	To develop and test web software	Use business software

4.5.1.1 Cloud Service Tasks and Trends

Cloud services are introduced in five layers. The top layer is for SaaS applications, as further subdivided into the five application areas in Figure 4.23, mostly for business applications. For example, CRM is heavily practiced in business promotion, direct sales, and marketing services. CRM offered the first SaaS on the cloud successfully. The approach is to widen market coverage by investigating customer behaviors and revealing opportunities by statistical analysis. SaaS tools also apply to distributed collaboration, and financial and human resources management. These cloud services have been growing rapidly in recent years.

PaaS is provided by Google, Salesforce.com, and Facebook, among others. IaaS is provided by Amazon, Windows Azure, and RackRack, among others. Collocation services require multiple cloud providers to work together to support supply chains in manufacturing. Network cloud services provide communications such as those by AT&T, Qwest, and AboveNet. Details can be found in Clou's introductory book on business clouds [18]. The vertical cloud services in Figure 4.25 refer to a sequence of cloud services that are mutually supportive. Often, cloud mashup is practiced in vertical cloud applications.

4.5.1.2 Software Stack for Cloud Computing

Despite the various types of nodes in the cloud computing cluster, the overall software stacks are built from scratch to meet rigorous goals (see Table 4.7). Developers have to consider how to design the system to meet critical requirements such as high throughput, HA, and fault tolerance. Even the operating system might be modified to meet the special requirement of cloud data processing. Based on the observations of some typical cloud computing instances, such as Google, Microsoft, and Yahoo!, the overall software stack structure of cloud computing software can be viewed as layers. Each layer has its own purpose and provides the interface for the upper layers just as the traditional software stack does. However, the lower layers are not completely transparent to the upper layers.

The platform for running cloud computing services can be either physical servers or virtual servers. By using VMs, the platform can be flexible, that is, the running services are not bound to specific hardware platforms. This brings flexibility to cloud computing platforms. The software layer on top of the platform is the layer for storing massive amounts of data. This layer acts like the file system in a traditional single machine. Other layers running on top of the file system are the layers for executing cloud computing applications. They include the

database storage system, programming for large-scale clusters, and data query language support. The next layers are the components in the software stack.

4.5.1.3 Runtime Support Services

As in a cluster environment, there are also some runtime supporting services in the cloud computing environment. Cluster monitoring is used to collect the runtime status of the entire cluster. One of the most important facilities is the cluster job management system introduced in Chapter 2. The scheduler queues the tasks submitted to the whole cluster and assigns the tasks to the processing nodes according to node availability. The distributed scheduler for the cloud application has special characteristics that can support cloud applications, such as scheduling the programs written in MapReduce style. The runtime support system keeps the cloud cluster working properly with high efficiency.

Runtime support is software needed in browser-initiated applications applied by thousands of cloud customers. The SaaS model provides the software applications as a service, rather than letting users purchase the software. As a result, on the customer side, there is no upfront investment in servers or software licensing. On the provider side, costs are rather low, compared with conventional hosting of user applications. The customer data is stored in the cloud that is either vendor proprietary or a publicly hosted cloud supporting PaaS and IaaS.

4.5.2 Resource Provisioning and Platform Deployment

The emergence of computing clouds suggests fundamental changes in software and hardware architecture. Cloud architecture puts more emphasis on the number of processor cores or VM instances. Parallelism is exploited at the cluster node level. In this section, we will discuss techniques to provision computer resources or VMs. Then we will talk about storage allocation schemes to interconnect distributed computing infrastructures by harnessing the VMs dynamically.

4.5.2.1 Provisioning of Compute Resources (VMs)

Providers supply cloud services by signing SLAs with end users. The SLAs must commit sufficient resources such as CPU, memory, and bandwidth that the user can use for a preset period. Underprovisioning of resources will lead to broken SLAs and penalties. Overprovisioning of resources will lead to resource underutilization, and consequently, a decrease in revenue for the provider. Deploying an autonomous system to efficiently provision resources to users is a challenging problem. The difficulty comes from the unpredictability of consumer demand, software and hardware failures, heterogeneity of services, power management, and conflicts in signed SLAs between consumers and service providers.

Efficient VM provisioning depends on the cloud architecture and management of cloud infrastructures. Resource provisioning schemes also demand fast discovery of services and data in cloud computing infrastructures. In a virtualized cluster of servers, this demands efficient installation of VMs, live VM migration, and fast recovery from failures. To deploy VMs, users treat them as physical hosts with customized operating systems for specific applications. For example, Amazon's EC2 uses Xen as the virtual machine monitor (VMM). The same VMM is used in IBM's Blue Cloud.

In the EC2 platform, some predefined VM templates are also provided. Users can choose different kinds of VMs from the templates. IBM's Blue Cloud does not provide any VM templates. In general, any type of VM can run on top of Xen. Microsoft also applies virtualization in its Azure cloud platform. The provider should offer resource-economic services. Power-efficient schemes for caching, query processing, and thermal management are mandatory due to increasing energy waste by heat dissipation from data centers. Public or private clouds promise to streamline the on-demand provisioning of software, hardware, and data as a service, achieving economies of scale in IT deployment and operation.

4.5.2.2 Resource Provisioning Methods

Figure 4.24 shows three cases of static cloud resource provisioning policies. In case (a), overprovisioning with the peak load causes heavy resource waste (shaded area). In case (b), underprovisioning (along the capacity line) of resources results in losses by both user and provider in that paid demand by the users (the shaded area above the capacity) is not served and wasted resources still exist for those demanded areas below the

provisioned capacity. In case (c), the constant provisioning of resources with fixed capacity to a declining user demand could result in even worse resource waste. The user may give up the service by canceling the demand, resulting in reduced revenue for the provider. Both the user and provider may be losers in resource provisioning without elasticity.

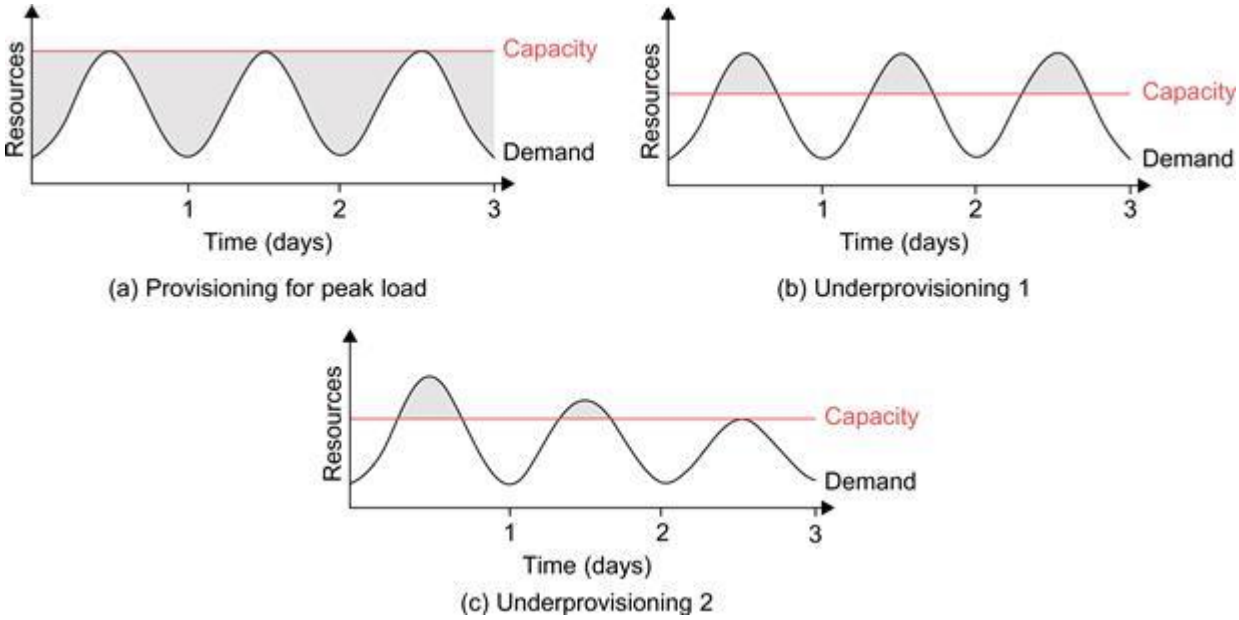


FIGURE 4.24 Three cases of cloud resource provisioning without elasticity: (a) heavy waste due to overprovisioning, (b) underprovisioning and (c) under- and then overprovisioning Courtesy of Armbrust, et al., UC Berkeley, 2009 [4]

Three resource-provisioning methods are presented in the following sections. The *demand-driven method* provides static resources and has been used in grid computing for many years. The *event-driven method* is based on predicted workload by time. The *popularity-driven method* is based on Internet traffic monitored. We characterize these resource provisioning methods as follows (see Figure 4.25).

4.5.2.3 Demand-Driven Resource Provisioning

This method adds or removes computing instances based on the current utilization level of the allocated resources. The demand-driven method automatically allocates two Xeon processors for the user application, when the user was using one Xeon processor more than 60 percent of the time for an extended period. In general, when a resource has surpassed a threshold for a certain amount of time, the scheme increases that resource based on demand. When a resource is below a threshold for a certain amount of time, that resource could be decreased accordingly. Amazon implements such an auto-scale feature in its EC2 platform. This method is easy to implement. The scheme does not work out right if the workload changes abruptly.

The x-axis in Figure 4.25 is the time scale in milliseconds. In the beginning, heavy fluctuations of CPU load are encountered. All three methods have demanded a few VM instances initially. Gradually, the utilization rate becomes more stabilized with a maximum of 20 VMs (100 percent utilization) provided for demand-driven provisioning in Figure 4.25(a). However, the event-driven method reaches a stable peak of 17 VMs toward the end of the event and drops quickly in Figure 4.25(b). The popularity provisioning shown in Figure 4.25(c) leads to a similar fluctuation with peak VM utilization in the middle of the plot..

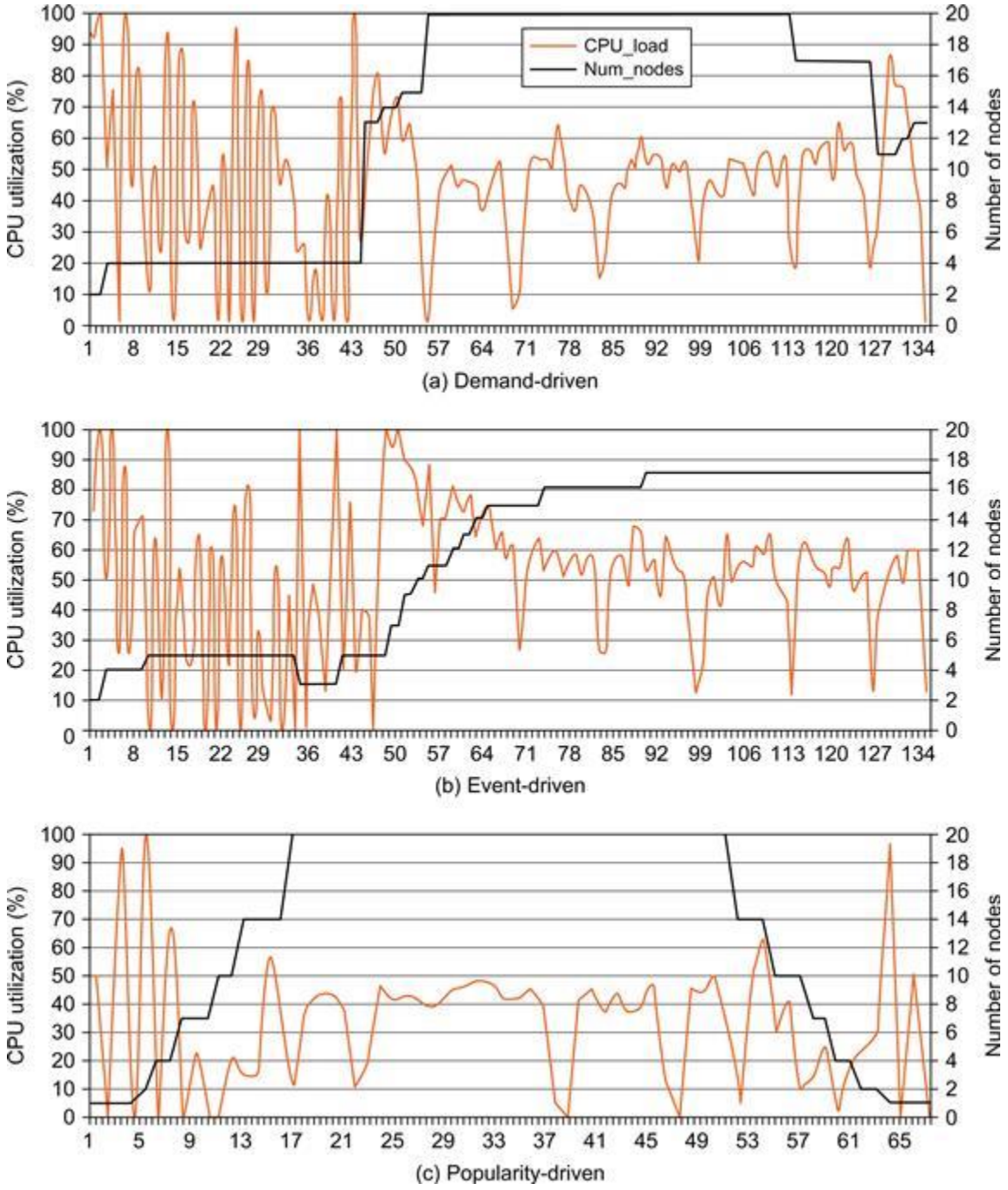


FIGURE 4.25 EC2 performance results on the AWS EC2 platform, collected from experiments at the University of Southern California using three resource provisioning methods Courtesy of Ken Wu, USC

4.5.2.4 Event-Driven Resource Provisioning

This scheme adds or removes machine instances based on a specific time event. The scheme works better for seasonal or predicted events such as Christmastime in the West and the Lunar New Year in the East. During these events, the number of users grows before the event period and then decreases during the event period. This scheme anticipates peak traffic before it happens. The method results in a minimal loss of QoS, if the event is predicted correctly. Otherwise, wasted resources are even greater due to events that do not follow a fixed pattern.

4.5.2.5 Popularity-Driven Resource Provisioning

In this method, the Internet searches for popularity of certain applications and creates the instances by popularity demand. The scheme anticipates increased traffic with popularity. Again, the scheme has a minimal loss of QoS, if the predicted popularity is correct. Resources may be wasted if traffic does not occur as expected. In Figure 4.25(c), EC2 performance by CPU utilization rate (the dark curve with the percentage scale shown on

the left) is plotted against the number of VMs provisioned (the light curves with scale shown on the right, with a maximum of 20 VMs provisioned).

4.5.2.6 Dynamic Resource Deployment

The cloud uses VMs as building blocks to create an execution environment across multiple resource sites. The InterGrid-managed infrastructure was developed by a Melbourne University group [19]. Dynamic resource deployment can be implemented to achieve scalability in performance. The InterGrid is a Java-implemented software system that lets users create execution cloud environments on top of all participating grid resources. Peering arrangements established between gateways enable the allocation of resources from multiple grids to establish the execution environment. In Figure 4.26, a scenario is illustrated by which an *intergrid gateway (IGG)* allocates resources from a local cluster to deploy applications in three steps: (1) requesting the VMs, (2) enacting the leases, and (3) deploying the VMs as requested. Under peak demand, this IGG interacts with another IGG that can allocate resources from a cloud computing provider.

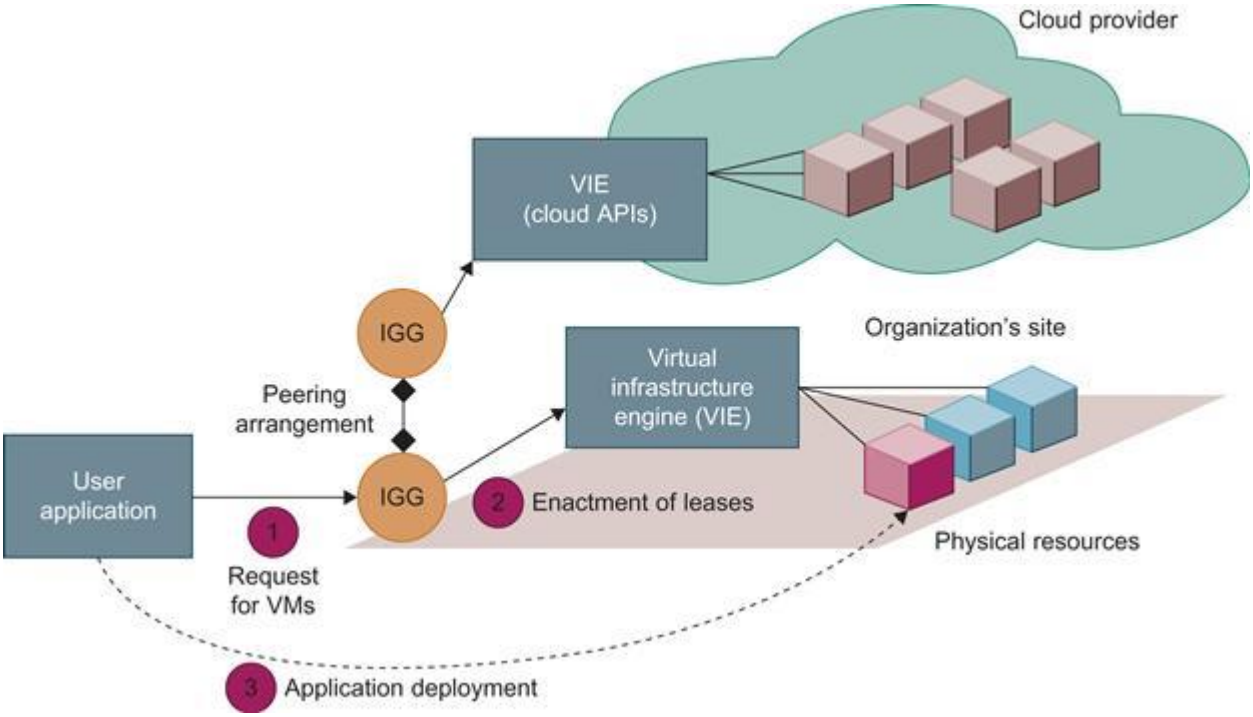


FIGURE 4.26 Cloud resource deployment using an IGG (intergrid gateway) to allocate the VMs from a Local cluster to interact with the IGG of a public cloud provider. Courtesy of Constanzo, et al. © IEEE [21]

A grid has predefined peering arrangements with other grids, which the IGG manages. Through multiple IGGs, the system coordinates the use of InterGrid resources. An IGG is aware of the peering terms with other grids, selects suitable grids that can provide the required resources, and replies to requests from other IGGs. Request redirection policies determine which peering grid InterGrid selects to process a request and a price for which that grid will perform the task. An IGG can also allocate resources from a cloud provider. The cloud system creates a virtual environment to help users deploy their applications. These applications use the distributed grid resources.

