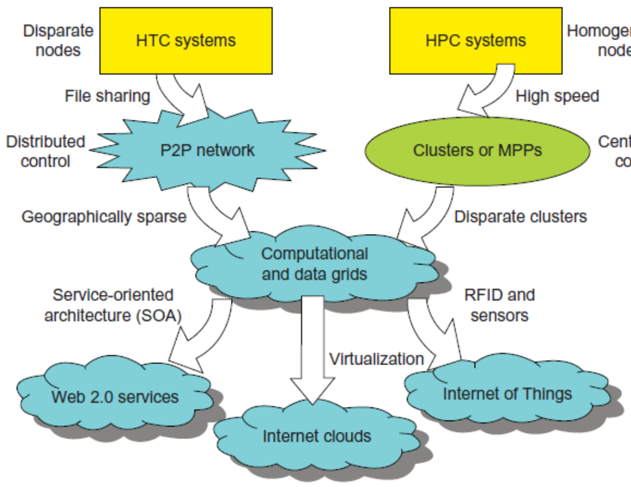
Subject : Cloud Computing (16MCA51)
Branch : MCA
Semester : V Sem
Faculty Name : Ms. Moumita Roy

1 Explain with a diagram the evaluation of computing platform.



Computer technology has gone through five generations of development, with each generation lasting from 10 to 20 years. Successive generations are overlapped in about 10 years. For instance, from 1950 to 1970, a handful of mainframes, including the IBM 360 and CDC 6400, were built to satisfy the demands of large businesses and government organizations. From 1960 to 1980, lower-cost mini-computers such as the DEC PDP 11 and VAX Series became popular among small businesses and on college campuses.

From 1970 to 1990, we saw widespread use of personal computers built with VLSI microprocessors. From 1980 to 2000, massive numbers of portable computers and pervasive devices appeared in both wired and wireless applications. Since 1990, the use of both HPC and HTC systems hidden in clusters, grids, or Internet clouds has proliferated. These systems are employed by both consumers and high-end web-scale computing and information services.

The general computing trend is to leverage shared web resources and massive amounts of data over the Internet. The figure above illustrates the evolution of HPC and HTC systems. On the HPC side, supercomputers (massively parallel processors or MPPs) are gradually replaced by clusters of cooperative computers out of a desire to share computing resources. The cluster is often a collection of homogeneous compute nodes that are physically connected in close range to one another.

On the HTC side, peer-to-peer (P2P) networks are formed for distributed file sharing and content delivery applications. A P2P system is built over many client machines. Peer machines are globally distributed in nature. P2P, cloud computing, and web service platforms are more focused on HTC applications than on HPC applications. Clustering and P2P technologies lead to the development of computational grids or data grids.

2. Distinguish the different computing paradigms.

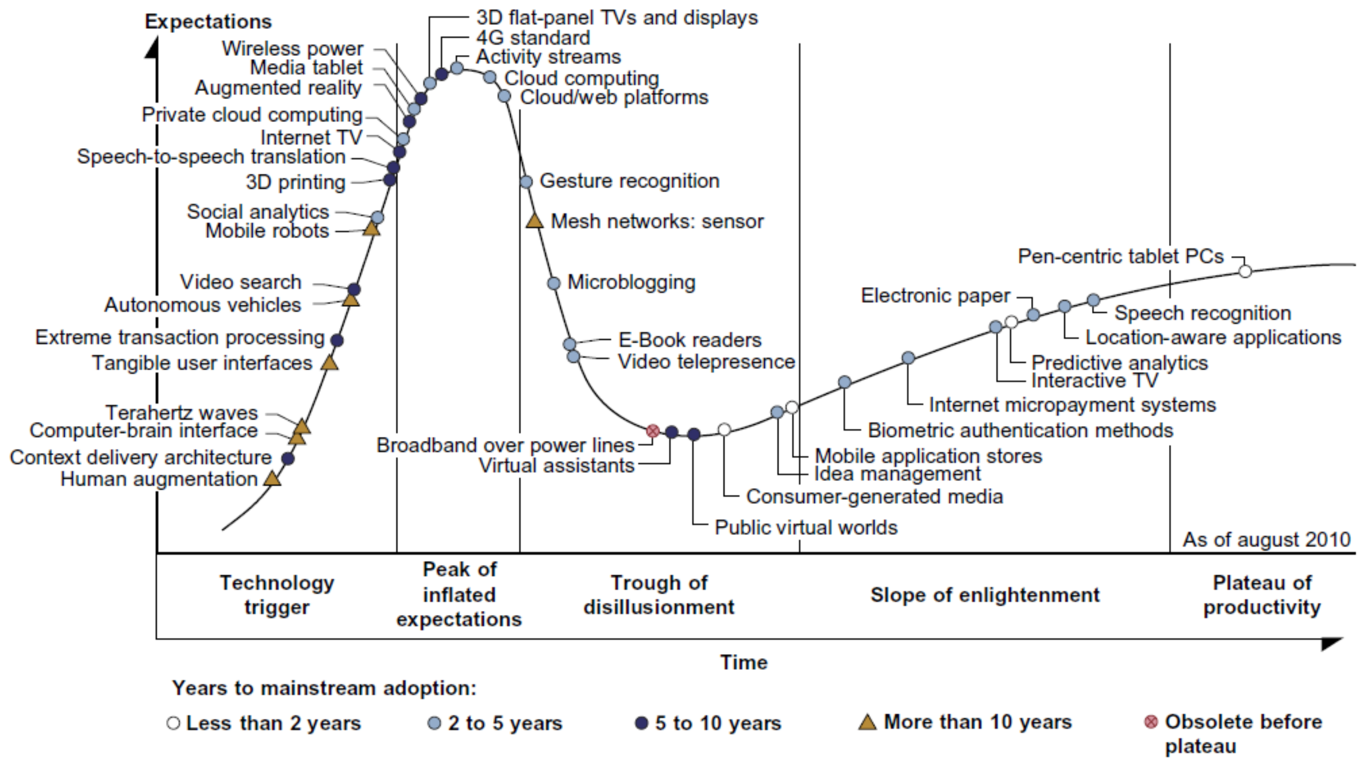
Centralized computing This is a computing paradigm by which all computer resources are centralized in one physical system. All resources (processors, memory, and storage) are fully shared and tightly coupled within one integrated OS. Many data centers and supercomputers are centralized systems, but they 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.

3. Discuss the hype-cycle of new technologies. Assist the answer with a diagram.



Any new and emerging computing and information technology may go through a hype cycle, as illustrated in the figure above. This cycle shows the expectations for the technology at five different stages. The expectations rise sharply from the trigger period to a high peak of inflated expectations. Through a short period of disillusionment, the expectation may drop to a valley and then increase steadily over a long enlightenment period to a plateau of productivity. The number of years for an emerging technology to reach a certain stage is marked by special symbols. The hollow circles indicate technologies that will reach mainstream adoption in two years. The gray circles represent technologies that will reach mainstream adoption in two to five years. The solid circles represent those that require five to 10 years to reach mainstream adoption, and the triangles denote those that require more than 10 years. The crossed circles represent technologies that will become obsolete before they reach the plateau.

The hype cycle in the figure shows the technology status as of August 2010. For example, at that time consumer-generated media was at the disillusionment stage, and it was predicted to take less than two years to reach its plateau of adoption. Internet micropayment systems were forecast to take two to five years to move from the enlightenment stage to maturity. It was believed that 3D printing would take five to 10 years to move from the rising expectation stage to mainstream adoption, and mesh network sensors were expected to take more than 10 years to move from the inflated expectation stage to a plateau of mainstream adoption.