The InterGrid allocates and provides a *distributed virtual environment* (DVE). This is a virtual cluster of VMs that runs isolated from other virtual clusters. A component called the *DVE manager* performs resource allocation and management on behalf of specific user applications. The core component of the IGG is a scheduler for implementing provisioning policies and peering with other gateways. The communication component provides an asynchronous message-passing mechanism. Received messages are handled in parallel by a thread pool.

4.5.2.7 Provisioning of Storage Resources

The data storage layer is built on top of the physical or virtual servers. As the cloud computing applications often provide service to users, it is unavoidable that the data is stored in the clusters of the cloud provider. The service can be accessed anywhere in the world. One example is e-mail systems. A typical large e-mail system

might have millions of users and each user can have thousands of e-mails and consume multiple gigabytes of disk space. Another example is a web searching application. In storage technologies, hard disk drives may be augmented with solid-state drives in the future. This will provide reliable and high-performance data storage. The biggest barriers to adopting flash memory in data centers have been price, capacity, and, to some extent, a lack of sophisticated query-processing techniques. However, this is about to change as the I/O bandwidth of solid-state drives becomes too impressive to ignore.

A distributed file system is very important for storing large-scale data. However, other forms of data storage also exist. Some data does not need the namespace of a tree structure file system, and instead, databases are built with stored data files. In cloud computing, another form of data storage is (Key, Value) pairs. Amazon S3 service uses SOAP to access the objects stored in the cloud. Table 4.8 outlines three cloud storage services provided by Google, Hadoop, and Amazon.

Table 4.8 Storage Services in Three Cloud Computing Systems

Storage System	Features
GFS: Google File System	Very large sustainable reading and writing bandwidth, mostly continuous accessing instead of random accessing. The programming interface is similar to that of the POSIX file system accessing interface.
HDFS: Hadoop Distributed File System	The open source clone of GFS. Written in Java. The programming interfaces are similar to POSIX but not identical.
Amazon S3 and EBS	S3 is used for retrieving and storing data from/to remote servers. EBS is built on top of S3 for using virtual disks in running EC2 instances.

Many cloud computing companies have developed large-scale data storage systems to keep huge amount of data collected every day. For example, Google's GFS stores web data and some other data, such as geographic data for Google Earth. A similar system from the open source community is the Hadoop Distributed File System (HDFS) for Apache. Hadoop is the open source implementation of Google's cloud computing infrastructure. Similar systems include Microsoft's Cosmos file system for the cloud.

Despite the fact that the storage service or distributed file system can be accessed directly, similar to traditional databases, cloud computing does provide some forms of structure or semistructure database processing capability. For example, applications might want to process the information contained in a web page. Web pages are an example of semistructural data in HTML format. If some forms of database capability can be used, application developers will construct their application logic more easily. Another reason to build a database-like service in cloud computing is that it will be quite convenient for traditional application developers to code for the cloud platform. Databases are quite common as the underlying storage device for many applications.

Thus, such developers can think in the same way they do for traditional software development. Hence, in cloud computing, it is necessary to build databases like large-scale systems based on data storage or distributed file systems. The scale of such a database might be quite large for processing huge amounts of data. The main purpose is to store the data in structural or semi-structural ways so that application developers can use it easily and build their applications rapidly. Traditional databases will meet the performance bottleneck while the system is expanded to a larger scale. However, some real applications do not need such strong consistency. The scale of such databases can be quite large. Typical cloud databases include BigTable from Google, SimpleDB from Amazon, and the SQL service from Microsoft Azure.

4.5.3 Virtual Machine Creation and Management

In this section, we will consider several issues for cloud infrastructure management. First, we will consider the resource management of independent service jobs. Then we will consider how to execute third-party cloud applications. Cloud-loading experiments are used by a Melbourne research group on the French Grid'5000 system. This experimental setting illustrates VM creation and management. This case study example reveals major VM management issues and suggests some plausible solutions for workload-balanced execution. Figure

4.27 shows the interactions among VM managers for cloud creation and management. The managers provide a public API for users to submit and control the VMs.

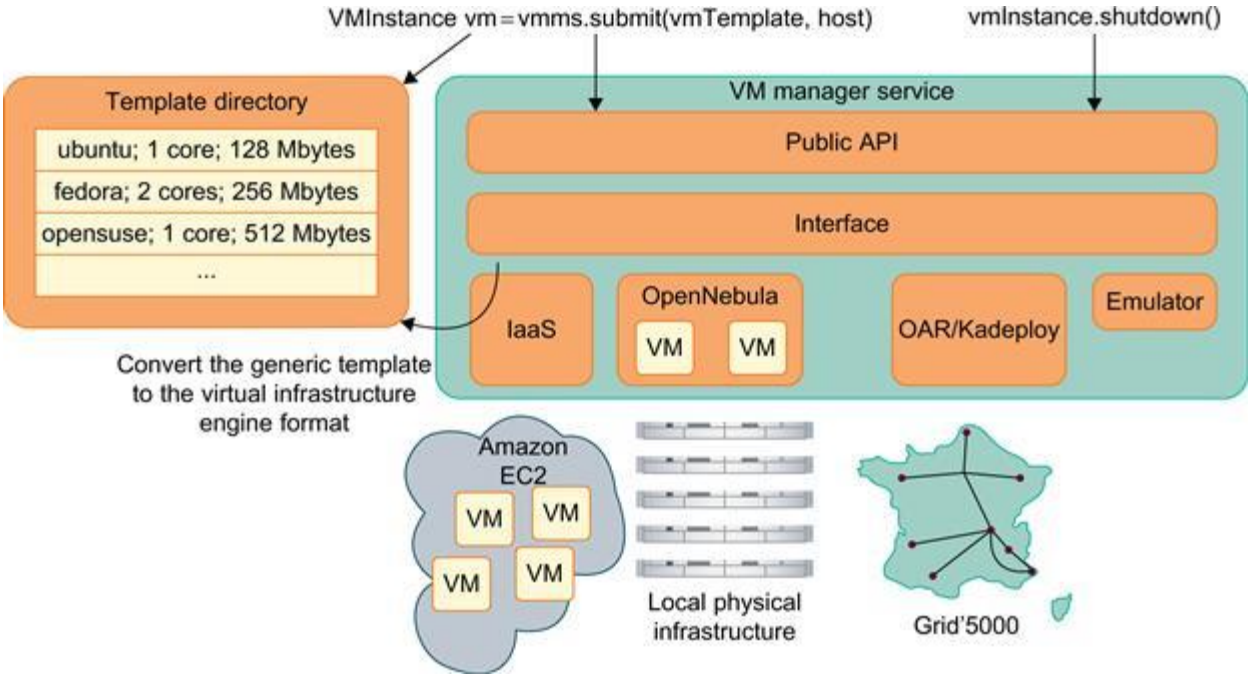


FIGURE 4.27 Interactions among VM managers for cloud creation and management; the manager provides a public API for users to submit and control the VMs Courtesy of Constanzo, et al. © IEEE [21]

4.5.3.1 Independent Service Management

Independent services request facilities to execute many unrelated tasks. Commonly, the APIs provided are some web services that the developer can use conveniently. In Amazon cloud computing infrastructure, SQS is constructed for providing a reliable communication service between different providers. Even the endpoint does not run while another entity has posted a message in SQS. By using independent service providers, the cloud applications can run different services at the same time. Some other services are used for providing data other than the compute or storage services.

4.5.3.2 Running Third-Party Applications

Cloud platforms have to provide support for building applications that are constructed by third-party application providers or programmers. As current web applications are often provided by using Web 2.0 forms (interactive applications with Ajax), the programming interfaces are different from the traditional programming interfaces such as functions in runtime libraries. The APIs are often in the form of services. Web service application engines are often used by programmers for building applications. The web browsers are the user interface for end users.

In addition to gateway applications, the cloud computing platform provides the extra capabilities of accessing backend services or underlying data. As examples, GAE and Microsoft Azure apply their own cloud APIs to get special cloud services. The WebSphere application engine is deployed by IBM for Blue Cloud. It can be used to develop any kind of web application written in Java. In EC2, users can use any kind of application engine that can run in VM instances.

4.5.3.3 Virtual Machine Manager

The VM manager is the link between the gateway and resources. The gateway doesn't share physical resources directly, but relies on virtualization technology for abstracting them. Hence, the actual resources it uses are VMs. The manager manage VMs deployed on a set of physical resources. The VM manager implementation is generic so that it can connect with different VIEs. Typically, VIEs can create and stop VMs on a physical cluster. The Melbourne group has developed managers for OpenNebula, Amazon EC2, and French Grid'5000. The manager using the OpenNebula OS (www.opennebula.org) to deploy VMs on local clusters.

OpenNebula runs as a daemon service on a master node, so the VMM works as a remote user. Users submit VMs on physical machines using different kinds of hypervisors, such as Xen (www.xen.org), which enables the running of several operating systems on the same host concurrently. The VMM also manages VM deployment on grids and IaaS providers. The InterGrid supports Amazon EC2. The connector is a wrapper for the command-line tool Amazon provides. The VM manager for Grid'5000 is also a wrapper for its command-line tools. To deploy a VM, the manager needs to use its template.

4.5.3.4 Virtual Machine Templates

A *VM template* is analogous to a computer's configuration and contains a description for a VM with the following static information:

- The number of cores or processors to be assigned to the VM
- The amount of memory the VM requires
- The kernel used to boot the VM's operating system
- The disk image containing the VM's file system
- The price per hour of using a VM

The gateway administrator provides the VM template information when the infrastructure is set up. The administrator can update, add, and delete templates at any time. In addition, each gateway in the InterGrid network must agree on the templates to provide the same configuration on each site. To deploy an instance of a given VM, the VMM generates a descriptor from the template. This descriptor contains the same fields as the template and additional information related to a specific VM instance. Typically the additional information includes:

- The disk image that contains the VM's file system
- The address of the physical machine hosting the VM
- The VM's network configuration
- The required information for deployment on an IaaS provider

Before starting an instance, the scheduler gives the network configuration and the host's address; it then allocates MAC and IP addresses for that instance. The template specifies the disk image field. To deploy several instances of the same VM template in parallel, each instance uses a temporary copy of the disk image. Hence, the descriptor contains the path to the copied disk image. The descriptor's fields are different for deploying a VM on an IaaS provider. Network information is not needed, because Amazon EC2 automatically assigns a public IP to the instances. The IGG works with a repository of VM templates, called the *VM template directory*.

4.5.3.5 Distributed VM Management

Figure 4.30 illustrates the interactions between InterGrid's components. A distributed VM manager makes requests for VMs and queries their status. This manager requests VMs from the gateway on behalf of the user application. The manager obtains the list of requested VMs from the gateway. This list contains a tuple of public IP/private IP addresses for each VM with Secure Shell (SSH) tunnels. Users must specify which VM template they want to use and the number of VM instances needed, the deadline, the wall time, and the address for an alternative gateway.

The local gateway tries to obtain resources from the underlying VIEs. When this is impossible, the local gateway starts a negotiation with any remote gateways to fulfill the request. When a gateway schedules the VMs, it sends the VM access information to the requester gateway. Finally, the manager configures the VM, sets up SSH tunnels, and executes the tasks on the VM. Under the peering policy, each gateway's scheduler uses conservative backfilling to schedule requests. When the scheduler can't start a request immediately using local resources, a redirection algorithm will be initiated.

Example 4.6 Experiments on an InterGrid Test Bed over the Grid'5000

The Melbourne group conducted two experiments to evaluate the InterGrid architecture. The first one evaluates the performance of allocation decisions by measuring how the IGG manages load via peering arrangements. The second considers its effectiveness in deploying a bag-of-tasks application. The experiment was conducted

on the French experimental grid platform Grid'5000. Grid'5000 comprises 4,792 processor cores on nine grid sites across France. Each gateway represents one Grid'5000 site, as shown in Figure 4.28.

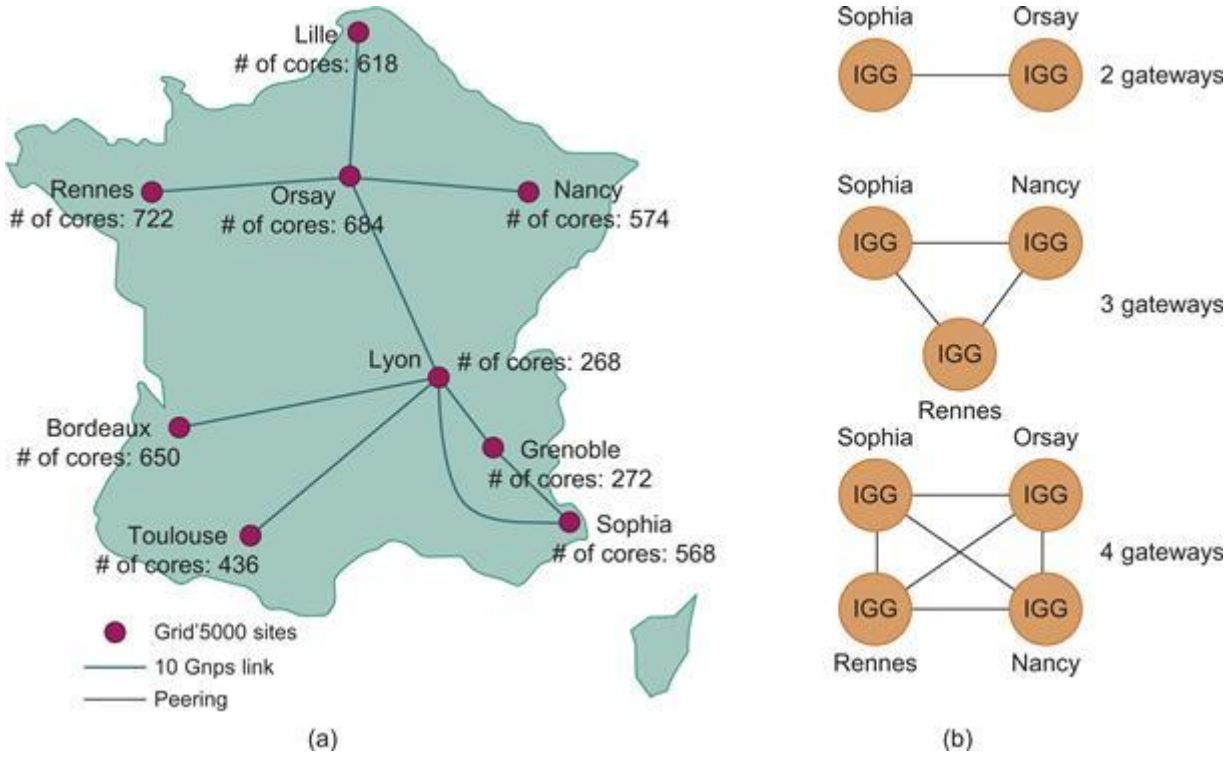


FIGURE 4.28 The InterGrid test bed over the French Grid'5000 located in nine cities across France Courtesy of Constanzo, et al. © IEEE [21]

To prevent the gateways from interfering with real Grid'5000 users, emulated VM managers were implemented to instantiate fictitious VMs. The number of emulated hosts is limited by the core number at each site. A balanced workload was configured among the sites. The maximum number of VMs requested does not exceed the number of cores in any site. The load characteristics are shown in Figure 4.29 under a four-gateway scenario. The teal bars indicate each grid site's load. The magenta bars show the load when gateways redirect requests to one another. The green bars correspond to the amount of load each gateway accepts from other gateways. The brown bars represent the amount of load that is redirected. The results show that the loading policy can balance the load across the nine sites. Rennes, a site with a heavy load, benefits from peering with other gateways as the gateway redirects a great share of its load to other sites.

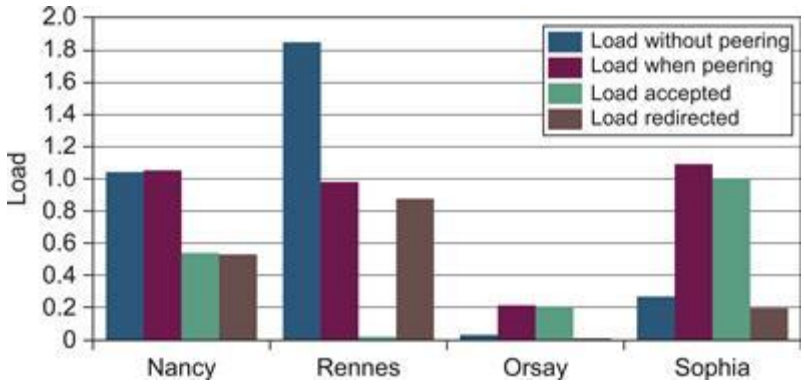


FIGURE 4.29 Cloud loading results at four gateways at resource sites in the Grid'5000 system Courtesy of Constanzo, et al. © IEEE [21]

4.5.4 Global Exchange of Cloud Resources

In order to support a large number of application service consumers from around the world, cloud infrastructure providers (i.e., IaaS providers) have established data centers in multiple geographical locations to provide redundancy and ensure reliability in case of site failures. For example, Amazon has data centers in the

United States (e.g., one on the East Coast and another on the West Coast) and Europe. However, currently Amazon expects its cloud customers (i.e., SaaS providers) to express a preference regarding where they want their application services to be hosted. Amazon does not provide seamless/automatic mechanisms for scaling its hosted services across multiple geographically distributed data centers.

This approach has many shortcomings. First, it is difficult for cloud customers to determine in advance the best location for hosting their services as they may not know the origin of consumers of their services. Second, SaaS providers may not be able to meet the QoS expectations of their service consumers originating from multiple geographical locations. This necessitates building mechanisms for seamless federation of data centers of a cloud provider or providers supporting dynamic scaling of applications across multiple domains in order to meet QoS targets of cloud customers. Figure 4.30shows the high-level components of the Melbourne group's proposed InterCloud architecture.