Also as shown in figure, the cloud technology had just crossed the peak of the expectation stage in 2010, and it was expected to take two to five more years to reach the productivity stage. However, broadband over power line technology was expected to become obsolete before leaving the valley of disillusionment stage in 2010. Many additional technologies (denoted by dark circles) were at their peak expectation stage in August 2010, and they were expected to take five to 10 years to reach their plateau of success. Once a technology begins to climb the slope of enlightenment, it may reach the productivity plateau within two to five years. Among these promising technologies are the clouds, biometric authentication, interactive TV, speech recognition, predictive analytics, and media tablets.

4. Explain the concepts of IoT and CPS.

Internet of Things

The traditional Internet connects machines to machines or web pages to web pages. The concept of the IoT was introduced in 1999 at MIT. 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.

With the introduction of the IPv6 protocol, 2^{128} IP addresses are available to distinguish all the objects on Earth, including all computers and pervasive devices. The IoT researchers have estimated that every human being will be surrounded by 1,000 to 5,000 objects. The IoT needs to be designed to track 100 trillion static or moving objects simultaneously. The IoT demands universal addressability of all of the objects or things. To reduce the complexity of identification, search, and storage, one can set the threshold to filter out fine-grain objects. The IoT obviously extends the Internet and is more heavily developed in Asia and European countries.

In the IoT era, all objects and devices are instrumented, interconnected, and interacted with each other intelligently. This communication can be made between people and things or among the things themselves. Three communication patterns co-exist: namely H2H (human-to-human), H2T (human-to-thing), and T2T (thing-to-thing). Here things include machines such as PCs and mobile phones. The idea here is to connect things (including human and machine objects) at any time and any place intelligently with low cost. Any place connections include at the PC, indoor (away from PC), outdoors, and on the move. Any time connections include daytime, night, outdoors and indoors, and on the move as well.

The dynamic connections will grow exponentially into a new dynamic network of networks, called the Internet of Things (IoT). The IoT is still in its infancy stage of development. Many proto-type IoTs with restricted areas of coverage are under experimentation at the time of this writing. Cloud computing researchers expect to use the cloud and future Internet technologies to support fast, efficient, and intelligent interactions among humans, machines, and any objects on Earth. A smart Earth should have intelligent cities, clean water, efficient power, convenient transportation, good food supplies, responsible banks, fast telecommunications, green IT, better schools, good health care, abundant resources, and so on. This dream living environment may take some time to reach fruition at different parts of the world.

Cyber Physical Systems

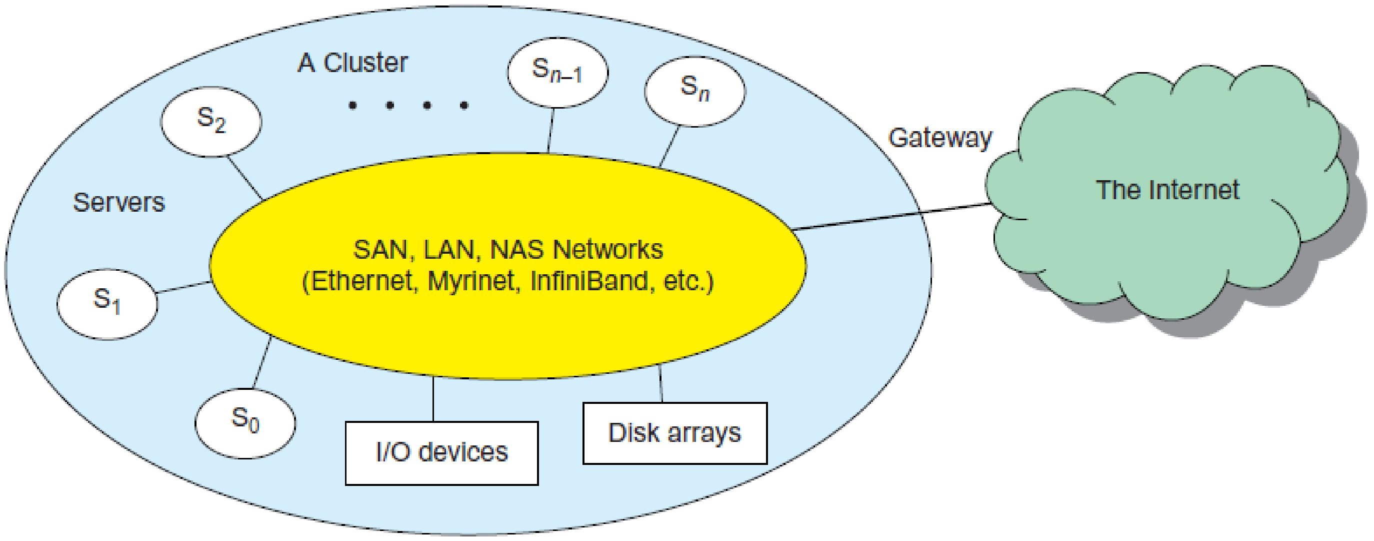
A cyber-physical system (CPS) is the result of interaction between computational processes and the physical world. A CPS integrates "cyber" (heterogeneous, asynchronous) with "physical" (concur-rent and information-dense) objects. A CPS merges the "3C" technologies of computation, communication, and control into an intelligent closed feedback system between the physical world and the information world, a concept which is actively explored in the United States. The IoT emphasizes various networking connections among physical objects, while the CPS emphasizes exploration of virtual reality (VR) applications in the physical world. We may transform how we interact with the physical world just like the Internet transformed how we interact with the virtual world.

5 Classify parallel and distributed systems.

Functionality, Applications	Computer Clusters	Peer-to-Peer Networks	Data/ Computational Grids	Cloud Platforms
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	Gnutella, eMule, BitTorrent, Napster, KaZaA, Skype, JXTA	TeraGrid, GriPhyN, UK EGEE, D-Grid, ChinaGrid, etc.	Google App Engine, IBM Bluecloud, AWS, and Microsoft

XT4, etc.	Azure

6. Explain Cluster Architecture and SSI.



The figure above 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 has indicated that an ideal cluster should merge multiple system images into single-system image (SSI). Cluster designers desire a cluster operating system or some middle-ware to support SSI at various levels, including the sharing of CPUs, memory, and I/O across all cluster nodes. An SSI is an illusion created by software or hardware that presents a collection of resources as one integrated, powerful resource. SSI makes the cluster appear like a single machine to the user. A cluster with multiple system images is nothing but a collection of independent computers.

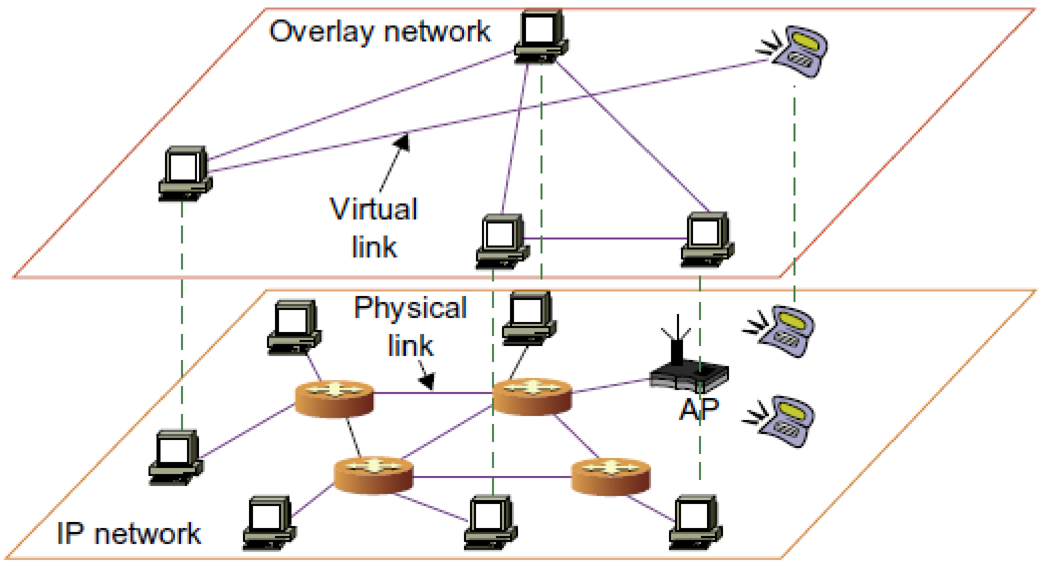
7. What is the hardware, software and middleware support needed to form a cluster?

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. For example, distributed memory has multiple images. Users may want all distributed memory to be shared by all servers by forming distributed shared memory (DSM). Many SSI features are expensive or difficult to achieve at various cluster operational levels. Instead of achieving SSI, many clusters are loosely coupled machines. Using virtualization, one can build many virtual clusters dynamically, upon user demand.

8. Discuss the characteristics of a P2P system.

In a P2P system, every node acts as both a client and a server, providing part of the system resources. Peer machines are simply client computers connected to the Internet. All client machines act autonomously to join or leave the system freely. This implies that no master-slave relationship exists among the peers. No central coordination or central database is needed. In other words, no peer machine has a global view of the entire P2P system. The system is self-organizing with distributed control.

The figure below shows the architecture of a P2P network at two abstraction levels. Initially, the peers are totally unrelated. Each peer machine joins or leaves the P2P network voluntarily. Only the participating peers form the physical network at any time. Unlike the cluster or grid, a P2P network does not use a dedicated interconnection network. The physical network is simply an ad hoc network formed at various Internet domains randomly using the TCP/IP and NAI protocols. Thus, the physical network varies in size and topology dynamically due to the free membership in the P2P network.



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. This overlay is a virtual network formed by mapping each physical machine with its ID, logically, through a virtual mapping as shown in the figure. When a new peer joins the system, its peer ID is added as a node in the overlay network. When an existing peer leaves the system, its peer ID is removed from the overlay network automatically. Therefore, it is the P2P overlay network that characterizes the logical connectivity among the peers.

There are two types of overlay networks: unstructured and structured. An unstructured overlay network is characterized by a random graph. There is no fixed route to send messages or files among the nodes. Often, flooding is applied to send a query to all nodes in an unstructured overlay, thus resulting in heavy network traffic and nondeterministic search results. Structured overlay net-works follow certain connectivity topology and rules for inserting and removing nodes (peer IDs) from the overlay graph. Routing mechanisms are developed to take advantage of the structured overlays.

9. What are the major categories of P2P Network Family?

System Features	Distributed File Sharing	Collaborative Platform	Distributed P2P Computing	P2P Platform
Attractive Applications	Content distribution of MP3 music, video, open software, etc.	Instant messaging, collaborative design and gaming	Scientific exploration and social networking	Open networks for public resources
Operational Problems	Loose security and serious online copyright violations	Lack of trust, disturbed by spam, privacy, and peer collusion	Security holes, selfish partners, and peer collusion	Lack of standards or protection protocols
Example Systems	Gnutella, Napster, eMule, BitTorrent, Aimster, KaZaA, etc.	ICQ, AIM, Groove, Magi, Multiplayer Games, Skype, etc.	SETI@home, Geonome@home, etc.	JXTA, .NET, FightingAid@home, etc.

10. What are the various performance metrics used for a distributed system? Explain each of them.

In a distributed system, performance is attributed to a large number of factors. System throughput is often measured in **MIPS** (Discuss MIPS as explained in class), **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 sys-tem 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.

11. Discuss 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 − α) of the code can be compiled for parallel execution by n processors. The total execution time of the program is calculated by T + (1 − α)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:

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

The maximum speed up of n is achieved only if the sequential bottleneck α is reduced to zero or the code is fully parallelizable with α = 0. As the cluster becomes sufficiently large, that is, n → ∞, S approaches 1/α, an upper bound on the speedup S. Surprisingly, this upper bound is independent of the cluster size n. The sequential bottleneck is the portion of the code that cannot be parallelized. For example, the maximum speedup achieved is 4, if α = 0.25 or 1 − α = 0.75, even if one uses hundreds of processors. Amdahl's law teaches us that we should make the sequential bottle-neck as small as possible. Increasing the cluster size alone may not result in a good speed up in this case.

b. What is the problem with fixed workload that Gustafson's Law overcomes?

Problem with Fixed Workload
In Amdahl's law, we have assumed the same amount of workload for both sequential and parallel execution of the program with a fixed problem size or data set. This was called fixed-workload speedup by Hwang and Xu. To execute a fixed workload on n processors, parallel processing may lead to a system efficiency defined as follows:

$$E = S/n = 1/[\alpha n + 1 - \alpha]$$

Very often the system efficiency is rather low, especially when the cluster size is very large. To execute the aforementioned program on a cluster with n = 256 nodes, extremely low efficiency E = 1/[0.25 × 256 + 0.75] = 1.5% is observed. This is because only a few processors (say, 4) are kept busy, while the majority of the nodes are left idling.

Gustafson's Law
To achieve higher efficiency when using a large cluster, we must consider scaling the problem size to match the cluster capability. This leads to the following speedup law proposed by John Gustafson (1988), referred as scaled-workload speedup in [14]. Let W be the workload in a given program. When using an n-processor system, the user scales the workload to W′ = αW + (1 − α)nW. Note that only the parallelizable portion of the workload is scaled n times in the second term. This scaled workload W′ is essentially the sequential execution time on a single processor. The parallel execution time of a scaled workload W′ on n processors is defined by a scaled-workload speedup as follows:

$$S' = W'/W = [\alpha W + (1-\alpha)nW]/W = \alpha + (1-\alpha)n$$

This speedup is known as Gustafson's law. By fixing the parallel execution time at level W, the following efficiency expression is obtained:

$$E' = S'/n = \alpha/n + (1-\alpha)$$

For the preceding program with a scaled workload, we can improve the efficiency of using a 256-node cluster to E′ = 0.25/256 + 0.75 = 0.751. One should apply Amdahl's law and Gustafson's law under different workload conditions. For a fixed workload, users should apply Amdahl's law. To solve scaled problems, users should apply Gustafson's law.

11. 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.

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.

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.

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.

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 a sprocess-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. 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.