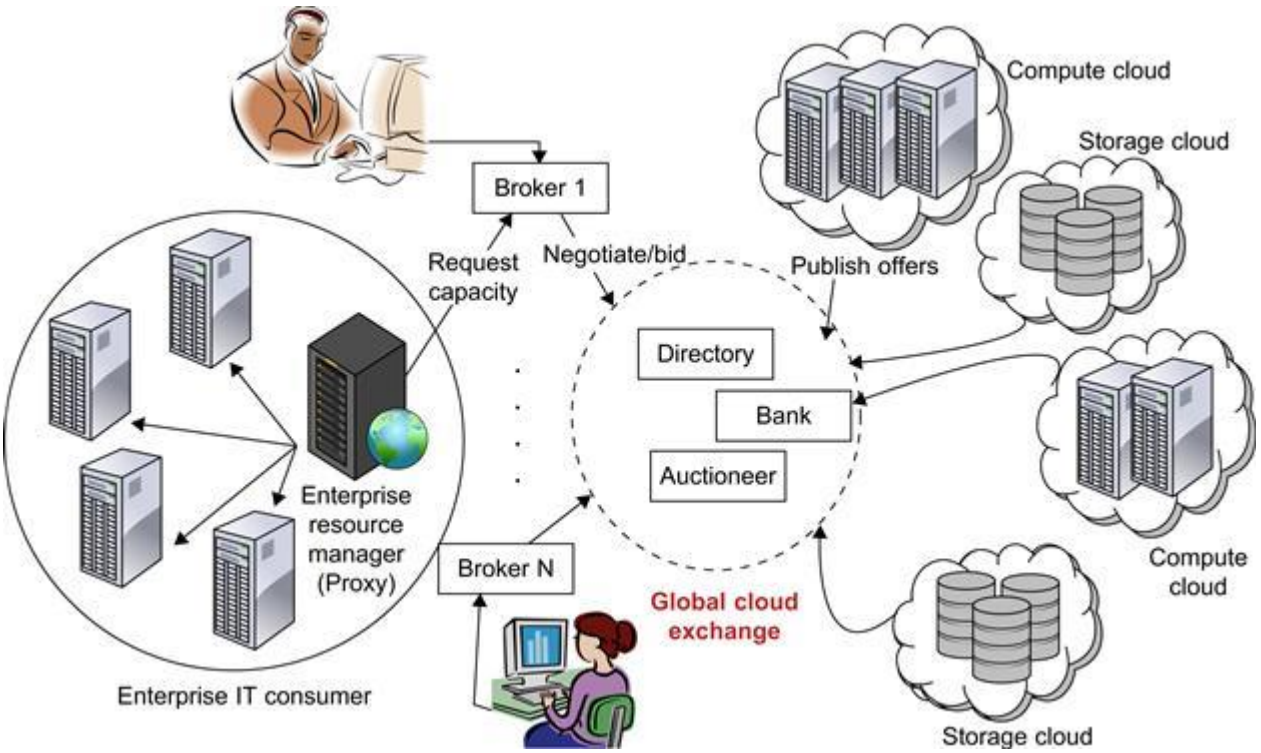


FIGURE 4.30 Inter-cloud exchange of cloud resources through brokeringCourtesy of R. Buyya, et al., University of Melbourne [12]

In addition, no single cloud infrastructure provider will be able to establish its data centers at all possible locations throughout the world. As a result, cloud application service (SaaS) providers will have difficulty in meeting QoS expectations for all their consumers. Hence, they would like to make use of services of multiple cloud infrastructure service providers who can provide better support for their specific consumer needs. This kind of requirement often arises in enterprises with global operations and applications such as Internet services, media hosting, and Web 2.0 applications. This necessitates federation of cloud infrastructure service providers for seamless provisioning of services across different cloud providers. To realize this, the Cloudbus Project at the University of Melbourne has proposed InterCloud architecture [12] supporting brokering and exchange of cloud resources for scaling applications across multiple clouds.

By realizing InterCloud architectural principles in mechanisms in their offering, cloud providers will be able to dynamically expand or resize their provisioning capability based on sudden spikes in workload demands by leasing available computational and storage capabilities from other cloud service providers; operate as part of a market-driven resource leasing federation, where application service providers such as Salesforce.com host their services based on negotiated SLA contracts driven by competitive market prices; and deliver on-demand, reliable, cost-effective, and QoS-aware services based on virtualization technologies while ensuring high QoS standards and minimizing service costs. They need to be able to utilize market-based utility models as the basis for provisioning of virtualized software services and federated hardware infrastructure among users with heterogeneous applications.

They consist of client brokering and coordinator services that support utility-driven federation of clouds: application scheduling, resource allocation, and migration of workloads. The architecture cohesively couples the administratively and topologically distributed storage and compute capabilities of clouds as part of a single resource leasing abstraction. The system will ease the cross-domain capability integration for on-demand, flexible, energy-efficient, and reliable access to the infrastructure based on virtualization technology [6,75].

The *Cloud Exchange (CEx)* acts as a market maker for bringing together service producers and consumers. It aggregates the infrastructure demands from application brokers and evaluates them against the available supply currently published by the cloud coordinators. It supports trading of cloud services based on competitive economic models such as commodity markets and auctions. CEx allows participants to locate providers and consumers with fitting offers. Such markets enable services to be commoditized, and thus will pave the way for creation of dynamic market infrastructure for trading based on SLAs. An SLA specifies the details of the service to be provided in terms of metrics agreed upon by all parties, and incentives and penalties for meeting and violating the expectations, respectively. The availability of a banking system within the market ensures that financial transactions pertaining to SLAs between participants are carried out in a secure and dependable environment.

4.6 Cloud Security and Trust Management

Lacking trust between service providers and cloud users has hindered the universal acceptance of cloud computing as a service on demand. In the past, trust models have been developed to protect mainly e-commerce and online shopping provided by eBay and Amazon. For web and cloud services, trust and security become even more demanding, because leaving user applications completely to the cloud providers has faced strong resistance by most PC and server users. Cloud platforms become worrisome to some users for lack of privacy protection, security assurance, and copyright protection. Trust is a social problem, not a pure technical issue. However, the social problem can be solved with a technical approach.

Common sense dictates that technology can enhance trust, justice, reputation, credit, and assurance in Internet applications. As a virtual environment, the cloud poses new security threats that are more difficult to contain than traditional client and server configurations. To solve these trust problems, a new data-protection model is presented in this section. In many cases, one can extend the trust models for P2P networks and grid systems to protect clouds and data centers.

4.6.1 Cloud Security Defense Strategies

A healthy cloud ecosystem is desired to free users from abuses, violence, cheating, hacking, viruses, rumors, pornography, spam, and privacy and copyright violations. The security demands of three cloud service models, IaaS, PaaS, and SaaS, are described in this section. These security models are based on various SLAs between providers and users.

4.6.1.1 Basic Cloud Security

Three basic cloud security enforcements are expected. First, facility security in data centers demands on-site security year round. Biometric readers, CCTV (close-circuit TV), motion detection, and man traps are often deployed. Also, network security demands fault-tolerant external firewalls, intrusion detection systems (IDSes), and third-party vulnerability assessment. Finally, platform security demands SSL and data decryption, strict password policies, and system trust certification. Figure 4.31shows the mapping of cloud models, where special security measures are deployed at various cloud operating levels.

Servers in the cloud can be physical machines or VMs. User interfaces are applied to request services. The provisioning tool carves out the systems from the cloud to satisfy the requested service. A security-aware cloud architecture demands security enforcement. Malware-based attacks such as network worms, viruses, and DDoS attacks exploit system vulnerabilities. These attacks compromise system functionality or provide intruders unauthorized access to critical information. Thus, security defenses are needed to protect all cluster servers and data centers. Here are some cloud components that demand special security protection:

- Protection of servers from malicious software attacks such as worms, viruses, and malware
- Protection of hypervisors or VM monitors from software-based attacks and vulnerabilities

• Protection of VMs and monitors from service disruption and DoS attacks
• Protection of data and information from theft, corruption, and natural disasters
• Providing authenticated and authorized access to critical data and services

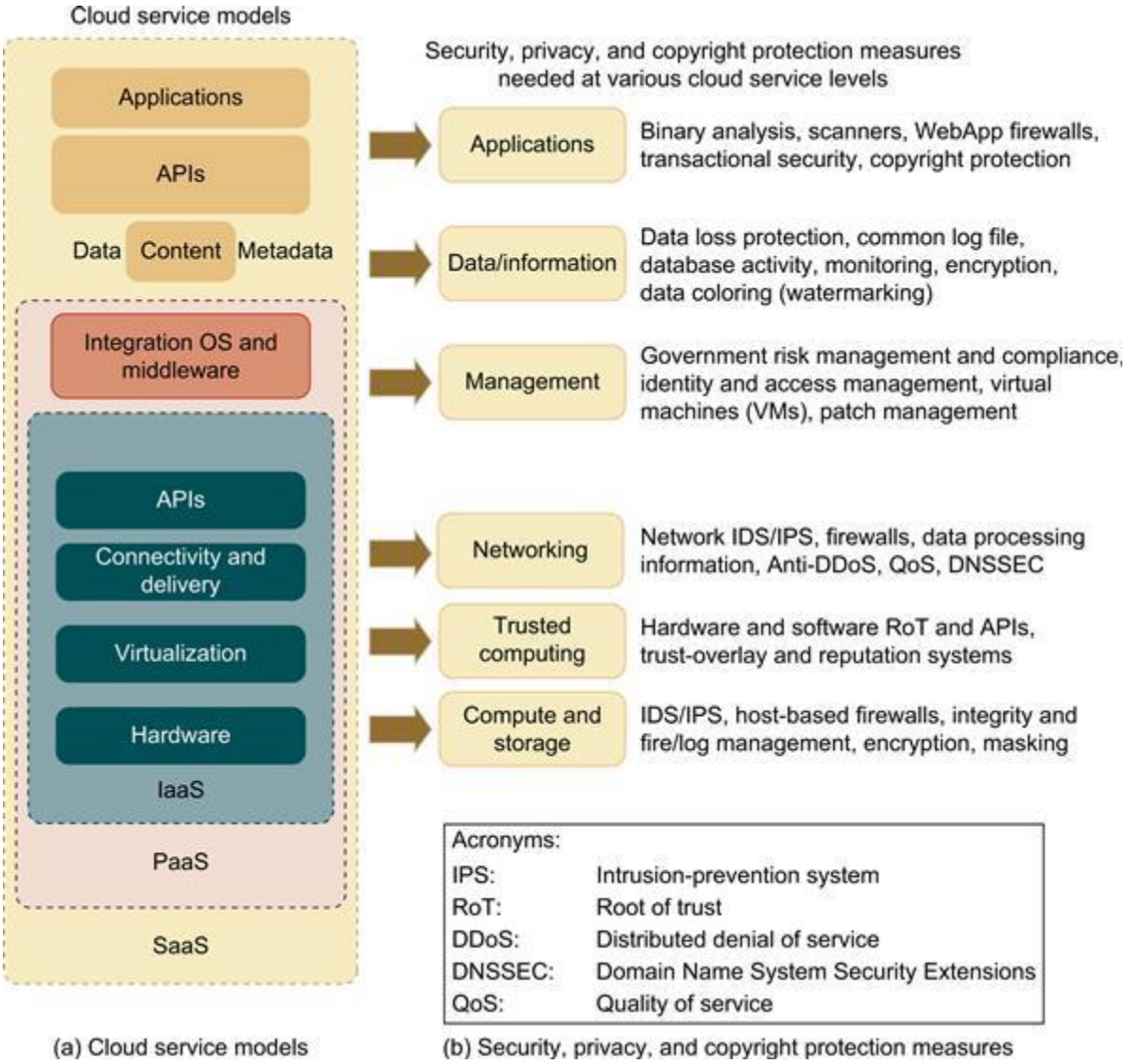


FIGURE 4.31 Cloud service models on the left (a) and corresponding security measures on the right (b); the IaaS is at the innermost level, PaaS is at the middle level, and SaaS is at the outermost level, including all hardware, software, datasets, and networking resources Courtesy of Hwang and Li [36]

4.6.1.2 Security Challenges in VMs

As we discussed earlier in this chapter, traditional network attacks include buffer overflows, DoS attacks, spyware, malware, rootkits, Trojan horses, and worms. In a cloud environment, newer attacks may result from hypervisor malware, guest hopping and hijacking, or VM rootkits. Another type of attack is the man-in-the-middle attack for VM migrations. In general, passive attacks steal sensitive data or passwords. Active attacks may manipulate kernel data structures which will cause major damage to cloud servers. An IDS can be a NIDS or a HIDS. Program shepherding can be applied to control and verify code execution. Other defense technologies include using the RIO dynamic optimization infrastructure, or VMware's vSafe and vShield tools, security compliance for hypervisors, and Intel vPro technology. Others apply a hardened OS environment or use isolated execution and sandboxing.

4.6.1.3 Cloud Defense Methods

Virtualization enhances cloud security. But VMs add an additional layer of software that could become a single point of failure. With virtualization, a single physical machine can be divided or partitioned into multiple VMs (e.g., server consolidation). This provides each VM with better security isolation and each partition is protected

from DoS attacks by other partitions. Security attacks in one VM are isolated and contained from affecting the other VMs. Table 4.9 lists eight protection schemes to secure public clouds and data centers. VM failures do not propagate to other VMs. The hypervisor provides visibility of the guest OS, with complete guest isolation. Fault containment and failure isolation of VMs provide a more secure and robust environment. Malicious intrusions may destroy valuable hosts, networks, and storage resources. Internet anomalies found in routers, gateways, and distributed hosts may stop cloud services. Trust negotiation is often done at the SLA level. Public Key Infrastructure (PKI) services could be augmented with data-center reputation systems. Worm and DDoS attacks must be contained. It is harder to establish security in the cloud because all data and software are shared by default.

Table 4.9 Physical and Cyber Security Protection at Cloud/Data Centers

Protection Schemes	Brief Description and Deployment Suggestions
Secure data centers and computer buildings	Choose hazard-free location, enforce building safety. Avoid windows, keep buffer zone around the site, bomb detection, camera surveillance, earthquake-proof, etc.
Use redundant utilities at multiple sites	Multiple power and supplies, alternate network connections, multiple databases at separate sites, data consistency, data watermarking, user authentication, etc.
Trust delegation and negotiation	Cross certificates to delegate trust across PKI domains for various data centers, trust negotiation among certificate authorities (CAs) to resolve policy conflicts
Worm containment and DDoS defense	Internet worm containment and distributed defense against DDoS attacks to secure all data centers and cloud platforms
Reputation system for data centers	Reputation system could be built with P2P technology; one can build a hierarchy of reputation systems from data centers to distributed file systems
Fine-grained file access control	Fine-grained access control at the file or object level; this adds to security protection beyond firewalls and IDSes
Copyright protection and piracy prevention	Piracy prevention achieved with peer collusion prevention, filtering of poisoned content, nondestructive read, alteration detection, etc.
Privacy protection	Uses double authentication, biometric identification, intrusion detection and disaster recovery, privacy enforcement by data watermarking, data classification, etc.

4.6.1.4 Defense with Virtualization

The VM is decoupled from the physical hardware. The entire VM can be represented as a software component and can be regarded as binary or digital data. The VM can be saved, cloned, encrypted, moved, or restored with ease. VMs enable HA and faster disaster recovery. Live migration of VMs was suggested by many researchers [36] for building *distributed intrusion detection systems (DIDSes)*. Multiple IDS VMs can be deployed at various resource sites including data centers. DIDS design demands trust negation among PKI domains. Security policy conflicts must be resolved at design time and updated periodically.

4.6.1.5 Privacy and Copyright Protection

The user gets a predictable configuration before actual system integration. Yahoo!'s Pipes is a good example of a lightweight cloud platform. With shared files and data sets, privacy, security, and copyright data could be compromised in a cloud computing environment. Users desire to work in a software environment that provides many useful tools to build cloud applications over large data sets. Google's platform essentially applies in-house software to protect resources. The Amazon EC2 applies HMEC and X.509 certificates in securing resources. It is necessary to protect browser-initiated application software in the cloud environment. Here are several security features desired in a secure cloud:

- Dynamic web services with full support from secure web technologies
- Established trust between users and providers through SLAs and reputation systems
- Effective user identity management and data-access management
- Single sign-on and single sign-off to reduce security enforcement overhead
- Auditing and copyright compliance through proactive enforcement

• Shifting of control of data operations from the client environment to cloud providers
• Protection of sensitive and regulated information in a shared environment

Example 4.7 Cloud Security Safeguarded by Gateway and Firewalls

Figure 4.32 shows a security defense system for a typical private cloud environment. The gateway is fully secured to protect access to commercial clouds that are wide open to the general public. The firewall provides an external shield. The gateway secures the application server, message queue, database, web service client, and browser with HTTP, JMS, SQL, XML, and SSL security protocols, etc. The defense scheme is needed to protect user data from server attacks. A user's private data must not be leaked to other users without permission.

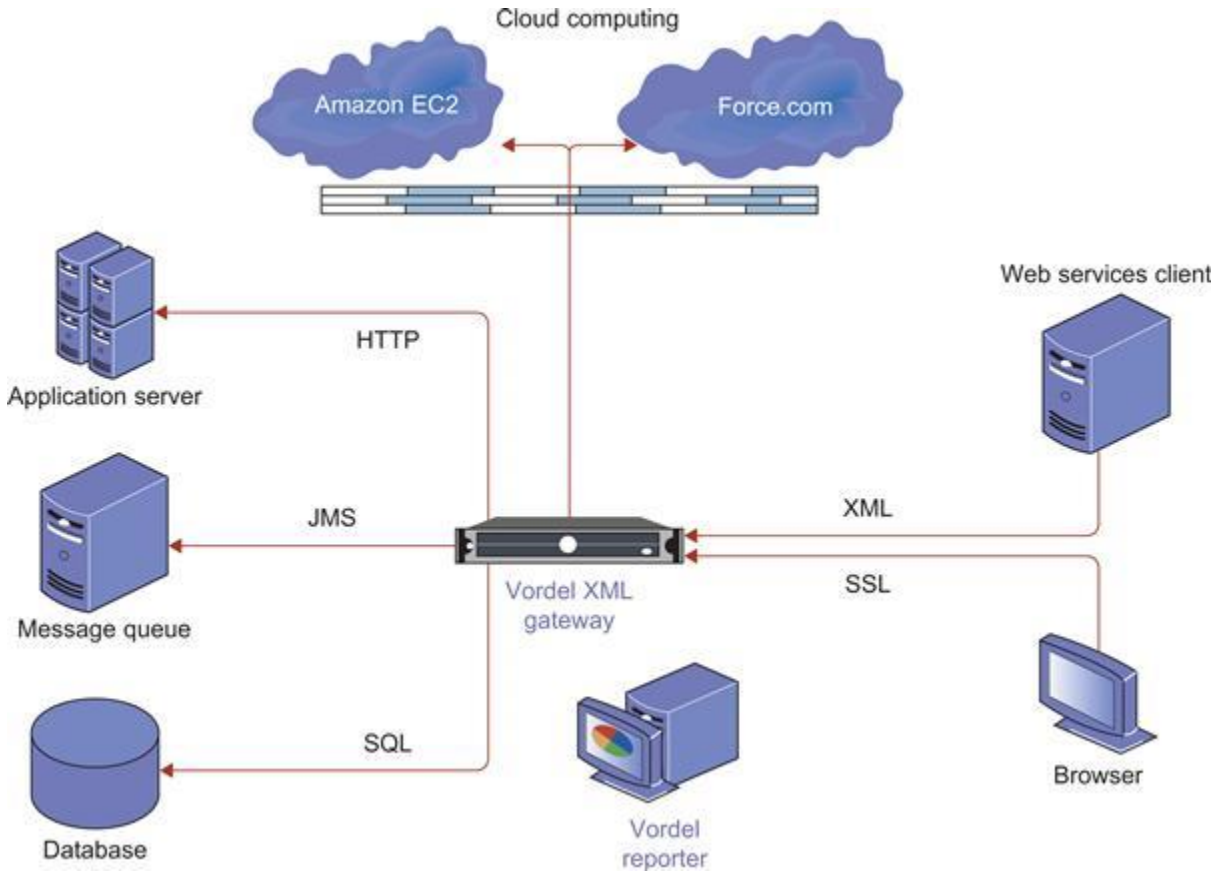


FIGURE 4.32 The typical security structure coordinated by a secured gateway plus external firewalls to safeguard the access of public or private clouds Courtesy of Vordel Company