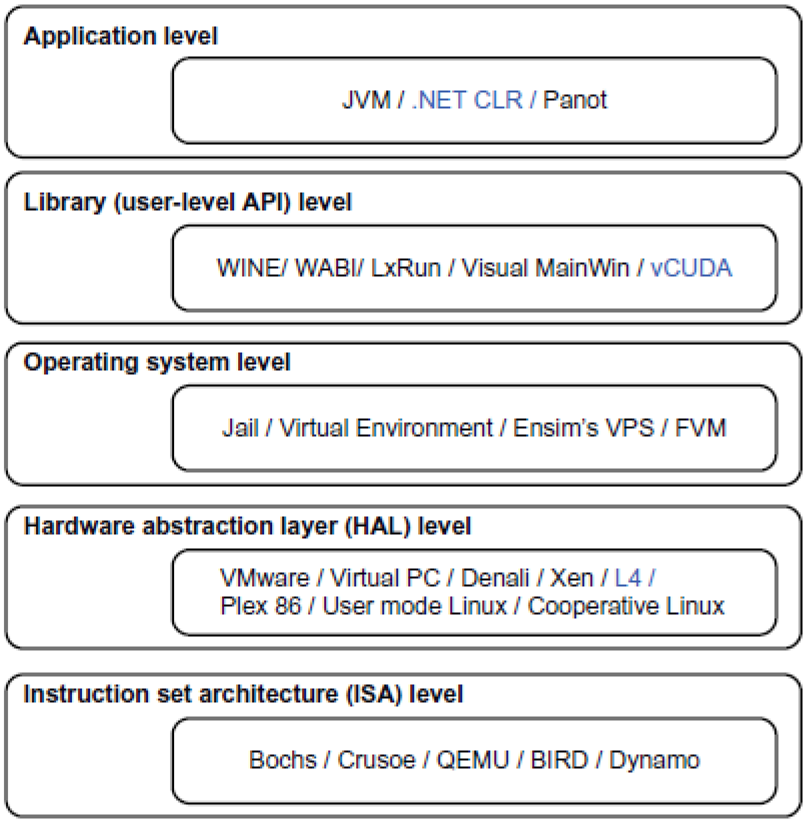

Application level
JVM / .NET CLR / Panot
Library (user-level API) level
WINE/ WABI/ LxRun / Visual MainWin / vCUDA
Operating system level
Jail / Virtual Environment / Ensim's VPS / FVM
Hardware abstraction layer (HAL) level
VMware / Virtual PC / Denali / Xen / L4 /
Plex 86 / User mode Linux / Cooperative Linux
Instruction set architecture (ISA) level
Bochs / Crusoe / QEMU / BIRD / Dynamo

FIGURE 3.2

Virtualization ranging from hardware to applications in five abstraction levels.

12.
Advantage:
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 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.

Disadvantage:
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. 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.

13

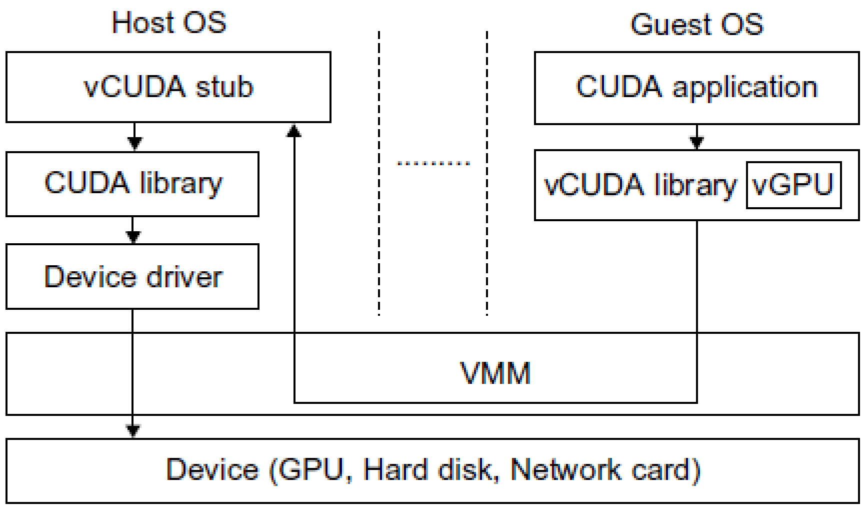


FIGURE 3.4

Basic concept of the vCUDA architecture.

.

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. 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 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.

14.

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

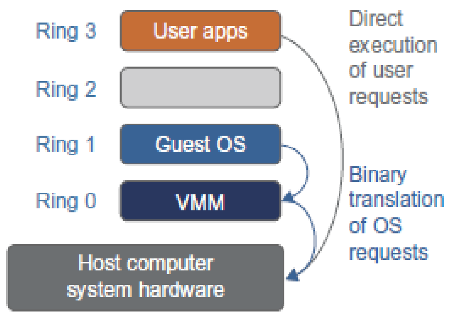

Ring 3
User apps
Ring 2
Ring 1
Guest OS
Ring 0
VMM
Host computer system hardware
Direct execution of user requests
Binary translation of OS requests

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.

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 the figure below, 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. 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.

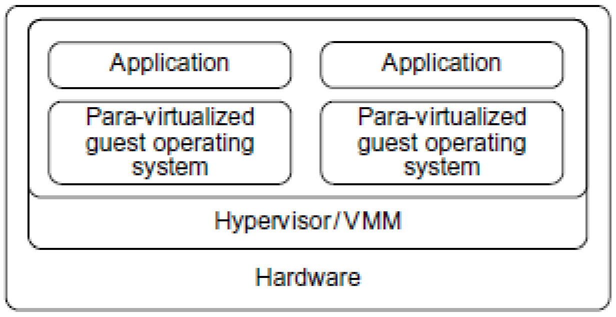

Application
Application
Para-virtualized guest operating system
Para-virtualized guest operating system
Hypervisor/VMM
Hardware

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.)

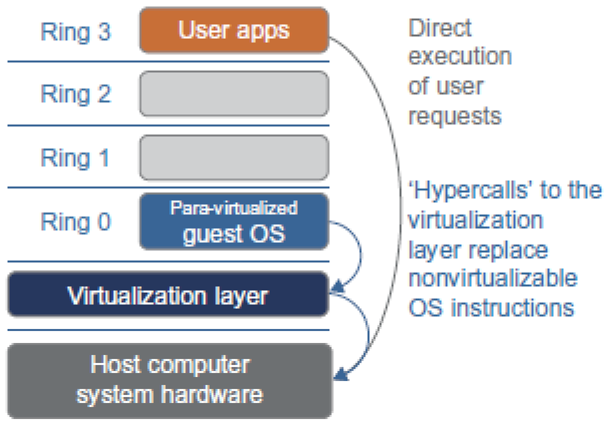

Ring 3
User apps
Ring 2
Ring 1
Ring 0
Para-virtualized guest OS
Virtualization layer
Host computer system hardware
Direct execution of user requests
'Hypercalls' to the virtualization layer replace nonvirtualizable OS instructions

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]*)

15.

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

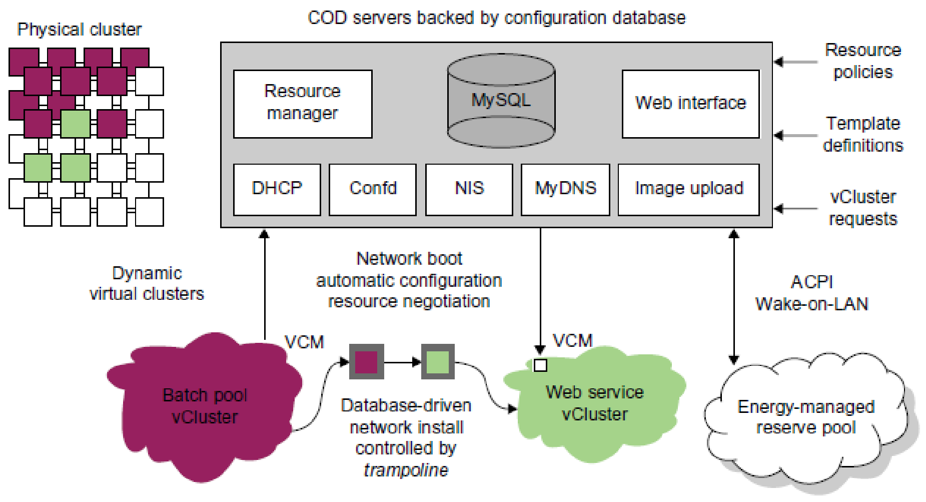


FIGURE 3.23

COD partitioning a physical cluster into multiple virtual clusters.