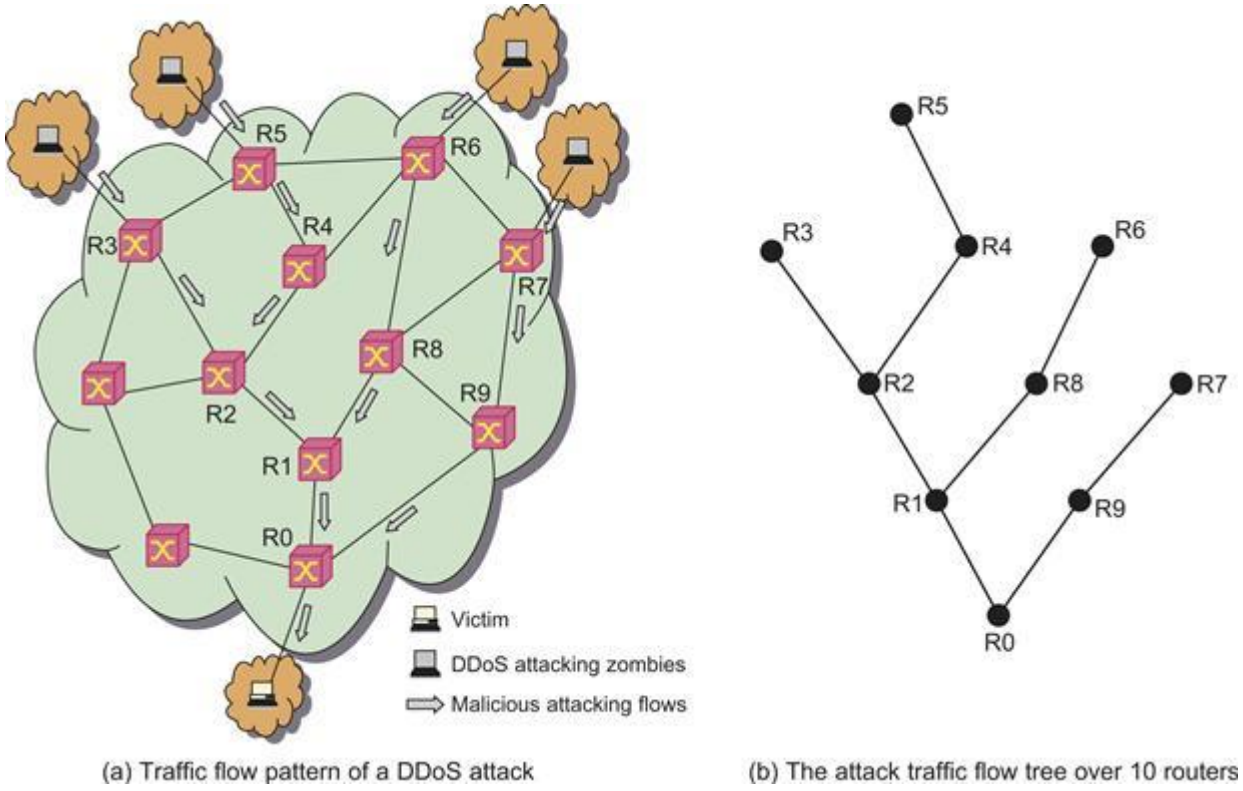
4.6.2 Distributed Intrusion/Anomaly Detection

Data security is the weakest link in all cloud models. We need new cloud security standards to apply common API tools to cope with the data lock-in problem and network attacks or abuses. The IaaS model represented by Amazon is most sensitive to external attacks. Role-based interface tools alleviate the complexity of the provisioning system. For example, IBM's Blue Cloud provisions through a role-based web portal. A SaaS bureau may order secretarial services from a common cloud platform. Many IT companies are now offering cloud services with no guaranteed security.

Security threats may be aimed at VMs, guest OSes, and software running on top of the cloud. IDSes attempt to stop these attacks before they take effect. Both signature matching and anomaly detection can be implemented on VMs dedicated to building IDSes. Signature-matching IDS technology is more mature, but require frequent updates of the signature databases. Network anomaly detection reveals abnormal traffic patterns, such as unauthorized episodes of TCP connection sequences, against normal traffic patterns. Distributed IDSes are needed to combat both types of intrusions.

4.6.2.1 Distributed Defense against DDoS Flooding Attacks

A DDoS defense system must be designed to cover multiple network domains spanned by a given cloud platform. These network domains cover the edge networks where cloud resources are connected. DDoS attacks come with widespread worms. The flooding traffic is large enough to crash the victim server by buffer overflow, disk exhaustion, or connection saturation. Figure 4.33(a) shows a flooding attack pattern. Here, the hidden attacker launched the attack from many zombies toward a victim server at the bottom router R_0.



(a) Traffic flow pattern of a DDoS attack (b) The attack traffic flow tree over 10 routers

FIGURE 4.33 DDoS attacks and defense by change-point detection at all routers on the flooding tree Courtesy of Chen, Hwang, and Ku [15]

The flooding traffic flows essentially with a tree pattern shown in Figure 4.33(b). Successive attack-transit routers along the tree reveal the abnormal surge in traffic. This DDoS defense system is based on change-point detection by all routers. Based on the anomaly pattern detected in covered network domains, the scheme detects a DDoS attack before the victim is overwhelmed. The detection scheme is suitable for protecting cloud core networks. The provider-level cooperation eliminates the need for intervention by edge networks.

Example 4.8 Man-in-the-Middle Attacks

Figure 4.34 shows VM migration from host machine VMM A to host machine VMM B, via a security vulnerable network. In a man-in-the-middle attack, the attacker can view the VM contents being migrated, steal sensitive data, or even modify the VM-specific contents including the OS and application states. An attacker posing this attack can launch an active attack to insert a VM-based rootkit into the migrating VM, which can subvert the entire operation of the migration process without the knowledge of the guest OS and embedded application.

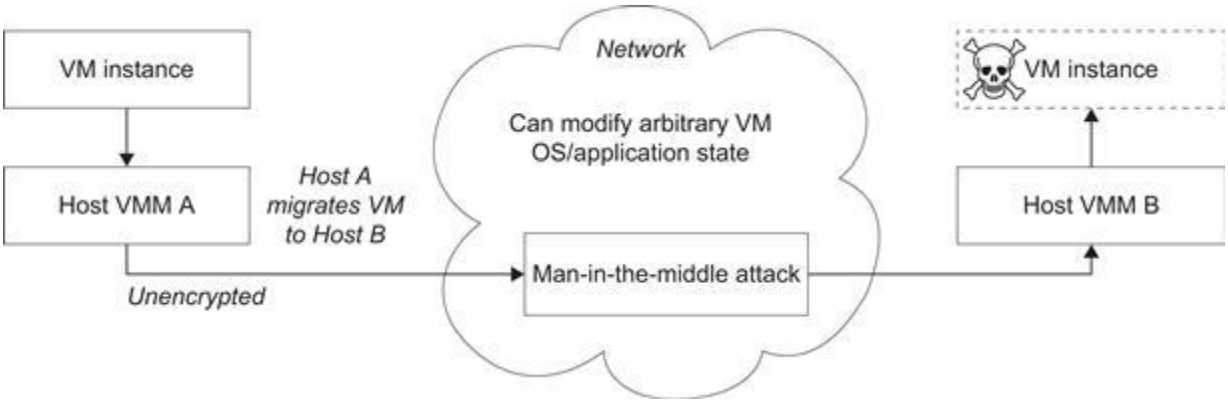


FIGURE 4.34 A VM migrating from host A to host B through a vulnerable network threatened by a man-in-the-middle attack to modify the VM template and OS state.

4.6.3 Data and Software Protection Techniques

In this section, we will introduce a data coloring technique to preserve data integrity and user privacy. Then we will discuss a watermarking scheme to protect software files from being widely distributed in a cloud environment.

4.6.3.1 Data Integrity and Privacy Protection

Users desire a software environment that provides many useful tools to build cloud applications over large data sets. In addition to application software for MapReduce, BigTable, EC2, 3S, Hadoop, AWS, GAE, and WebSphere2, users need some security and privacy protection software for using the cloud. Such software should offer the following features:

- Special APIs for authenticating users and sending e-mail using commercial accounts
- Fine-grained access control to protect data integrity and deter intruders or hackers
- Shared data sets protected from malicious alteration, deletion, or copyright violation
- Ability to secure the ISP or cloud service provider from invading users' privacy
- Personal firewalls at user ends to keep shared data sets from Java, JavaScript, and ActiveX applets
- A privacy policy consistent with the cloud service provider's policy, to protect against identity theft, spyware, and web bugs
- VPN channels between resource sites to secure transmission of critical data objects

4.6.3.2 Data Coloring and Cloud Watermarking

With shared files and data sets, privacy, security, and copyright information could be compromised in a cloud computing environment. Users desire to work in a trusted software environment that provides useful tools to build cloud applications over protected data sets. In the past, watermarking was mainly used for digital copyright management. As shown in Figure 4.35, the system generates special colors for each data object. Data coloring means labeling each data object by a unique color. Differently colored data objects are thus distinguishable.

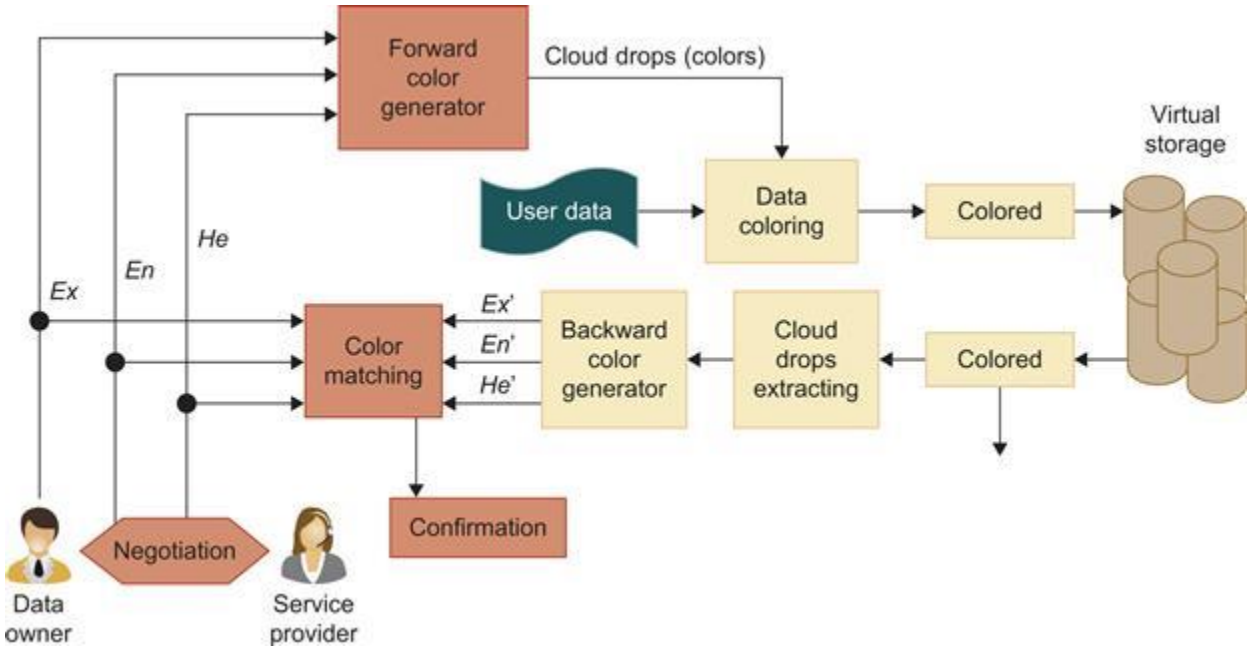


FIGURE 4.35 Data coloring with cloud watermarking for trust management at various security clearance levels in data centers Courtesy of Hwang and Li [36]

The user identification is also colored to be matched with the data colors. This color matching process can be applied to implement different trust management events. Cloud storage provides a process for the generation, embedding, and extraction of the watermarks in colored objects. Interested readers may refer to the articles by Hwang and Li [36] for details on the data coloring and matching process. In general, data protection was done by encryption or decryption which is computationally expensive. The data coloring takes a minimal number of calculations to color or decolor the data objects. Cryptography and watermarking or coloring can be used jointly in a cloud environment.

4.6.3.3 Data Lock-in Problem and Proactive Solutions

Cloud computing moves both the computation and the data to the server clusters maintained by cloud service providers. Once the data is moved into the cloud, users cannot easily extract their data and programs from cloud servers to run on another platform. This leads to a data lock-in problem. This has hindered the use of cloud computing. Data lock-in is attributed to two causes: lack of interoperability, whereby each cloud vendor has its proprietary API that limits users to extract data once submitted; and lack of application compatibility, in that most computing clouds expect users to write new applications from scratch, when they switch cloud platforms.

One possible solution to data lock-in is the use of standardized cloud APIs. This requires building standardized virtual platforms that adhere to OVF, a platform-independent, efficient, extensible, and open format for VMs. This will enable efficient, secure software distribution, facilitating the mobility of VMs. Using OVF one can move data from one application to another. This will enhance QoS, and thus enable cross-cloud applications, allowing workload migration among data centers to user-specific storage. By deploying applications, users can access and intermix applications across different cloud services.

Storage Systems

A cloud provides the vast amounts of storage and computing cycles demanded by many applications. The network-centric content model allows a user to access data stored on a cloud from any device con-nected to the Internet. Mobile devices with limited power reserves and local storage take advantage of cloud environments to store audio and video files. Clouds provide an ideal environment for multimedia content delivery.

A variety of sources feed a continuous stream of data to cloud applications. An ever-increasing number of cloud-based services collect detailed data about their services and information about the users of these services. Then the service providers use the clouds to analyze that data.

Storage and processing on the cloud are intimately tied to one another; indeed, sophisticated strategies to reduce the access time and to support real-time multimedia access are necessary to satisfy the requirements of content delivery. On the other hand, most cloud applications process very large amounts of data; effective data replication and storage management strategies are critical to the computations performed on the cloud.

A new concept, "*big data*," reflects the fact that many applications use data sets so large that they cannot be stored and processed using local resources. The consensus is that "big data" growth can be viewed as a three-dimensional phenomenon; it implies an increased volume of data, requires an increased processing speed to process more data and produce more results, and at the same time it involves a diversity of data sources and data types.

Applications in many areas of science, including genomics, structural biology, high-energy physics, astronomy, meteorology, and the study of the environment, carry out complex analysis of data sets, often of the order of terabytes.[1] In 2010, the four main detectors at the Large Hadron Collider (LHC) produced 13 PB of data; the Sloan Digital Sky Survey (SDSS) collects about 200 GB of data per night. As a result of this increasing appetite for data, file systems such as Btrfs, XFS, ZFS, exFAT, NTFS, HFS Plus, and ReFS support disk formats with theoretical volume sizes of several Exabyte.

Our discussion of cloud storage starts with a review of the storage technology in Section 8.1, followed by an overview of storage models in Section 8.2. Then we follow the evolution of file systems, from distributed to parallel file systems, then to the file systems capable of handling massive amounts of data. In Sections 8.3 and 8.4 we discuss distributed file systems, General Parallel File Systems, and the Google

File System, respectively. A locking service, *Chubby*, based on the Paxos algorithm, is presented in Section 8.7, followed by a discussion of transaction-processing systems and *NoSQL* databases in Section 8.8. The next two sections, Sections 8.9 and 8.10, analyze *BigTable* and *Megastore* systems.

1. The evolution of storage technology

The technological capacity to store information has grown over time at an accelerated pace [164,178]:

- 1986: 2.6 EB; equivalent to less than one 730 MB CD-ROM of data per computer user.
- 1993: 15.8 EB; equivalent to four CD-ROMs per user.
- 2000: 54.5 EB; equivalent to 12 CD-ROMs per user.
- 2007: 295 EB; equivalent to almost 61 CD-ROMs per user.

Though it pales in comparison with processor technology, the evolution of storage technology is astounding. A 2003 study [251] shows that during the 1980–2003 period the storage density of hard disk drives (HDD) increased by four orders of magnitude, from about 0.01 Gb/in^2 to about 100 Gb/in^2. During the same period the prices fell by five orders of magnitude, to about 1 cent/Mbyte. HDD densities are projected to climb to 1,800 Gb/in^2 by 2016, up from 744 Gb/in^2 in 2011.

The density of Dynamic Random Access Memory (DRAM) increased from about 1 Gb/in^2 in 1990 to 100 Gb/in^2 in 2003. The cost of DRAM tumbled from about \$80/MB to less than \$1/MB during the same period. In 2010 Samsung announced the first monolithic, 4 gigabit, low-power, double-data-rate (LPDDR2) DRAM using a 30 nm process.

These rapid technological advancements have changed the balance between initial investment in storage devices and system management costs. Now the cost of storage management is the dominant element of the total cost of a storage system. This effect favors the centralized storage strategy supported by a cloud; indeed, a centralized approach can automate some of the storage management functions, such as replication and backup, and thus reduce substantially the storage management cost.

While the density of storage devices has increased and the cost has decreased dramatically, the access time has improved only slightly. The performance of I/O subsystems has not kept pace with the performance of computing engines, and that affects multimedia, scientific and engineering, and other modern applications that process increasingly large volumes of data.

The storage systems face substantial pressure because the volume of data generated has increased exponentially during the past few decades; whereas in the 1980s and 1990s data was primarily generated by humans, nowadays machines generate data at an unprecedented rate. Mobile devices, such as smart-phones and tablets, record static images, as well as movies and have limited local storage capacity, so they transfer the data to cloud storage systems. Sensors, surveillance cameras, and digital medical imaging devices generate data at a high rate and dump it onto storage systems accessible via the Internet. Online digital libraries, ebooks, and digital media, along with reference data,[2] add to the demand for massive amounts of storage.As the volume of data increases, new methods and algorithms for data mining that require powerful computing systems have been developed. Only a concentration of resources could provide the CPU

cycles along with the vast storage capacity necessary to perform such intensive computations and access the very large volume of data.

Although we emphasize the advantages of a concentration of resources, we have to be acutely aware that a cloud is a large-scale distributed system with a very large number of components that must work in concert. The management of such a large collection of systems poses significant challenges and requires novel approaches to systems design. Case in point: Although the early distributed file systems used custom-designed reliable components, nowadays large-scale systems are built with off-the-shelf components. The emphasis of the design philosophy has shifted from *performance at any cost* to *reliability at the lowest possible cost*. This shift is evident in the evolution of ideas, from the early distributed file systems of the 1980s, such as the Network File System (NFS) and the Andrew File System (AFS), to today's Google File System (GFS) and the *Megastore*.

2.Storage models, file systems, and databases

A *storage model* describes the layout of a data structure in physical storage; a *data model* captures the most important logical aspects of a data structure in a database. The physical storage can be a local disk, a removable media, or storage accessible via a network.