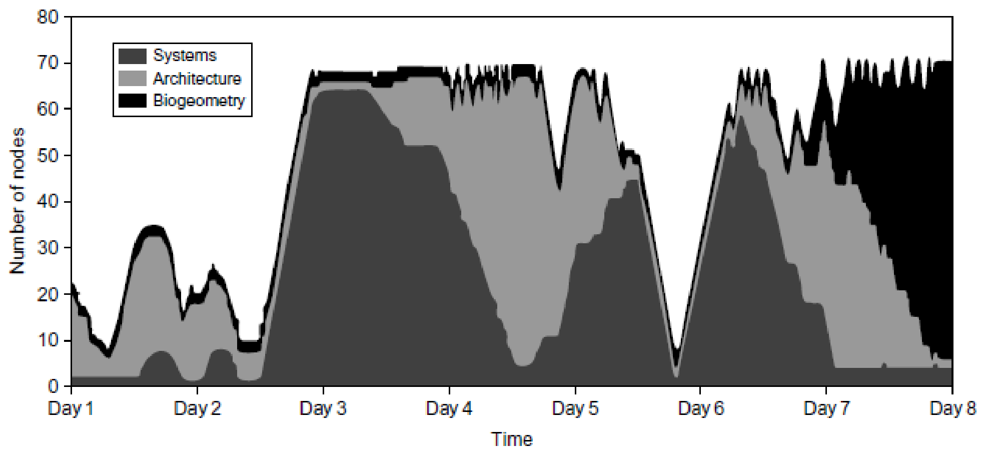


FIGURE 3.24

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

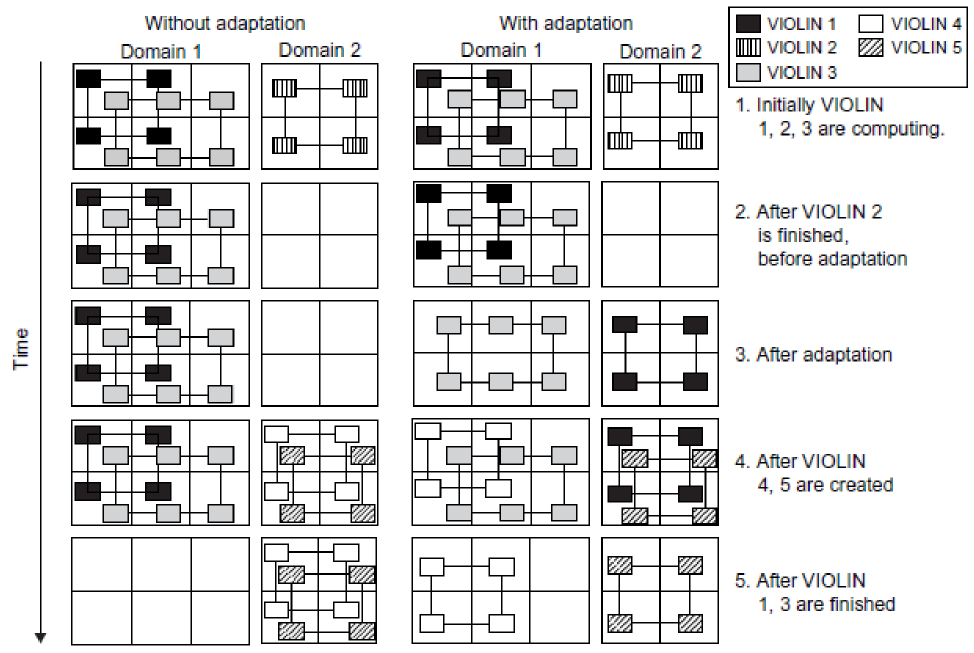

Without adaptation
With adaptation
Domain 1
Domain 2
Domain 1
Domain 2
VIOLIN 1
VIOLIN 2
VIOLIN 3
VIOLIN 4
VIOLIN 5
Time
1. Initially VIOLIN 1, 2, 3 are computing.
2. After VIOLIN 2 is finished, before adaptation
3. After adaptation
4. After VIOLIN 4, 5 are created
5. After VIOLIN 1, 3 are finished

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.

16.

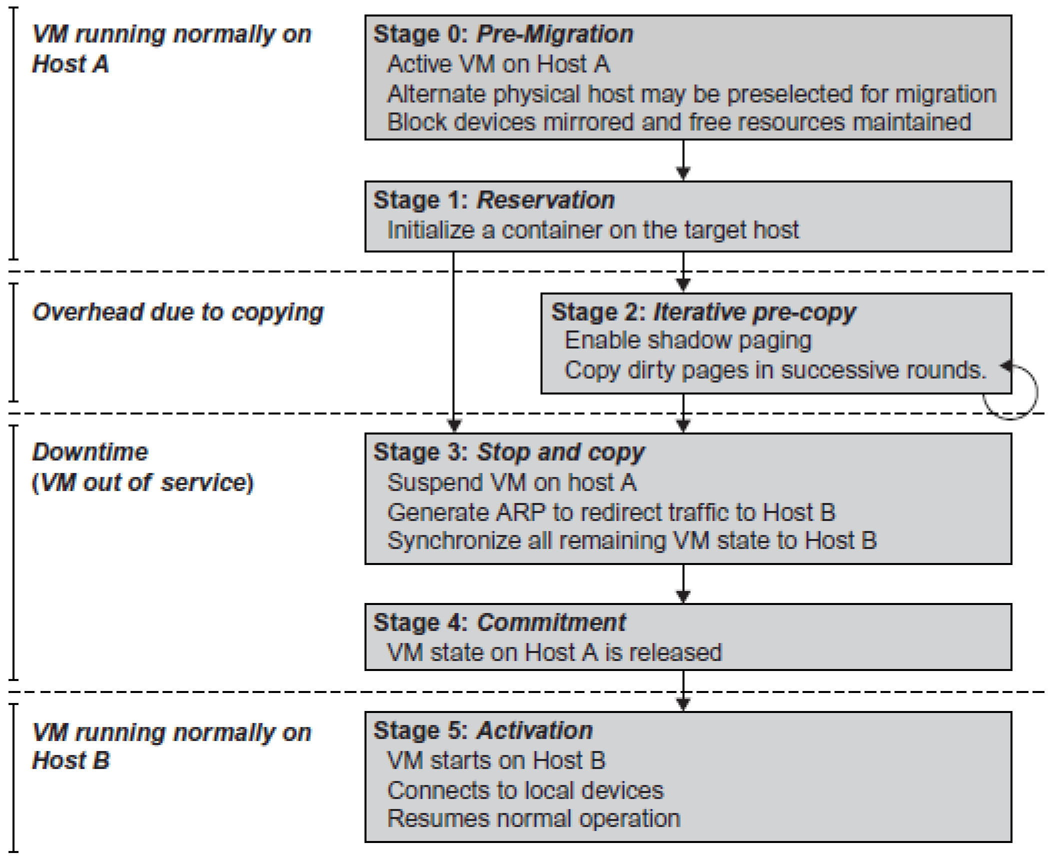

VM running normally on Host A
Stage 0: Pre-Migration
Active VM on Host A
Alternate physical host may be preselected for migration
Block devices mirrored and free resources maintained
Stage 1: Reservation
Initialize a container on the target host
Overhead due to copying
Stage 2: Iterative pre-copy
Enable shadow paging
Copy dirty pages in successive rounds.
Downtime (VM out of service)
Stage 3: Stop and copy
Suspend VM on host A
Generate ARP to redirect traffic to Host B
Synchronize all remaining VM state to Host B
Stage 4: Commitment
VM state on Host A is released
VM running normally on Host B
Stage 5: Activation
VM starts on Host B
Connects to local devices
Resumes normal operation

FIGURE 3.20

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

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.