Two abstract models of storage are commonly used: *cell storage* and *journal storage*. Cell storage assumes that the storage consists of cells of the same size and that each object fits exactly in one cell. This model reflects the physical organization of several storage media; the primary memory of a computer is organized as an array of memory cells, and a secondary storage device (e.g., a disk) is organized in sectors or blocks `read` and written as a unit. `read/write` *coherence* and *before-or-after atomicity* are two highly desirable properties of any storage model and in particular of cell storage (see Figure 8.1).

Journal storage is a fairly elaborate organization for storing composite objects such as records consisting of multiple fields. Journal storage consists of a *manager* and *cell storage*, where the entire

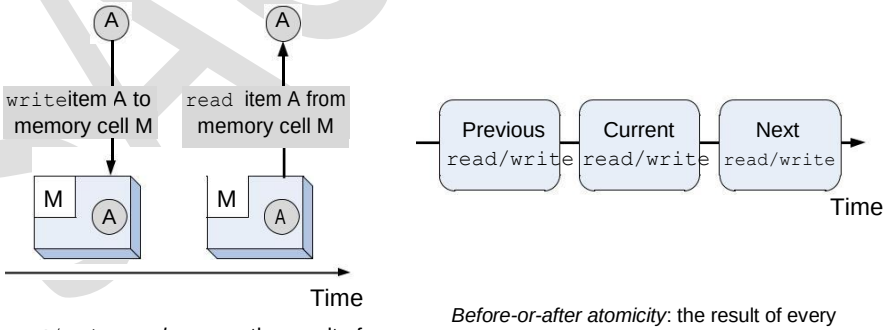


FIGURE 8.1

Illustration capturing the semantics of `read/write` *coherence* and *before-or-after atomicity*.

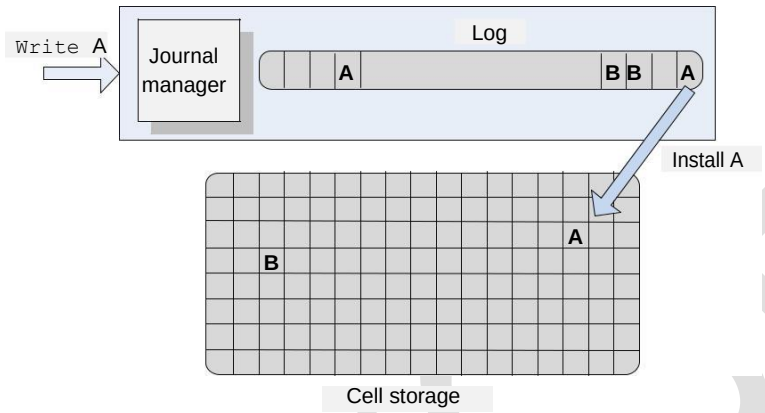


FIGURE 8.2

A *log* contains the entire history of all variables. The log is stored on nonvolatile media of *journal storage*. If the system fails after the new value of a variable is stored in the log but before the value is stored in cell memory, then the value can be recovered from the log. If the system fails while writing the log, the cell memory is not updated. This guarantees that all actions are *all-or-nothing*. Two variables, **A** and **B**, in the log and cell storage are shown. A new value of **A** is written first to the *log* and then *installed* on cell memory at the unique address assigned to **A**.

history of a variable is maintained, rather than just the current value. The user does not have direct access to the *cell storage*; instead the user can request the *journal manager* to (i) start a new action; (ii) `read` the value of a cell; (iii) `write` the value of a cell; (iv) commit an action; or (v) abort an action. The *journal manager* translates user requests to commands sent to the cell storage: (i) `read` a cell; (ii) `write` a cell; (iii) allocate a cell; or (iv) deallocate a cell.

In the context of storage systems, a *log* contains a history of all variables in *cell storage*. The information about the updates of each data item forms a record appended at the end of the log. A log provides authoritative information about the outcome of an action involving *cell storage*; the cell storage can be reconstructed using the log, which can be easily accessed – we only need a pointer to the last record.

An *all-or-nothing* action first records the action in a *log* in *journal storage* and then *installs* the change in the *cell storage* by overwriting the previous version of a data item (see Figure 8.2). The *log* is always kept on nonvolatile storage (e.g., disk) and the considerably larger *cell storage* resides typically on nonvolatile memory, but can be held in memory for real-time access or using a write-through cache.

Many cloud applications must support online transaction processing and have to guarantee the correctness of the transactions. Transactions consist of multiple actions; for example, the transfer of funds from one account to another requires withdrawing funds from one account and crediting it to another. The system may fail during or after each one of the actions, and steps to ensure correctness must be taken. Correctness of a transaction means that the result should be guaranteed to be the same as though the actions were applied one after another, regardless of the order. More stringent conditions must

sometimes be observed; for example, banking transactions must be processed in the order in which they are issued, the so-called *external time consistency*. To guarantee correctness, a transaction-processing system supports *all-or-nothing atomicity*, discussed in Section 2.10.

A *file system* consists of a collection of *directories*. Each directory provides information about a set of files. Today high-performance systems can choose among three classes of file system: network file systems (NFSs), storage area networks (SANs), and parallel file systems (PFSs). The NFS is very popular and has been used for some time, but it does not scale well and has reliability problems; an NFS server could be a single point of failure.

Advances in networking technology allow the separation of storage systems from computational servers; the two can be connected by a SAN. SANs offer additional flexibility and allow cloud servers to deal with nondisruptive changes in the storage configuration. Moreover, the storage in a SAN can be *pooled* and then allocated based on the needs of the servers; pooling requires additional software and hardware support and represents another advantage of a centralized storage system. A SAN-based implementation of a file system can be expensive, since each node must have a Fibre Channel adapter to connect to the network.

Parallel file systems are scalable, are capable of distributing files across a large number of nodes, and provide a global naming space. In a parallel data system, several I/O nodes serve data to all computational nodes and include a metadata server that contains information about the data stored in the I/O nodes. The interconnection network of a parallel file system could be a SAN.

Most cloud applications do not interact directly with file systems but rather through an application layer that manages a database. A *database* is a collection of logically related records. The software that controls the access to the database is called a *database management system (DBMS)*. The main functions of a DBMS are to enforce data integrity, manage data access and concurrency control, and support recovery after a failure.

A DBMS supports a *query language*, a dedicated programming language used to develop database applications. Several database models, including the navigational model of the 1960s, the relational model of the 1970s, the object-oriented model of the 1980s, and the *NoSQL* model of the first decade of the 2000s, reflect the limitations of the hardware available at the time and the requirements of the most popular applications of each period.

Most cloud applications are data intensive and test the limitations of the existing infrastructure. For example, they demand DBMSs capable of supporting rapid application development and short time to market. At the same time, cloud applications require low latency, scalability, and high availability and demand a consistent view of the data.

These requirements cannot be satisfied simultaneously by existing database models; for example, relational databases are easy to use for application development but do not scale well. As its name implies, the *NoSQL* model does not support SQL as a query language and may not guarantee the *atomicity*, *consistency*, *isolation*, *durability* (ACID) properties of traditional databases. *NoSQL* usually guarantees the eventual consistency for transactions limited to a single data item. The *NoSQL* model is useful when the structure of the data does not require a relational model and the amount of data is very large. Several types of *NoSQL* database have emerged in the last few years. Based on the way the *NoSQL* databases store data, we recognize several types, such as key-value stores, *BigTable* implementations, document store databases, and graph databases.

Replication, used to ensure fault tolerance of large-scale systems built with commodity components, requires mechanisms to guarantee that all replicas are consistent with one another. This is another example of increased complexity of modern computing and communication systems due to physical characteristics of components, a topic discussed in Chapter 10. Section 8.7 contains an in-depth analysis of a service implementing a consensus algorithm to guarantee that replicated objects are consistent.

3.Distributed file systems: The precursors

In this section we discuss the first distributed file systems, developed in the 1980s by software companies and universities. The systems covered are the Network File System developed by Sun Microsystems in 1984, the Andrew File System developed at Carnegie Mellon University as part of the Andrew project, and the Sprite Network File System developed by John Osterhout's group at UC Berkeley as a component of the *Unix*-like distributed operating system called Sprite. Other systems developed at about the same time are Locus [365], Apollo [211], and the Remote File System (RFS) [35]. The main concerns in the design of these systems were scalability, performance, and security (see Table 8.1.)

In the 1980s many organizations, including research centers, universities, financial institutions, and design centers, considered networks of workstations to be an ideal environment for their operations. Diskless workstations were appealing due to reduced hardware costs and because of lower maintenance and system administration costs. Soon it became obvious that a distributed file system could be very useful for the management of a large number of workstations. Sun Microsystems, one of the main promoters of distributed systems based on workstations, proceeded to develop the NFS in the early 1980s.

Network File System (NFS). NFS was the first widely used distributed file system; the development of this application based on the client-server model was motivated by the need to share a file system among a number of clients interconnected by a local area network.

Table 8.1 A comparison of several network file systems [250].

File System	Cache Size and Location	Writing Policy	Consistency Guarantees	Cache Validation
NFS	Fixed, memory	On close or 30 s delay	Sequential	On open, with server consent
AFS	Fixed, disk	On close	Sequential	When modified server asks client
Sprite	Variable, memory	30 s delay	Sequential, concurrent	On open, with server consent
Locus	Fixed, memory	On close	Sequential, concurrent	On open, with server consent
Apollo	Variable, memory	Delayed or on unlock	Sequential	On open, with server consent
RFS	Fixed, memory	Write-through	Sequential, concurrent	On open, with server consent

A majority of workstations were running under *Unix*; thus, many design decisions for the NFS were influenced by the design philosophy of the *Unix File System* (UFS). It is not surprising that the NFS designers aimed to:

- Provide the same semantics as a local UFS to ensure compatibility with existing applications.
- Facilitate easy integration into existing UFS.
- Ensure that the system would be widely used and thus support clients running on different operating systems.
- Accept a modest performance degradation due to remote access over a network with a bandwidth of several Mbps.

Before we examine NFS in more detail, we have to analyze three important characteristics of the *Unix* File System that enabled the extension from local to remote file management:

- The layered design provides the necessary *flexibility* for the file system; layering allows separation of concerns and minimization of the interaction among the modules necessary to implement the system. The addition of the *vnode* layer allowed the *Unix* File System to treat local and remote file access uniformly.
- The hierarchical design supports *scalability* of the file system; indeed, it allows grouping of files into special files called *directories* and supports multiple levels of directories and collections of directories and files, the so-called *file systems*. The hierarchical file structure is reflected by the file-naming convention.
- The metadata supports a systematic rather than an ad hoc design philosophy of the file system. The so called *inodes* contain information about individual files and directories. The inodes are kept on persistent media, together with the data. Metadata includes the file owner, the access rights, the creation time or the time of the last modification of the file, the file size, and information about the structure of the file and the persistent storage device cells where data is stored. Metadata also supports device independence, a very important objective due to the very rapid pace of storage technology development.

The *logical organization* of a file reflects the data model – the view of the data from the perspective of the application. The *physical organization* reflects the storage model and describes the manner in which the file is stored on a given storage medium. The layered design allows UFS to separate concerns for the physical file structure from the logical one.

Recall that a *file* is a linear array of cells stored on a persistent storage device. The *file pointer* identifies a cell used as a starting point for a `read` or `write` operation. This linear array is viewed by an application as a collection of logical records; the file is stored on a physical device as a set of physical records, or blocks, of a size dictated by the physical media.

The lower three layers of the UFS hierarchy – the block, the file, and the inode layer – reflect the physical organization. The block layer allows the system to locate individual blocks on the phys-ical device; the file layer reflects the organization of blocks into files; and the inode layer provides the metadata for the objects (files and directories). The upper three layers – the path name, the absolute path name, and the symbolic path name layer – reflect the logical organization. The file-name layer mediates between the machine-oriented and the user-oriented views of the file system (see Figure 8.3).

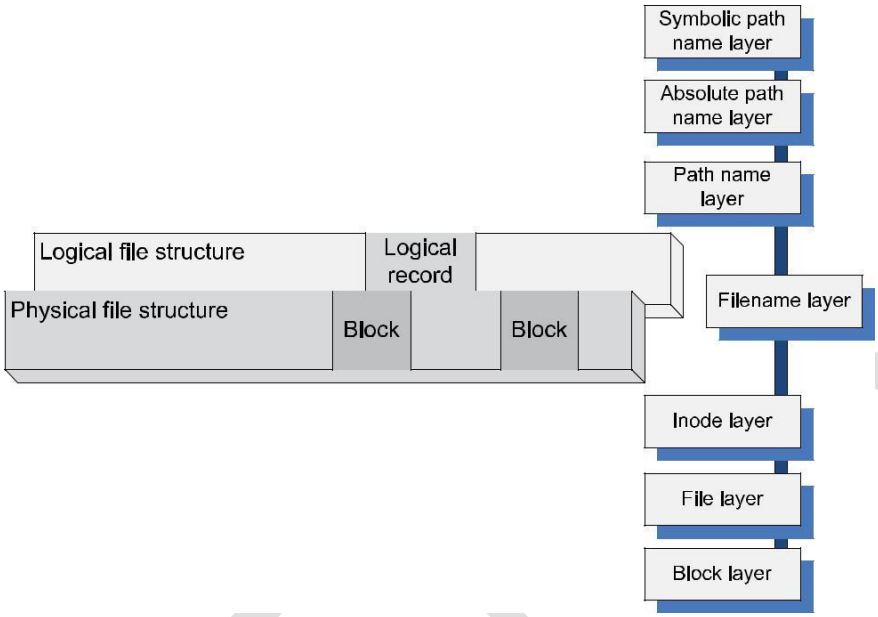


FIGURE 8.3

The layered design of the *Unix* File System separates the physical file structure from the logical one. The lower three layers – block, file, and inode – are related to the physical file structure, while the upper three layers – path name, absolute path name, and symbolic path name – reflect the logical organization. The filename layer mediates between the two.

Several *control structures* maintained by the kernel of the operating system support file handling by a running process. These structures are maintained in the user area of the process address space and can only be accessed in kernel mode. To access a file, a process must first establish a connection with the file system by opening the file. At that time a new entry is added to the *file description table*, and the meta-information is brought into another control structure, the *open file table*.

A *path* specifies the location of a file or directory in a file system; a *relative path* specifies this location relative to the current/working directory of the process, whereas a *full path*, also called an *absolute path*, specifies the location of the file independently of the current directory, typically relative to the root directory. A local file is uniquely identified by a *file descriptor (fd)*, generally an index in the open file table.

The Network File System is based on the client-server paradigm. The client runs on the local host while the server is at the site of the remote file system, and they interact by means of remote procedure calls (RPCs) (see Figure 8.4). The API interface of the local file system distinguishes file operations on a local file from the ones on a remote file and, in the latter case, invokes the RPC client. Figure 8.5 shows the API for a *Unix* File System, with the calls made by the RPC client in response to API calls issued by a user program for a remote file system as well as some of the actions carried out by the NFS

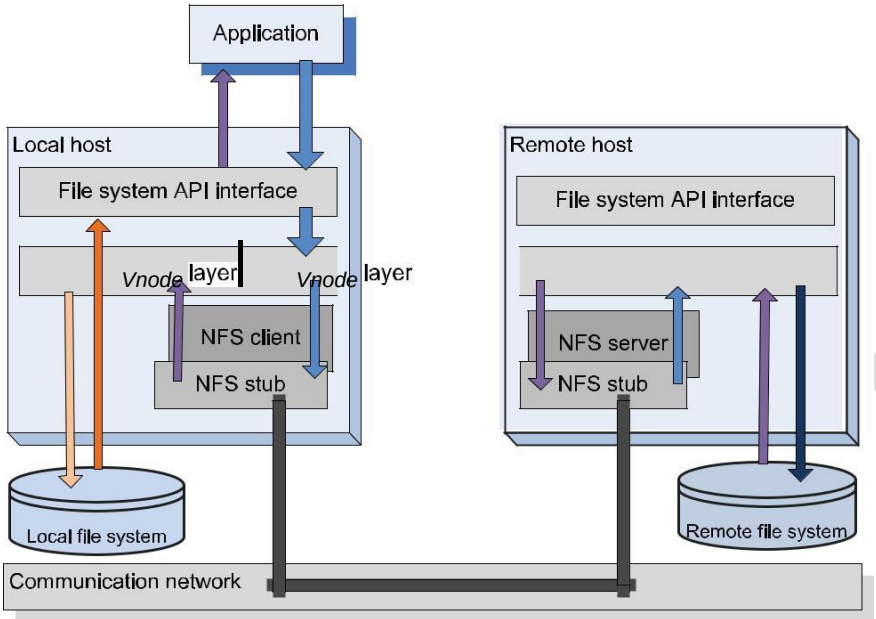


FIGURE 8.4

The NFS client-server interaction. The *vnode* layer implements file operation in a uniform manner, regardless of whether the file is local or remote. An operation targeting a local file is directed to the local file system, whereas one for a remote file involves NFS. An NSF client packages the relevant information about the target and the NFS server passes it to the *vnode* layer on the remote host, which, in turn, directs it to the remote file system.

server in response to an RPC call. NFS uses a *vnode* layer to distinguish between operations on local and remote files, as shown in Figure 8.4.

A remote file is uniquely identified by a *file handle (fh)* rather than a file descriptor. The file handle is a 32-byte internal name, a combination of the file system identification, an inode number, and a generation number. The file handle allows the system to locate the remote file system and the file on that system; the generation number allows the system to reuse the inode numbers and ensures correct semantics when multiple clients operate on the same remote file.

Although many RPC calls, such as `read`, are idempotent,[3] communication failures could sometimes lead to unexpected behavior. Indeed, if the network fails to deliver the response to a `read` RPC, then the call can be repeated without any side effects. By contrast, when the network fails to deliver the response to the `rmdir` RPC, the second call returns an error code to the user if the call was successful the first time. If the server fails to execute the first call, the second call returns normally. Note also that there is no `close` RPC because this action only makes changes in the process open file structure and does not affect the remote file.