17. 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. 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.
• 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 the Figure refer to different virtual clusters. In a virtual cluster system, it is quite important to store the large number of VM images efficiently. 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 the Figure. Four virtual clusters are created on the right, over the physical clusters. The physical machines are also called host systems. In contrast, the VMs are guest systems. The host and guest systems may run with different operating systems. Each VM can be installed on a remote server or replicated on multiple servers belonging to the same or different physical clusters. The boundary of a virtual cluster can change as VM nodes are added, removed, or migrated dynamically over time.

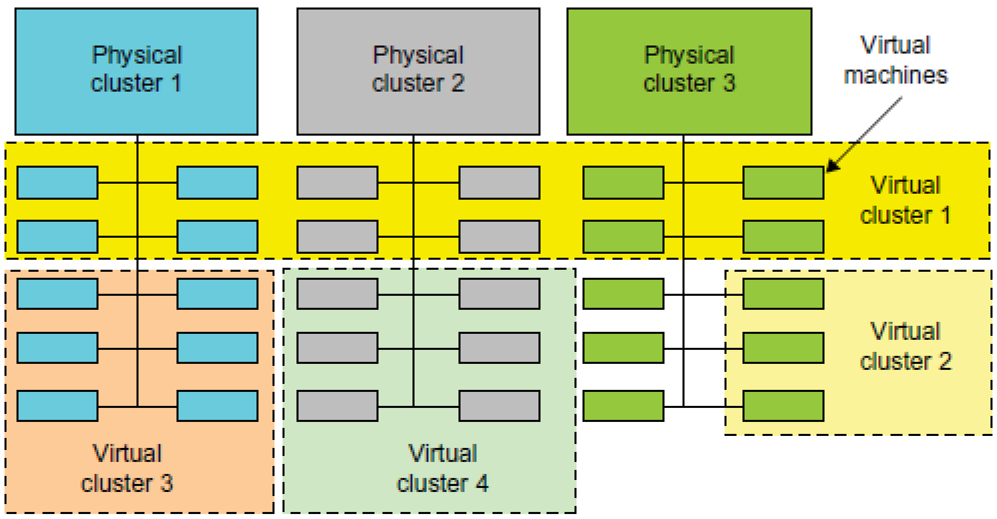
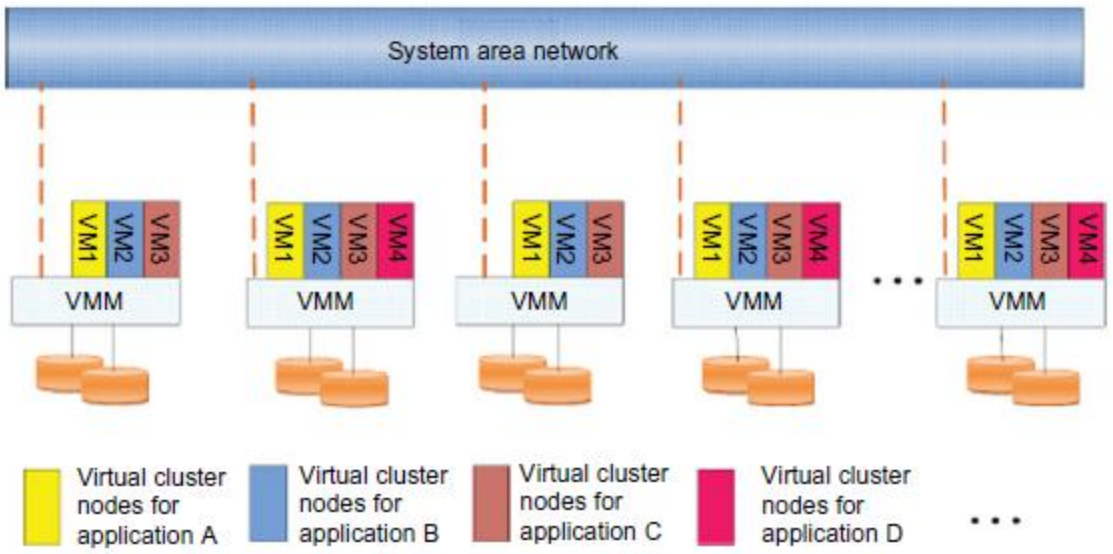


FIGURE 3.18

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

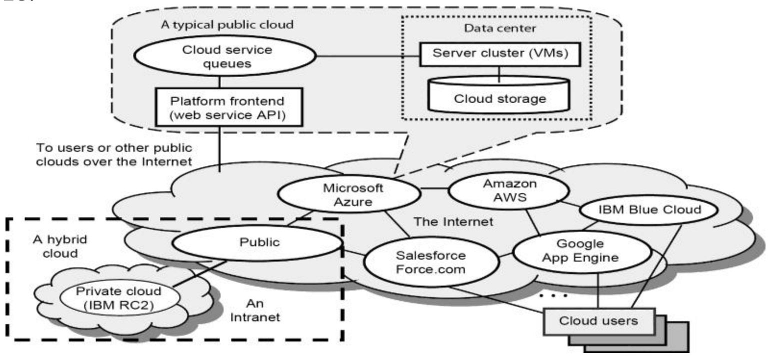


System area network
VM1
VM2
VM3
VM4
VMM
Virtual cluster nodes for application A
Virtual cluster nodes for application B
Virtual cluster nodes for application C
Virtual cluster nodes for application D

FIGURE 3.19

The concept of a virtual cluster based on application partitioning.

18.



What is a public cloud?

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 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.

Advantages of public clouds:

- Lower costs—no need to purchase hardware or software and you pay only for the service you use.
- No maintenance—your service provider provides the maintenance.
- Near-unlimited scalability—on-demand resources are available to meet your business needs.
- High reliability—a vast network of servers ensures against failure.

What is a private cloud?

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.

Advantages of private clouds:

- More flexibility—your organisation can customise its cloud environment to meet specific business needs.
- Improved security—resources are not shared with others, so higher levels of control and security are possible.
- High scalability—private clouds still afford the scalability and efficiency of a public cloud.

What is a hybrid cloud?

A hybrid cloud is built with both public and private clouds, as shown at the lower-left corner of Figure. 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.

Advantages of hybrid clouds:

- Control—your organisation can maintain a private infrastructure for sensitive assets.
- Flexibility—you can take advantage of additional resources in the public cloud when you need them.
- Cost-effectiveness—with the ability to scale to the public cloud, you pay for extra computing power only when needed.
- Ease—transitioning to the cloud does not have to be overwhelming because you can migrate gradually—phasing in workloads over time.

19. IaaS: Infrastructure as a Service

Cloud infrastructure services, known as Infrastructure as a Service (IaaS), are made of highly scalable and automated compute resources. IaaS is fully self-service for accessing and monitoring things like compute, networking, storage, and other services, and it allows businesses to purchase resources on-demand and as-needed instead of having to buy hardware outright.
Some characteristics to look for when considering IaaS are:

- Resources are available as a service
- The cost varies depending on consumption
- Services are highly scalable
- Typically includes multiple users on a single piece of hardware

- Provides complete control of the infrastructure to organizations
- Dynamic and flexible

PaaS: Platform as a Service

Cloud platform services, or Platform as a Service (PaaS), provide cloud components to certain software while being used mainly for applications. PaaS provides a framework for developers that they can build upon and use to create customized applications. All servers, storage, and networking can be managed by the enterprise or a third-party provider while the developers can maintain management of the applications.

PaaS has many characteristics that define it as a cloud service, including:

- It is built on virtualization technology, meaning resources can easily be scaled up or down as your business changes
- Provides a variety of services to assist with the development, testing, and deployment of apps
- Numerous users can access the same development application
- Web services and databases are integrated

SaaS: Software as a Service

Software as a Service, also known as cloud application services, represent the most commonly utilized option for businesses in the cloud market. SaaS utilizes the internet to deliver applications to its users, which are managed by a third-party vendor. A majority of SaaS applications are run directly through the web browser, and do not require any downloads or installations on the client side.

There are a few ways to help you determine when SaaS is being utilized:

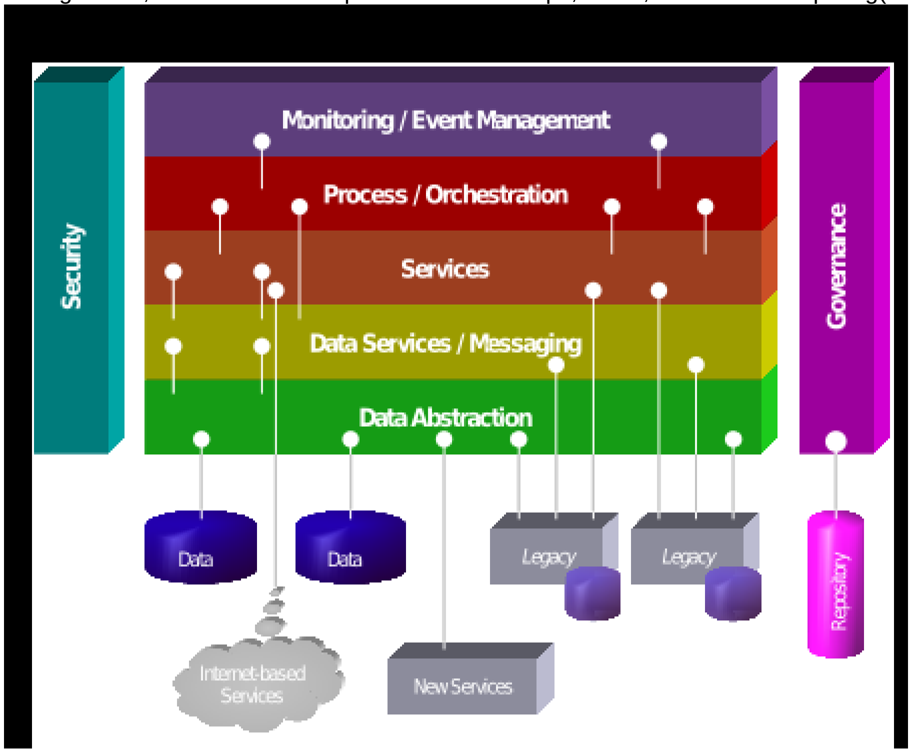
- Managed from a central location
- Hosted on a remote server
- Accessible over the internet
- Users not responsible for hardware or software updates

20. Summarize on Service-oriented Architecture with a neat diagram
A **service-oriented architecture** (**SOA**) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network. The basic principles of service-oriented architecture are independent of vendors, products and technologies.[1] A service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently, such as retrieving a credit card statement online.
A service has four properties according to one of many definitions of SOA
1. It logically represents a business activity with a specified outcome.
2. It is self-contained.
3. It is a black box for its consumers.
4. It may consist of other underlying services.

Different services can be used in conjunction to provide the functionality of a large software application, a principle it shares with modular programming. Service-oriented architecture integrates distributed, separately-maintained and deployed software components. It is enabled by technologies and standards that make it easier for components to communicate and cooperate over a network, especially an IP network.
In SOA, services use protocols that describe how they pass and parse messages using description metadata. This metadata describes both the functional characteristics of the service and quality-of-service characteristics. Service-oriented architecture aims to allow users to combine large chunks of functionality to form applications which are built purely from existing services and combining them in an ad hoc manner. A service presents a simple interface to the requester that abstracts away the underlying complexity acting as a black box. Further users can also access these independent services without any knowledge of their internal implementation.
The related buzzword service-orientation promotes *loose coupling* between services. SOA separates functions into distinct units, or services, which developers make accessible over a network in order to allow users to combine and reuse them in the production of applications. These services and their corresponding consumers communicate with each other by passing data in a well-defined, shared format, or by coordinating an activity between two or more services.[7]
A manifesto was published for service-oriented architecture in October, 2009. This came up with six core values which are listed as follows

1. **Business value** is given more importance than technical strategy.
2. **Strategic goals** are given more importance than project-specific benefits.
3. **Intrinsic inter-operability** is given more importance than custom integration.
4. **Shared services** are given more importance than specific-purpose implementations.
5. **Flexibility** is given more importance than optimization.
6. **Evolutionary refinement** is given more importance than pursuit of initial perfection.

SOA can be seen as part of the continuum which ranges from the older concept of distributed computing and modular programming, through SOA, and on to current practices of mashups, SaaS, and cloud computing(which some see as the offspring of SOA).



21. Define Message Passing Interface and Map Reduce

Message passing interface (MPI)
The message passing interface (MPI) is a standardized means of exchanging messages between multiple computers running a parallel program across distributed memory. In parallel computing, multiple computers -- or even multiple processor cores within the same computer -- are called nodes. Each node in the parallel arrangement typically works on a portion of the overall computing problem. The challenge then is to synchronize the actions of each parallel node, exchange data between nodes and provide command and control over the entire parallel cluster. The message passing interface defines a standard suite of functions for these tasks. MPI is not endorsed as an official standard by any standards organization such as IEEE or ISO, but it is generally considered to be the industry standard and it forms the basis for most communication interfaces adopted by parallel computing programmers. The older MPI 1.3 standard (dubbed MPI-1) provides over 115 functions. The later MPI 2.2 standard (or MPI-2) offers over 500 functions and is largely backward compatible with MPI-1. However, not all MPI libraries provide a full implementation of MPI-2.
MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster. A MapReduce program is composed of a **map** procedure (or method), which performs filtering and sorting (such as sorting students by first name into queues, one queue for each name), and a *reduce* method, which performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

22. Write a note on Network Treats and Data Integrity

Clusters, grids, P2P networks, and clouds demand security and copyright protection if they are to be accepted in today's digital society. 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. 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. Lack of authentication or authorization leads to attackers' illegitimate use of computing resources. Open resources such as data centers, P2P networks, and grid and cloud infrastructures could become the next targets. Users need to protect clusters, grids, clouds, and P2P systems.

Otherwise, users should not use or trust them for outsourced work. Malicious intrusions to these systems may destroy valuable hosts, as well as network and storage resources. Internet anomalies found in routers, gateways, and distributed hosts may hinder the acceptance of these public-resource computing services.