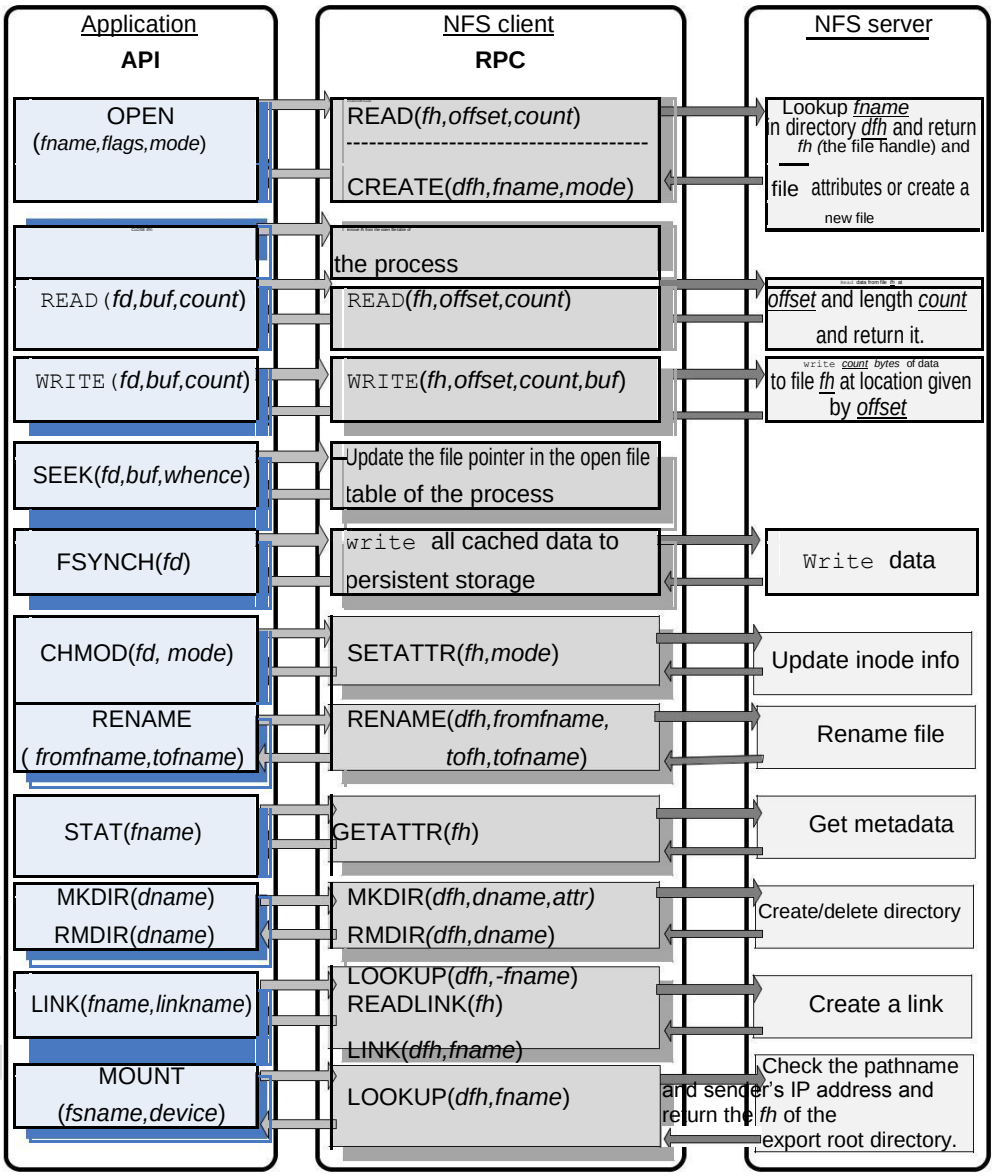


FIGURE 8.5

The API of the *Unix* File System and the corresponding RPC issued by an NFS client to the NFS server. The actions of the server in response to an RPC issued by the NFS client are too complex to be fully described. *fd* stands for file descriptor, *fh* for file handle, *fname* for filename, *dname* for directory name, *dfh* for the directory where the file handle can be found, *count* for the number of bytes to be transferred, *buf* for the buffer to transfer the data to/from, and *device* for the device on which the file system is located *fsname* (stands for files system name).

The NFS has undergone significant transformations over the years. It has evolved from Version 2 [314], discussed in this section, to Version 3 [286] in 1994 and then to Version 4 [287] in 2000 (see Section 8.11).

Andrew File System (AFS). AFS is a distributed file system developed in the late 1980s at Carnegie Mellon University (CMU) in collaboration with IBM [250]. The designers of the system envisioned a very large number of workstations interconnected with a relatively small number of servers; it was anticipated that each individual at CMU would have an Andrew workstation, so the system would connect up to 10,000 workstations. The set of trusted servers in AFS forms a structure called Vice. The OS on a workstation, 4.2 BSD *Unix*, intercepts file system calls and forwards them to a user-level process called Venus, which caches files from Vice and stores modified copies of files back on the servers they came from. Reading and writing from/to a file are performed directly on the cached copy and bypass Venus; only when a file is opened or closed does Venus communicate with Vice.

The emphasis of the AFS design was on performance, security, and simple management of the file system [170]. To ensure scalability and to reduce response time, the local disks of the workstations are used as persistent cache. The master copy of a file residing on one of the servers is updated only when the file is modified. This strategy reduces the load placed on the servers and contributes to better system performance.

Another major objective of the AFS design was improved security. The communications between clients and servers are encrypted, and all file operations require secure network connections. When a user signs into a workstation, the password is used to obtain security tokens from an authentication server. These tokens are then used every time a file operation requires a secure network connection.

The AFS uses *access control lists* (ACLs) to allow control sharing of the data. An ACL specifies the access rights of an individual user or group of users. A set of tools supports ACL management. Another facet of the effort to reduce user involvement in file management is *location transparency*. The files can be accessed from any location and can be moved automatically or at the request of system administrators without user involvement and/or inconvenience. The relatively small number of servers drastically reduces the efforts related to system administration because operations, such as backups, affect only the servers, whereas workstations can be added, removed, or moved from one location to another without administrative intervention.

Sprite Network File System (SFS). SFS is a component of the Sprite network operating system [165]. SFS supports non-write-through caching of files on the client as well as the server systems [255]. Processes running on all workstations enjoy the same semantics for file access as they would if they were run on a single system. This is possible due to a cache consistency mechanism that flushes portions of the cache and disables caching for shared files opened for `read`/`write` operations.

Caching not only hides the network latency, it also reduces server utilization and obviously improves performance by reducing response time. A file access request made by a client process could be satisfied at different levels. First, the request is directed to the local cache; if it's not satisfied there, it is passed to the local file system of the client. If it cannot be satisfied locally then the request is sent to the remote server. If the request cannot be satisfied by the remote server's cache, it is sent to the file system running on the server.

The design decisions for the Sprite system were influenced by the resources available at a time when a typical workstation had a 1–2 MIPS processor and 4–14 Mbytes of physical memory. The main-memory

caches allowed diskless workstations to be integrated into the system and enabled the development of unique caching mechanisms and policies for both clients and servers. The results of a file-intensive benchmark report [255] show that SFS was 30–35% faster than either NFS or AFS.

The file cache is organized as a collection of 4 KB blocks; a cache block has a virtual address consisting of a unique file identifier supplied by the server and a block number in the file. Virtual addressing allows the clients to create new blocks without the need to communicate with the server. File servers map virtual to physical disk addresses. Note that the page size of the virtual memory in Sprite is also 4K.

The size of the cache available to an SFS client or a server system changes dynamically as a function of the needs. This is possible because the Sprite operating system ensures optimal sharing of the physical memory between file caching by SFS and virtual memory management.

The file system and the virtual memory manage separate sets of physical memory pages and maintain a time of last access for each block or page, respectively. Virtual memory uses a version of the clock algorithm [254] to implement a least recently used (LRU) page replacement algorithm, and the file system implements a strict LRU order, since it knows the time of each `read` and `write` operation. Whenever the file system or the virtual memory management experiences a file cache miss or a page fault, it compares the age of its oldest cache block or page, respectively, with the age of the oldest one of the other system; the oldest cache block or page is forced to release the real memory frame.

An important design decision related to the SFS was to *delay write-backs;* this means that a block is first written to cache, and the writing to the disk is delayed for a time of the order of tens of seconds. This strategy speeds up writing and avoids writing when the data is discarded before the time to `write` it to the disk. The obvious drawback of this policy is that data can be lost in case of a system failure. *Write-through* is the alternative to the delayed write-back; it guarantees reliability because the block is written to the disk as soon as it is available on the cache, but it increases the time for a write operation.

Most network file systems guarantee that once a file is closed, the server will have the newest version on persistent storage. As far as concurrency is concerned, we distinguish *sequential write sharing*, when a file cannot be opened simultaneously for reading and writing by several clients, from *concurrent write sharing*, when multiple clients can modify the file at the same time. Sprite allows both modes of concurrency and delegates the cache consistency to the servers. In case of concurrent `write` sharing, the client caching for the file is disabled; all `reads` and `writes` are carried out through the server.

Table 8.1 presents a comparison of caching, writing strategy, and consistency of NFS, AFS [250], Sprite [165], Locus [365], Apollo [211], and the Remote File System (RFS) [35].

4.General Parallel File System

Parallel I/O implies execution of multiple input/output operations concurrently. Support for parallel I/O is essential to the performance of many applications [236]. Therefore, once distributed file systems became ubiquitous, the natural next step in the evolution of the file system was to support parallel access. Parallel file systems allow multiple clients to `read` and `write` concurrently from the same file.

Concurrency control is a critical issue for parallel file systems. Several semantics for handling the shared access are possible. For example, when the clients share the file pointer, successive reads issued by multiple clients advance the file pointer; another semantic is to allow each client to have its own

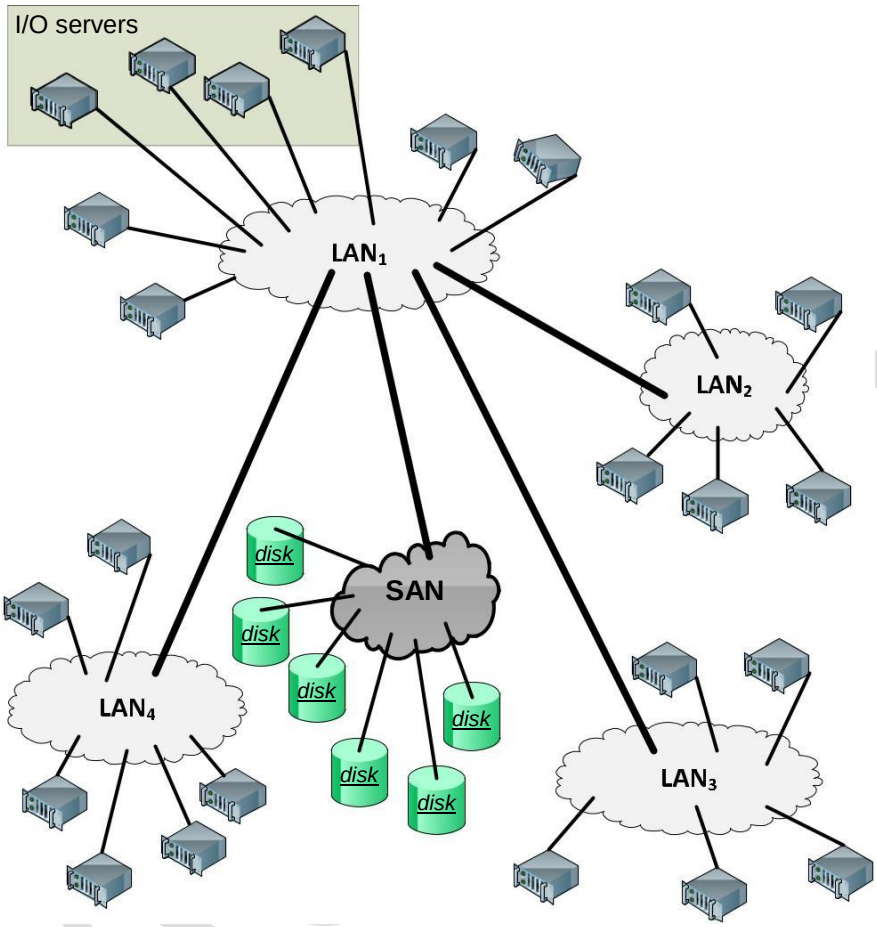


FIGURE 8.6

A GPFS configuration. The disks are interconnected by a SAN and compute servers are distributed in four LANs, *LAN*1–*LAN*4. The I/O nodes/servers are connected to *LAN*1.

file pointer. Early supercomputers such as the Intel Paragon[4] took advantage of parallel file systems to support applications based on the same program, multiple data (SPMD) paradigm.

The General Parallel File System (GPFS) [317] was developed at IBM in the early 2000s as a successor to the TigerShark multimedia file system [159]. GPFS is a parallel file system that emulates closely the behavior of a general-purpose POSIX system running on a single system. GPFS was designed for optimal performance of large clusters; it can support a file system of up to 4 PB consisting of up to 4**,** 096 disks of 1 TB each (see Figure 8.6).

The maximum file size is $(2^{63} - 1)$ bytes. A file consists of blocks of equal size, ranging from 16 KB to 1 MB striped across several disks. The system could support not only very large files but also a very large number of files. The directories use *extensible hashing* techniques[5] to access a file. The system maintains user data, file metadata such as the time when last modified, and file system metadata such as allocation maps. Metadata, such as file attributes and data block addresses, is stored in inodes and indirect blocks.

Reliability is a major concern in a system with many physical components. To recover from system failures, GPFS records all metadata updates in a *write-ahead* log file. *Write-ahead* means that updates are written to persistent storage only after the log records have been written. For example, when a new file is created, a directory block must be updated and an inode for the file must be created. These records are transferred from cache to disk after the log records have been written. When the directory block is written and then the I/O node fails before writing the inode, then the system ends up in an inconsistent state and the log file allows the system to recreate the inode record.

The log files are maintained by each I/O node for each file system it mounts; thus, any I/O node is able to initiate recovery on behalf of a failed node. Disk parallelism is used to reduce access time. Multiple I/O `read` requests are issued in parallel and data is prefetched in a buffer pool.

Data striping allows concurrent access and improves performance but can have unpleasant side-effects. Indeed, when a single disk fails, a large number of files are affected. To reduce the impact of such undesirable events, the system attempts to mask a single disk failure or the failure of the access path to a disk. The system uses RAID devices with the stripes equal to the block size and dual-attached RAID controllers. To further improve the fault tolerance of the system, GPFS data files as well as metadata are replicated on two different physical disks.

Consistency and performance, critical to any distributed file system, are difficult to balance. Support for concurrent access improves performance but faces serious challenges in maintaining consistency. In GPFS, consistency and synchronization are ensured by a distributed locking mechanism; a *central lock manager* grants *lock tokens* to *local lock managers* running in each I/O node. Lock tokens are also used by the cache management system.

Lock granularity has important implications in the performance of a file system, and GPFS uses a variety of techniques for various types of data. *Byte-range tokens* are used for `read` and `write` operations to data files as follows: The first node attempting to `write` to a file acquires a token covering the entire file, $[0, \infty]$. This node is allowed to carry out all `reads` and `writes` to the file without any need for permission until a second node attempts to write to the same file. Then the range of the token given to the first node is restricted. More precisely, if the first node writes sequentially at offset fp_1 and the second one at offset $fp_2 > fp_1$, the range of the tokens for the two tokens are $[0, fp_2]$ and $[fp_2, \infty]$, respectively, and the two nodes can operate concurrently, without the need for further negotiations. Byte-range tokens are rounded to block boundaries.

Byte-range token negotiations among nodes use the *required range* and the *desired range* for the offset and for the length of the current and future operations, respectively. *Data shipping*, an alternative to byte-range locking, allows fine-grained data sharing. In this mode the file blocks are controlled by the I/O nodes in a round-robin manner. A node forwards a `read` or `write` operation to the node controlling the target block, the only one allowed to access the file.

.

A *token manager* maintains the state of all tokens; it creates and distributes tokens, collects tokens once a file is closed, and downgrades or upgrades tokens when additional nodes request access to a file. Token management protocols attempt to reduce the load placed on the token manager; for example, when a node wants to revoke a token, it sends messages to all the other nodes holding the token and forwards the reply to the token manager.

Access to metadata is synchronized. For example, when multiple nodes `write` to the same file, the file size and the modification dates are updated using a *shared write lock* to access an inode. One of the nodes assumes the role of a *metanode*, and all updates are channeled through it. The file size and the last update time are determined by the metanode after merging the individual requests. The same strategy is used for updates of the indirect blocks. GPFS global data such as access control lists (ACLs), quotas, and configuration data are updated using the distributed locking mechanism.

GPFS uses *disk maps* to manage the disk space. The GPFS block size can be as large as 1 MB, and a typical block size is 256 KB. A block is divided into 32 subblocks to reduce disk fragmentation for small files; thus, the block map has 32 bits to indicate whether a subblock is free or used. The system disk map is partitioned into *n* regions, and each disk map region is stored on a different I/O node. This strategy reduces conflicts and allows multiple nodes to allocate disk space at the same time. An *allocation manager* running on one of the I/O nodes is responsible for actions involving multiple disk map regions. For example, it updates free space statistics and helps with deallocation by sending periodic hints of the regions used by individual nodes.

A detailed discussion of system utilities and the lessons learned from the deployment of the file sys-tem at several installations in 2002 can be found in [317]; the documentation of the GPFS is available from [177].

5.Google File System

The Google File System (GFS) was developed in the late 1990s. It uses thousands of storage systems built from inexpensive commodity components to provide petabytes of storage to a large user community with diverse needs [136]. It is not surprising that a main concern of the GFS designers was to ensure the reliability of a system exposed to hardware failures, system software errors, application errors, and last but not least, human errors.

The system was designed after a careful analysis of the file characteristics and of the access models. Some of the most important aspects of this analysis reflected in the GFS design are:

- Scalability and reliability are critical features of the system; they must be considered from the beginning rather than at some stage of the design.
- The vast majority of files range in size from a few GB to hundreds of TB.
- The most common operation is to `append` to an existing file; random `write` operations to a file are extremely infrequent.
- Sequential `read` operations are the norm.
- The users process the data in bulk and are less concerned with the response time.
- The consistency model should be relaxed to simplify the system implementation, but without placing an additional burden on the application developers.

Several design decisions were made as a result of this analysis:

1. Segment a file in large chunks.
2. Implement an atomic file `append` operation allowing multiple applications operating concurrently to `append` to the same file.
3. Build the cluster around a high-bandwidth rather than low-latency interconnection network. Separate the flow of control from the data flow; schedule the high-bandwidth data flow by pipelining the data transfer over TCP connections to reduce the response time. Exploit network topology by sending data to the closest node in the network.
4. Eliminate caching at the client site. Caching increases the overhead for maintaining consistency among cached copies at multiple client sites and it is not likely to improve performance.
5. Ensure consistency by channeling critical file operations through a *master*, a component of the cluster that controls the entire system.
6. Minimize the involvement of the *master* in file access operations to avoid hot-spot contention and to ensure scalability.
7. Support efficient checkpointing and fast recovery mechanisms.
8. Support an efficient garbage-collection mechanism.

GFS files are collections of fixed-size segments called *chunks*; at the time of file creation each chunk is assigned a unique *chunk handle*. A chunk consists of 64 KB blocks and each block has a 32-bit checksum. Chunks are stored on *Linux* files systems and are replicated on multiple sites; a user may change the number of the replicas from the standard value of three to any desired value. The chunk size is 64 MB; this choice is motivated by the desire to optimize performance for large files and to reduce the amount of metadata maintained by the system.

A large chunk size increases the likelihood that multiple operations will be directed to the same chunk; thus it reduces the number of requests to locate the chunk and, at the same time, it allows the application to maintain a persistent network connection with the server where the chunk is located. Space fragmentation occurs infrequently because the chunk for a small file and the last chunk of a large file are only partially filled.

The architecture of a GFS cluster is illustrated in Figure 8.7. A *master* controls a large number of *chunk servers*; it maintains metadata such as filenames, access control information, the location of all the replicas for every chunk of each file, and the state of individual chunk servers. Some of the metadata is stored in persistent storage (e.g., the *operation log* records the file namespace as well as the file-to-chunk mapping).

The locations of the chunks are stored only in the control structure of the *master*'s memory and are updated at system startup or when a new chunk server joins the cluster. This strategy allows the *master* to have up-to-date information about the location of the chunks.

System reliability is a major concern, and the operation log maintains a historical record of metadata changes, enabling the *master* to recover in case of a failure. As a result, such changes are atomic and are not made visible to the clients until they have been recorded on multiple replicas on persistent storage. To recover from a failure, the *master* replays the operation log. To minimize the recovery time, the *master* periodically checkpoints its state and at recovery time replays only the log records after the last checkpoint.

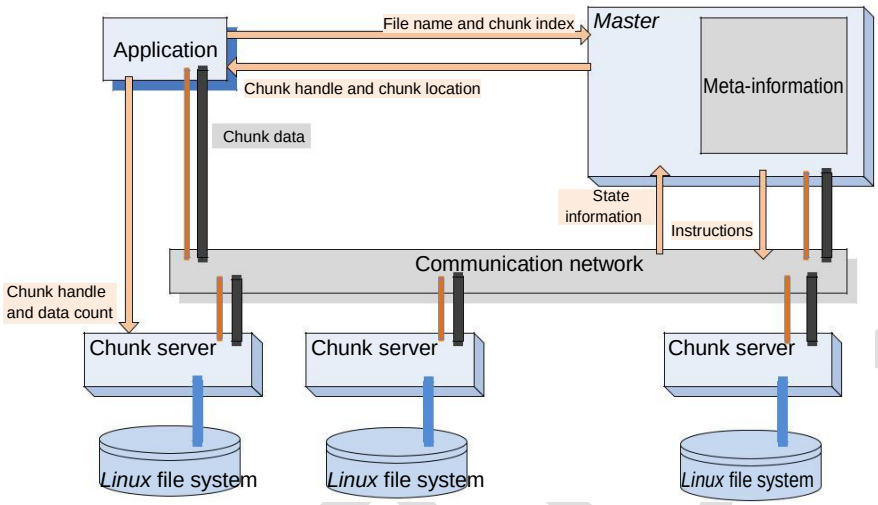


FIGURE 8.7

The architecture of a GFS cluster. The *master* maintains state information about all system components; it controls a number of *chunk servers*. A chunk server runs under *Linux*; it uses metadata provided by the *master* to communicate directly with the application. The data flow is decoupled from the control flow. The data and the control paths are shown separately, data paths with thick lines and control paths with thin lines. Arrows show the flow of control among the application, the *master*, and the chunk servers.

Each chunk server is a commodity *Linux* system; it receives instructions from the *master* and responds with status information. To access a file, an application sends to the *master* the filename and the chunk index, the offset in the file for the `read` or `write` operation; the *master* responds with the chunk handle and the location of the chunk. Then the application communicates directly with the chunk server to carry out the desired file operation.

The consistency model is very effective and scalable. Operations, such as file creation, are atomic and are handled by the *master*. To ensure scalability, the *master* has minimal involvement in file mutations and operations such as `write` or `append` that occur frequently. In such cases the *master* grants a lease for a particular chunk to one of the chunk servers, called the *primary*; then, the primary creates a serial order for the updates of that chunk.

When data for a `write` straddles the chunk boundary, two operations are carried out, one for each chunk. The steps for a `write` request illustrate a process that buffers data and decouples the control flow from the data flow for efficiency:

1. The client contacts the *master*, which assigns a lease to one of the chunk servers for a particular chunk if no lease for that chunk exists; then the *master* replies with the ID of the primary as well as secondary chunk servers holding replicas of the chunk. The client caches this information.
2. The client sends the data to all chunk servers holding replicas of the chunk; each one of the chunk servers stores the data in an internal LRU buffer and then sends an acknowledgment to the client.

3. The client sends a `write` request to the primary once it has received the acknowledgments from all chunk servers holding replicas of the chunk. The primary identifies mutations by consecutive sequence numbers.
4. The primary sends the `write` requests to all secondaries.
5. Each secondary applies the mutations in the order of the sequence numbers and then sends an acknowledgment to the primary.
6. Finally, after receiving the acknowledgments from all secondaries, the primary informs the client.

The system supports an efficient checkpointing procedure based on *copy-on-write* to construct system snapshots. A lazy garbage collection strategy is used to reclaim the space after a file deletion. In the first step the filename is changed to a hidden name and this operation is time stamped. The *master* periodically scans the namespace and removes the metadata for the files with a hidden name older than a few days; this mechanism gives a window of opportunity to a user who deleted files by mistake to recover the files with little effort.

Periodically, chunk servers exchange with the *master* the list of chunks stored on each one of them; the *master* supplies them with the identity of orphaned chunks whose metadata has been deleted, and such chunks are then deleted, Even when control messages are lost, a chunk server will carry out the housecleaning at the next *heartbeat* exchange with the *master*. Each chunk server maintains in core the checksums for the locally stored chunks to guarantee data integrity.

CloudStore is an open-source C++ implementation of GFS that allows client access not only from C++ but also from Java and Python.

6. *Apache Hadoop*

A wide range of data-intensive applications such as marketing analytics, image processing, machine learning, and Web crawling use *Apache Hadoop*, an open-source, Java-based software system.[6] *Hadoop* supports distributed applications handling extremely large volumes of data. Many members of the community contributed to the development and optimization of *Hadoop* and several related Apache projects such as *Hive* and *HBase*.

Hadoop is used by many organizations from industry, government, and research; the long list of *Hadoop* users includes major IT companies such as Apple, IBM, HP, Microsoft, Yahoo!, and Amazon; media companies such as The New York Times and Fox; social networks, including Twitter, Facebook, and LinkedIn; and government agencies, such as the U.S. Federal Reserve. In 2011, the Facebook *Hadoop* cluster had a capacity of 30 PB.

A *Hadoop* system has two components, a *MapReduce* engine and a database (see Figure 8.8). The database could be the *Hadoop File System (HDFS)*, Amazon *S3*, or *CloudStore*, an implementation of the Google File System discussed in Section 8.5. *HDFS* is a distributed file system written in Java; it is portable, but it cannot be directly mounted on an existing operating system. *HDFS* is not fully POSIX compliant, but it is highly performant.

The *Hadoop* engine on the master of a multinode cluster consists of a *job tracker* and a *task tracker*, whereas the engine on a slave has only a *task tracker*. The *job tracker* receives a *MapReduce* job

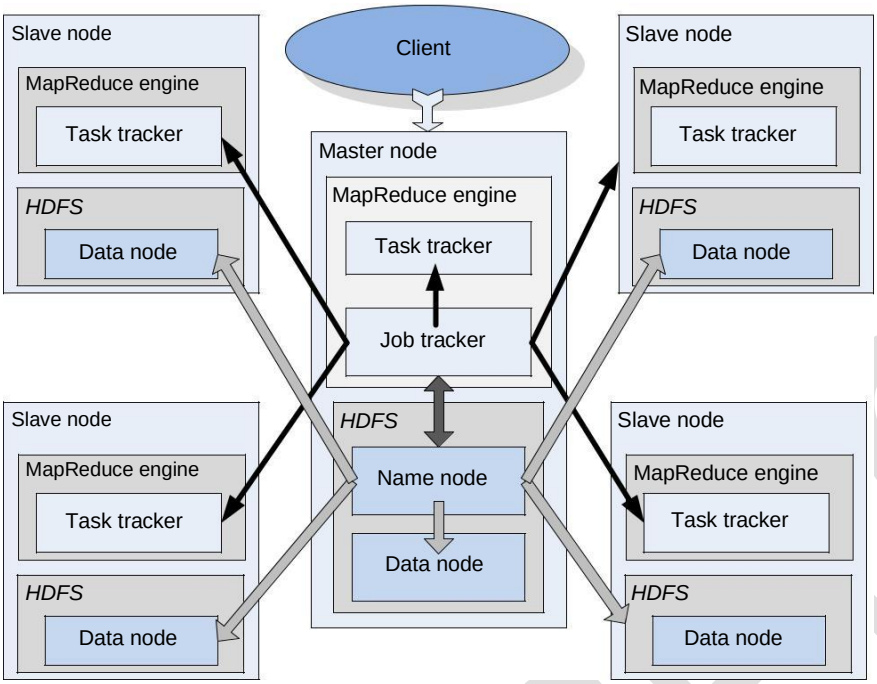


FIGURE 8.8

A *Hadoop* cluster using *HDFS*. The cluster includes a master and four slave nodes. Each node runs a *MapReduce* engine and a database engine, often *HDFS*. The *job tracker* of the master's engine communicates with the *task trackers* on all the nodes and with the *name node* of *HDFS*. The *name node* of *HDFS* shares information about data placement with the *job tracker* to minimize communication between the nodes on which data is located and the ones where it is needed.

from a client and dispatches the work to the *task trackers* running on the nodes of a cluster. To increase efficiency, the *job tracker* attempts to dispatch the tasks to available slaves closest to the place where it stored the task data. The *task tracker* supervises the execution of the work allocated to the node. Several scheduling algorithms have been implemented in *Hadoop* engines, including Facebook's fair scheduler and Yahoo!'s capacity scheduler; see Section 6.8 for a discussion of cloud scheduling algorithms.

HDFS replicates data on multiple nodes. The default is three replicas; a large dataset is distributed over many nodes. The *name node* running on the master manages the data distribution and data replication and communicates with *data nodes* running on all cluster nodes; it shares with the *job tracker* information about data placement to minimize communication between the nodes on which data is located and the ones where it is needed. Although *HDFS* can be used for applications other than those based on the *MapReduce* model, its performance for such applications is not at par with the ones for which it was originally designed.

7.Locks and *Chubby*: A locking service

Locks support the implementation of reliable storage for loosely coupled distributed systems; they enable controlled access to shared storage and ensure atomicity of `read` and `write` operations. Furthermore, critically important to the design of reliable distributed storage systems are distributed consensus prob-lems, such as the election of a master from a group of data servers. A master has an important role in system management; for example, in GFS the master maintains state information about all system components.

Locking and the election of a master can be done using a version of the Paxos algorithm for asynchronous consensus, discussed in Section 2.11. The algorithm guarantees safety without any timing assumptions, a necessary condition in a large-scale system when communication delays are unpredictable. Nevertheless, the algorithm must use clocks to ensure liveliness and to overcome the impossi-bility of reaching consensus with a single faulty process [123]. Coordination using the Paxos algorithm is discussed in Section 4.5.

Distributed systems experience communication problems such as lost messages, messages out of sequence, or corrupted messages. There are solutions for handling these undesirable phenomena; for example, one can use virtual time, that is, sequence numbers, to ensure that messages are processed in an order consistent with the time they were sent by all processes involved, but this complicates the algorithms and increases the processing time.

Advisory locks are based on the assumption that all processes play by the rules. Advisory locks do not have any effect on processes that circumvent the locking mechanisms and access the shared objects directly. *Mandatory locks* block access to the locked objects to all processes that do not hold the locks, regardless of whether they use locking primitives.

Locks that can be held for only a very short time are called *fine-grained*, whereas *coarse-grained* locks are held for a longer time. Some operations require meta-information about a lock, such as the name of the lock, whether the lock is shared or held in exclusivity, and the generation number of the lock. This meta-information is sometimes aggregated into an opaque byte string called a *sequencer*.

The question of how to most effectively support a locking and consensus component of a large-scale distributed system demands several design decisions. A first design decision is whether the locks should be mandatory or advisory. *Mandatory locks* have the obvious advantage of enforcing access control; a traffic analogy is that a mandatory lock is like a drawbridge. Once it is up, all traffic is forced to stop.

An *advisory lock* is like a stop sign; those who obey the traffic laws will stop, but some might not. The disadvantages of mandatory locks are added overhead and less flexibility. Once a data item is locked, even a high-priority task related to maintenance or recovery cannot access the data unless it forces the application holding the lock to terminate. This is a very significant problem in large-scale systems where partial system failures are likely.

A second design decision is whether the system should be based on fine-grained or coarse-grained locking. *Fine-grained locks* allow more application threads to access shared data in any time interval, but they generate a larger workload for the lock server. Moreover, when the lock server fails for a period of time, a larger number of applications are affected. Advisory locks and *coarse-grained locks* seem to be a better choice for a system expected to scale to a very large number of nodes distributed in data centers that are interconnected via wide area networks.

A third design decision is how to support a systematic approach to locking. Two alternatives come to mind: (i) delegate to the clients the implementation of the consensus algorithm and provide a library of functions needed for this task, or (ii) create a locking service that implements a version of the asynchronous Paxos algorithm and provide a library to be linked with an application client to support service calls. Forcing application developers to invoke calls to a Paxos library is more cumbersome and more prone to errors than the service alternative. Of course, the lock service itself has to be scalable to support a potentially heavy load.

Another design consideration is flexibility, the ability of the system to support a variety of applica-tions. A name service comes immediately to mind because many cloud applications `read` and `write` small files. The names of small files can be included in the namespace of the service to allow atomic file operations. The choice should also consider the performance; a service can be optimized and clients can cache control information. Finally, we should consider the overhead and resources for reaching consensus. Again, the service seems to be more advantageous because it needs fewer replicas for high availability.

In the early 2000s, when Google started to develop a lock service called *Chubby* [61], it was decided to use advisory and coarse-grained locks. The service is used by several Google systems, including the GFS discussed in Section 8.5 and *BigTable* (see Section 8.9).

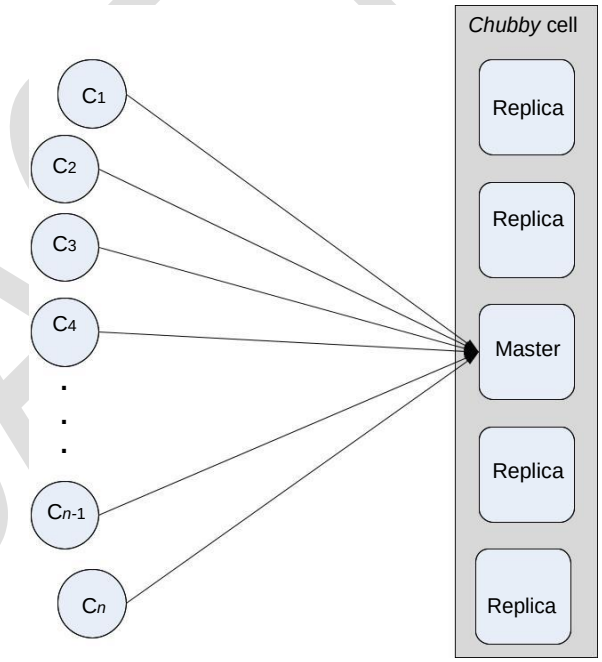


FIGURE 8.9

A *Chubby* cell consisting of five replicas, one of which is elected as a master; *n* clients use RPCs to communicate with the master.

The basic organization of the system is shown in Figure 8.9. A *Chubby* cell typically serves one data center. The cell server includes several *replicas*, the standard number of which is five. To reduce the probability of correlated failures, the servers hosting replicas are distributed across the campus of a data center.

The replicas use a distributed consensus protocol to elect a new *master* when the current one fails. The master is elected by a majority, as required by the asynchronous Paxos algorithm, accompanied by the commitment that a new master will not be elected for a period called a *master lease.* A *session* is a connection between a client and the cell server maintained over a period of time; the data cached by the client, the locks acquired, and the handles of all files locked by the client are valid for only the duration of the session.

Clients use RPCs to request services from the master. When it receives a `write` request, the master propagates the request to all replicas and waits for a reply from a majority of replicas before responding. When it receives a `read` request the master responds without consulting the replicas. The client interface of the system is similar to, yet simpler than, the one supported by the *Unix* File System. In addition, it includes notification of events related to file or system status. A client can subscribe to events such as file content modification, change or addition of a child node, master failure, lock acquired, conflicting lock requests, and invalid file handle.

The files and directories of the *Chubby* service are organized in a tree structure and use a naming scheme similar to that of *Unix*. Each file has a *file handle* similar to the file descriptor. The master of a cell periodically writes a snapshot of its database to a GFS file server.

Each file or directory can act as a lock. To `write` to a file the client must be the only one holding the file handle, whereas multiple clients may hold the file handle to `read` from the file. Handles are created by a call to *open ()* and destroyed by a call to *close ()*. Other calls supported by the service are *GetContentsAndStat ()*, to get the file data and meta-information, *SetContents*, and *Delete ()* and several calls allow the client to acquire and release locks. Some applications may decide to create and manipulate a sequencer with calls to *SetSequencer ()*, which associates a sequencer with a handle, *Get-Sequencer ()* to obtain the sequencer associated with a handle, or check the validity of a sequencer with *CheckSequencer ()*.

The sequence of calls *SetContents()*, *SetSequencer()*, *GetContentsAndStat()*, and *CheckSequencer()* can be used by an application for the election of a master as follows: all candidate threads attempt to open a lock file, call it *lfile*, in exclusive mode; the one that succeeds in acquiring the lock for *lfile* becomes the master, writes its identity in *lfile*, creates a sequencer for the lock of *lfile*, call it *lfseq*, and passes it to the server. The other threads read the *lfile* and discover that they are replicas; periodically they check the sequencer *lfseq* to determine whether the lock is still valid. The example illustrates the use of *Chubby* as a name server. In fact, this is one of the most frequent uses of the system.

We now take a closer look at the actual implementation of the service. As pointed out earlier, *Chubby* locks and *Chubby* files are stored in a database, and this database is replicated. The architecture of these replicas shows that the stack consists of the Chubby component, which implements the Chubby protocol for communication with the clients, and the active components, which `write` log entries and files to the local storage of the replica see (Figure 8.10).

Recall that an *atomicity log* for a transaction-processing system allows a crash recovery procedure to undo all-or-nothing actions that did not complete or to finish all-or-nothing actions that committed

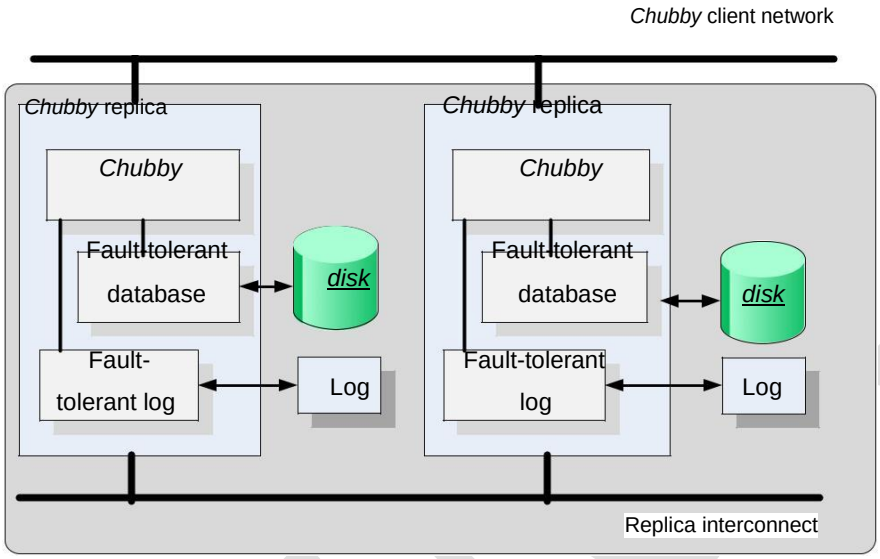


FIGURE 8.10

Chubby replica architecture. The *Chubby* component implements the communication protocol with the clients. The system includes a component to transfer files to a fault-tolerant database and a fault-tolerant log component to `write` log entries. The fault-tolerant log uses the Paxos protocol to achieve consensus. Each replica has its own local file system; replicas communicate with one another using a dedicated interconnect and communicate with clients through a client network.

but did not record all of their effects. Each replica maintains its own copy of the log; a new log entry is appended to the existing log and the Paxos algorithm is executed repeatedly to ensure that all replicas have the same sequence of log entries.

The next element of the stack is responsible for the maintenance of a fault-tolerant database – in other words, making sure that all local copies are consistent. The database consists of the actual data, or the *local snapshot* in *Chubby* speak, and a *replay log* to allow recovery in case of failure. The state of the system is also recorded in the database.

The Paxos algorithm is used to reach consensus on sets of values (e.g., the sequence of entries in a replicated log). To ensure that the Paxos algorithm succeeds in spite of the occasional failure of a replica, the following three phases of the algorithm are executed repeatedly:

1. Elect a replica to be the master/coordinator. When a master fails, several replicas may decide to assume the role of a master. To ensure that the result of the election is unique, each replica generates a sequence number larger than any sequence number it has seen, in the range $(1, r)$, where r is the number of replicas, and broadcasts it in a *propose* message. The replicas that have not seen a higher sequence number broadcast a *promise* reply and declare that they will reject proposals from other candidate masters. If the number of respondents represents a majority of replicas, the one that sent the *propose* message is elected master.

2. The master broadcasts to all replicas an *accept* message, including the value it has selected, and waits for replies, either *acknowledge* or *reject*.
3. Consensus is reached when the majority of the replicas send an *acknowledge* message; then the master broadcasts the *commit* message.

Implementation of the Paxos algorithm is far from trivial. Although the algorithm can be expressed in as few as ten lines of pseudocode, its actual implementation could be several thousand lines of C++ code [71]. Moreover, practical use of the algorithm cannot ignore the wide variety of failure modes, including algorithm errors and bugs in its implementation, and testing a software system of a few thousands lines of codes is challenging.

8.Transaction processing and *NoSQL* databases

Many cloud services are based on *online transaction processing* (OLTP) and operate under tight latency constraints. Moreover, these applications have to deal with extremely high data volumes and are expected to provide reliable services for very large communities of users. It did not take very long for companies heavily involved in cloud computing, such as Google and Amazon, e-commerce companies such as eBay, and social media networks such as Facebook, Twitter, or LinkedIn, to discover that traditional relational databases are not able to handle the massive amount of data and the real-time demands of online applications that are critical for their business models.

The search for alternate models with which to store the data on a cloud is motivated by the need to decrease the latency by caching frequently used data in memory on dedicated servers, rather than fetching it repeatedly. In addition, distributing the data on a large number of servers allows multiple transactions to occur at the same time and decreases the response time. The relational schema are of little use for such applications in which conversion to key-value databases seems a much better approach. Of course, such systems do not store meaningful metadata information, unless they use extensions that cannot be exported easily.

A major concern for the designers of OLTP systems is to reduce the response time. The term *memcaching* refers to a general-purpose distributed memory system that caches objects in main memory (RAM); the system is based on a very large hash table distributed across many servers. The *memcached* system is based on a client-server architecture and runs under several operating systems, including *Linux*, *Unix*, *Mac OS X*, and *Windows*. The servers maintain a key-value associative array. The API allows the clients to add entries to the array and to query it. A key can be up to 250 bytes long, and a value can be no larger than 1 MB. The *memcached* system uses an LRU cache-replacement strategy.

Scalability is the other major concern for cloud OLTP applications and implicitly for datastores. There is a distinction between *vertical scaling*, where the data and the workload are distributed to systems that share resources such as cores and processors, disks, and possibly RAM, and *horizontal scaling*, where the systems do not share either primary or secondary storage [66].

Cloud stores such as *document stores* and *NoSQL* databases are designed to scale well, do not exhibit a single point of failure, have built-in support for consensus-based decisions, and support partitioning and replication as basic primitives. Systems such as Amazon's *SimpleDB*, discussed in Section 3.1; *CouchDB* (see `http://couchdb.apache.org/`), or *Oracle NoSQL database* [277] are very popular, though they provide less functionality than traditional databases. The *key-value* data model

is very popular. Several such systems, including Voldemort, Redis, Scalaris, and Tokyo cabinet, are discussed in [66].

The "soft-state" approach in the design of *NoSQL* allows data to be inconsistent and transfers the task of implementing only the subset of the ACID properties required by a specific application to the application developer. The *NoSQL* systems ensure that data will be "eventually consistent" at some future point in time instead of enforcing consistency at the time when a transaction is "committed." Data partitioning among multiple storage servers and data replication are also tenets of the *NoSQL* philosophy[7]; they increase availability, reduce response time, and enhance scalability.

The name *NoSQL* given to this storage model is misleading. Michael Stonebreaker notes [335] that "blinding performance depends on removing overhead. Such overhead has nothing to do with SQL, but instead revolves around traditional implementations of ACID transactions, multi-threading, and disk management."

The overhead of OLTP systems is due to four sources with equal contribution: logging, locking, latching, and buffer management. Logging is expensive because traditional databases require transaction durability; thus, every `write` to the database can be completed only after the log has been updated. To guarantee atomicity, transactions lock every record, and this requires access to a lock table. Many operations require multithreading, and the access to shared data structures, such as lock tables, demands short-term latches[8] for coordination. The breakdown of the instruction count for these operations in existing DBMSs is as follows: 34.6% for buffer management, 14.2% for latching, 16.3% for locking, 11.9% for logging, and 16.2% for hand-coded optimization [157].

Today OLTP databases could exploit the vast amounts of resources of modern computing and com-munication systems to store the data in main memory rather than rely on disk-resident B-trees and heap files, locking-based concurrency control, and support for multithreading optimized for the computer technology of past decades [157]. Logless, single-threaded, and transaction-less databases could replace the traditional ones for some cloud applications.

Data replication is critical not only for system reliability and availability, but also for its performance. In an attempt to avoid catastrophic failures due to power blackouts, natural disasters, or other causes (see also Section 1.6), many companies have established multiple data centers located in different geographic regions. Thus, data replication must be done over a *wide area network* (WAN). This could be quite challenging, especially for log data, metadata, and system configuration information, due to increased probability of communication failure and larger communication delays. Several strategies are possible, some based on master/slave configurations, others based on homogeneous replica groups.

Master/slave replication can be asynchronous or synchronous. In the first case the master replicates write-ahead log entries to at least one slave, and each slave acknowledges appending the log record as soon as the operation is done. In the second case the master must wait for the acknowledgments from all slaves before proceeding. Homogeneous replica groups enjoy shorter latency and higher availabil-ity than master/slave configurations. Any member of the group can initiate mutations that propagate asynchronously.

---

[7] It was suggested in the literature that a *NoSQL* database could be associated with the BASE acronym constructed from its relevant properties: *basically available*, *soft state*, and *eventually consistent*. This is in contrast with traditional databases characterized by ACID properties: *atomicity*, *consistency*, *isolation*, and *durability*.

[8] A *latch* is a counter that triggers an event when it reaches zero. For example, a master thread initiates a counter with the number of worker threads and waits to be notified when all of them have finished.

In summary, the "one-size-fits-all" approach to traditional storage system design is replaced by a flexible one tailored to the specific requirements of the applications. Sometimes the data management ecosystem of a cloud computing environment integrates multiple databases; for example, Oracle integrates its *NoSQL* database with the *HDFS* discussed in Section 8.6, with the Oracle Database, and with the Oracle Exadata. Another approach, discussed in Section 8.10, is to partition the data and to guarantee full ACID semantics within a partition, while supporting eventual consistency among partitions.

9. *BigTable*

BigTable is a distributed storage system developed by Google to store massive amounts of data and to scale up to thousands of storage servers [73]. The system uses the Google File System discussed in Section 8.5 to store user data as well as system information. To guarantee atomic `read` and `write` operations, it uses the *Chubby* distributed lock service (see Section 8.7); the directories and the files in the namespace of *Chubby* are used as locks.

The system is based on a simple and flexible data model. It allows an application developer to exercise control over the data format and layout and reveals data locality information to the application clients. Any `read` or `write` row operation is atomic, even when it affects more than one column. The column keys identify *column families*, which are units of access control. The data in a column family is of the same type. Client applications written in C++ can add or delete values, search for a subset of data, and look up data in a row.

A row key is an arbitrary string of up to 64 KB, and a row range is partitioned into *tablets* serving as units for load balancing. The time stamps used to index various versions of the data in a cell are 64-bit integers; their interpretation can be defined by the application, whereas the default is the time of an event in microseconds. A column key consists of a string defining the family name, a set of printable characters, and an arbitrary string as qualifier.

The organization of a *BigTable* (see Figure 8.11) shows a sparse, distributed, multidimensional map for an email application. The system consists of three major components: a library linked to application clients to access the system, a master server, and a large number of tablet servers. The master server controls the entire system, assigns tablets to tablet servers and balances the load among them, manages garbage collection, and handles table and column family creation and deletion.

Internally, the space management is ensured by a three-level hierarchy: the *root tablet*, the location of which is stored in a *Chubby* file, points to entries in the second element, the *metadata* tablet, which, in turn, points to *user* tablets, collections of locations of users' tablets. An application client searches through this hierarchy to identify the location of its tablets and then caches the addresses for further use.

The performance of the system reported in [73] is summarized in Table 8.2. The table shows the number of random and sequential `read` and `write` and scan operations for 1**,** 000 bytes, when the number of servers increases from 1 to 50, then to 250, and finally to 500. Locking prevents the system from achieving a linear speed-up, but the performance of the system is still remarkable due to a fair number of optimizations. For example, the number of scans on 500 tablet servers is 7**,**843**/**2 × 10^3 instead of 15**,**385**/**2 × 10^3. It is reported that only 12 clusters use more than 500 tablet servers, whereas some 259 clusters use between 1 and 19 tablet servers.

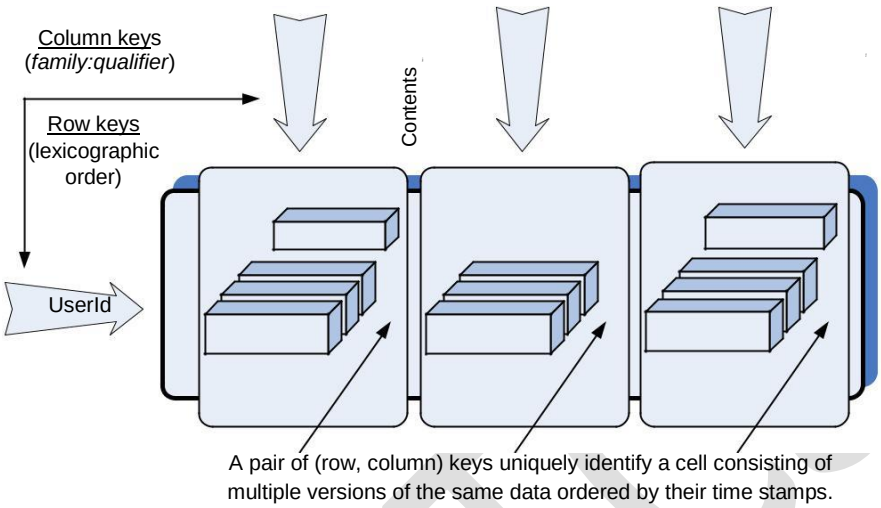


FIGURE 8.11

A *BigTable* example. The organization of an email application as a sparse, distributed, multidimensional map. The slice of the *BigTable* shown consists of a row with the key "UserId" and three family columns. The "Contents" key identifies the cell holding the contents of emails received, the cell with key "Subject" identifies the subject of emails, and the cell with the key "Reply" identifies the cell holding the replies. The versions of records in each cell are ordered according to their time stamps. The row keys of this *BigTable* are ordered lexicographically. A column key is obtained by concatenating the *family* and the *qualifier* fields. Each value is an uninterpreted array of bytes.

Table 8.2 *BigTable* performance: the number of operations per tablet server.

Number of Tablet Servers	Random Read	Sequential Read	Random Write	Sequential Write	Scan
1	1,212	4,425	8,850	8,547	15,385
50	593	2,463	3,745	3,623	10,526
250	479	2,625	3,425	2,451	9,524
500	241	2,469	2,000	1,905	7,843

BigTable is used by a variety of applications, including *Google Earth*, *Google Analytics*, *Google Finance*, and Web crawlers. For example, *Google Earth* uses two tables, one for preprocessing and one for serving client data. The preprocessing table stores raw images; the table is stored on disk because it contains some 70 TB of data. Each row of data consists of a single image; adjacent geographic segments are stored in rows in close proximity to one another. The column family is very sparse; it contains a column for every raw image. The preprocessing stage relies heavily on *MapReduce* to

clean and consolidate the data for the serving phase. The serving table stored on GFS is "only" 500 GB, and it is distributed across several hundred tablet servers, which maintain in-memory column families. This organization enables the serving phase of *Google Earth* to provide a fast response time to tens of thousands of queries per second.

Google Analytics provides aggregate statistics such as the number of visitors to a Web page per day. To use this service, Web servers embed a *JavaScript* code into their Web pages to record information every time a page is visited. The data is collected in a *raw-click BigTable* of some 200 TB, with a row for each end-user session. A *summary* table of some 20 TB contains predefined summaries for a Website.

10.Megastore

Megastore is scalable storage for online services. The system, distributed over several data centers, has a very large capacity, 1 PB in 2011, and it is highly available. *Megastore* is widely used internally at Google; it handles some 23 billion transactions daily: 3 billion `write` and 20 billion `read` transactions [37].

The basic design philosophy of the system is to partition the data into *entity groups* and replicate each partition independently in data centers located in different geographic areas. The system supports full ACID semantics within each partition and provides limited consistency guarantees across partitions (see Figure 8.12). *Megastore* supports only those traditional database features that allow the system to scale well and that do not drastically affect the response time.

Another distinctive feature of the system is the use of the Paxos consensus algorithm, discussed in Section 2.11, to replicate primary user data, metadata, and system configuration information across data centers and for locking. The version of the Paxos algorithm used by *Megastore* does not require a single master. Instead, any node can initiate `read` and `write` operations to a write-ahead log replicated to a group of symmetric peers.

The entity groups are application-specific and store together logically related data. For example, an email account could be an entity group for an email application. Data should be carefully partitioned to avoid excessive communication between entity groups. Sometimes it is desirable to form multiple entity groups, as in the case of blogs [37].

The middle ground between traditional and *NoSQL* databases taken by the *Megastore* designers is also reflected in the data model. The data model is declared in a *schema* consisting of a set of *tables* composed of *entries*, each entry being a collection of named and typed *properties*. The unique primary key of an entity in a table is created as a composition of entry properties. A *Megastore* table can be a *root* or a *child* table. Each *child entity* must reference a special entity, called a *root entity* in its root table. An entity group consists of the primary entity and all entities that reference it.

The system makes extensive use of *BigTable*. Entities from different *Megastore* tables can be mapped to the same *BigTable* row without collisions. This is possible because the *BigTable* column name is a concatenation of the *Megastore* table name and the name of a property. A *BigTable* row for the root entity stores the transaction and all metadata for the entity group. As we saw in Section 8.9, multiple versions of the data with different time stamps can be stored in a cell. *Megastore* takes advantage of this feature to implement *multi-version concurrency control* (MVCC); when a mutation of a transaction occurs, this mutation is recorded along with its time stamp, rather than marking the old data as obsolete

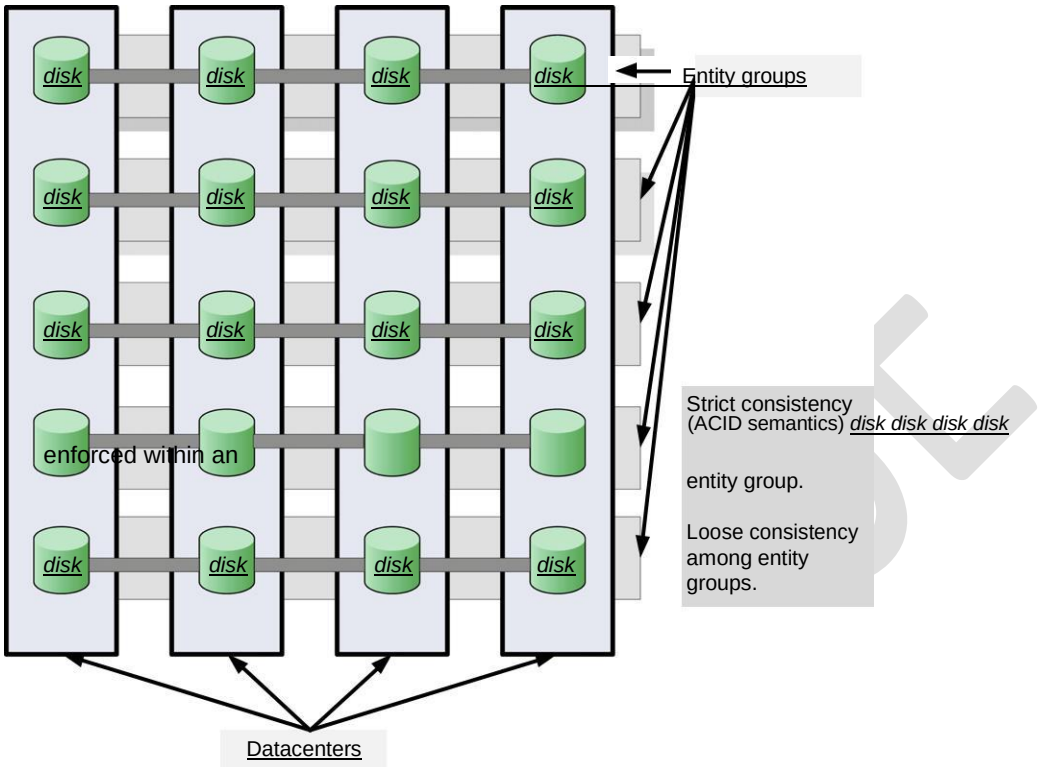


FIGURE 8.12

Megastore organization. The data is partitioned into *entity groups*; full ACID semantics within each partition and limited consistency guarantees across partitions are supported. A partition is replicated across data centers in different geographic areas.

and adding the new version. This strategy has several advantages: `read` and `write` operations can proceed concurrently, and a `read` always returns the last fully updated version.

A `write` transaction involves the following steps: (1) Get the timestamp and the log position of the last committed transaction. (2) Gather the `write` operations in a log entry. (3) Use the consensus algorithm to `append` the log entry and then commit. (4) Update the *BigTable* entries. (5) Clean up.