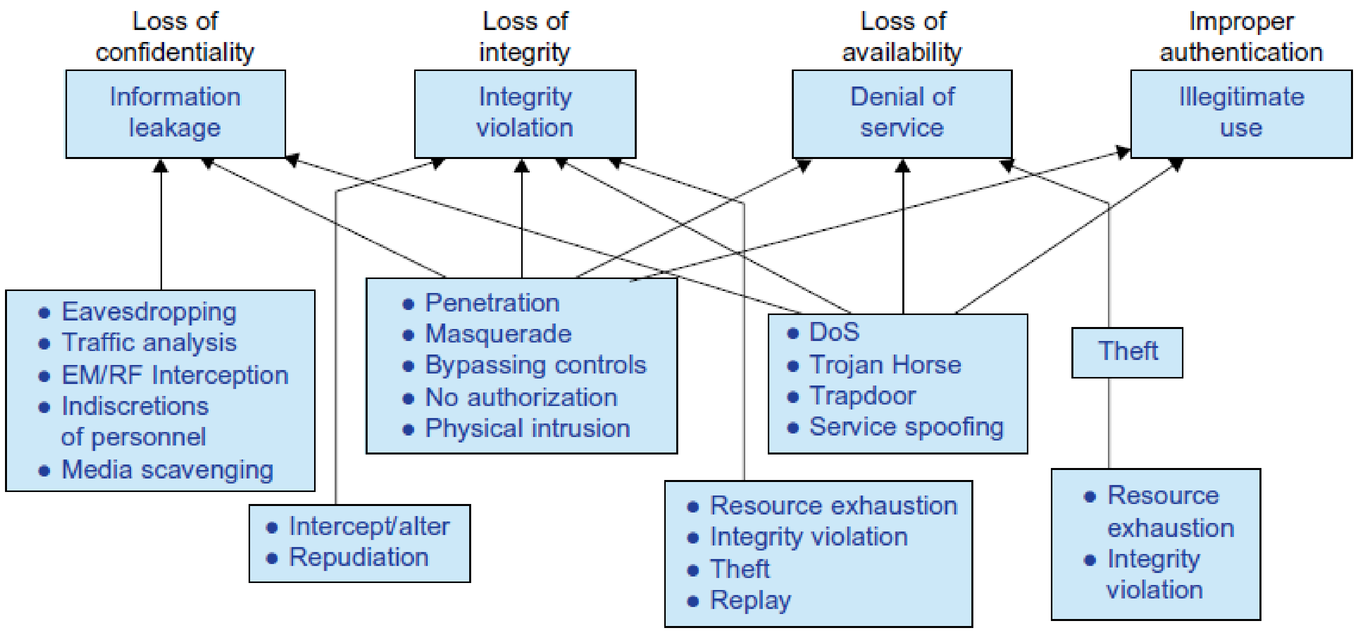

Loss of confidentiality
Loss of integrity
Loss of availability
Improper authentication
Information leakage
Integrity violation
Denial of service
Illegitimate use
• Eavesdropping • Traffic analysis • EM/RF Interception • Indiscretions of personnel • Media scavenging
• Penetration • Masquerade • Bypassing controls • No authorization • Physical intrusion
• DoS • Trojan Horse • Trapdoor • Service spoofing
Theft
• Intercept/alter • Repudiation
• Resource exhaustion • Integrity violation • Theft • Replay
• Resource exhaustion • Integrity violation

FIGURE 1.25

Various system attacks and network threats to the cyberspace, resulting 4 types of losses.

Three security requirements are often considered: confidentiality, integrity, and availability for most Internet service providers and cloud users. In the order of SaaS, PaaS, and IaaS, the providers gradually release the responsibility of security control to the cloud users. In summary, the SaaS model relies on the cloud provider to perform all security functions. At the other extreme, the IaaS model wants the users to assume almost all security functions, but to leave availability in the hands of the providers. The PaaS model relies on the provider to maintain data integrity and availability, but burdens the user with confidentiality and privacy control.
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. One can develop a proactive content poisoning scheme to stop colluders and pirates from alleged copyright infringements in P2P file sharing. Pirates are detected in a timely manner with identity-based signatures and timestamped tokens. This scheme stops collusive piracy from occurring without hurting
legitimate P2P clients.
Three generations of network defense technologies have appeared in the past. In the first generation, tools were designed to prevent or avoid intrusions. These tools usually manifested themselves as access control policies or tokens, cryptographic systems, and so forth. However, an intruder could always penetrate a secure system because there is always a weak link in the security provisioning process. The second generation detected intrusions in a timely manner to exercise remedial actions. These techniques included firewalls, intrusion detection systems (IDSes), PKI services, reputation systems, and so on. The third generation provides more intelligent responses to intrusions.
Security infrastructure is required to safeguard web and cloud services. At the user level, one needs to perform trust negotiation and reputation aggregation over all users. At the application end, we need to establish security precautions in worm containment and intrusion detection against virus, worm, and distributed DoS (DDoS) attacks. We also need to deploy mechanisms to prevent online piracy and copyright violations of digital content. Security responsibilities are divided between cloud providers and users differently for the three cloud service models. The providers are totally responsible for platform availability. The IaaS users are more responsible for the confidentiality issue. The IaaS providers are more responsible for data integrity. In PaaS and SaaS services, providers and users are equally responsible for preserving data integrity and confidentiality.

23. Explain fault tolerance and system availability

In addition to performance, system availability and application flexibility are two other important design goals in a distributed computing system. 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.
In general, as a distributed system increases in size, availability decreases due to a higher chance of failure and a difficulty in isolating the failures. Both SMP and MPP are very vulnerable with centralized resources under one OS. NUMA machines have improved in availability due to the use of multiple OSes. Most clusters are designed to have HA with failover capability. Meanwhile, private clouds are created out of virtualized data centers; hence, a cloud has an estimated availability similar to that of the hosting cluster. A grid is visualized as a hierarchical cluster of clusters. Grids have higher availability due to the isolation of faults. Therefore, clusters, clouds, and grids have decreasing availability as the system increases in size. A P2P file-sharing network has the highest aggregation of client machines. However, it operates independently with low availability, and even many peer nodes depart or fail simultaneously.

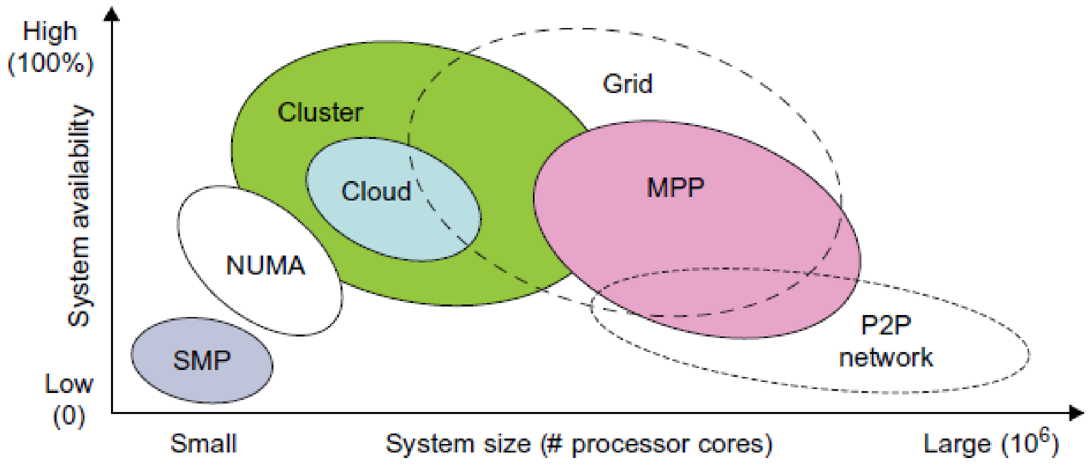


FIGURE 1.24

Estimated system availability by system size of common configurations in 2010.

24. Describe computational grid
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. The resource sites offer complementary computing resources, including workstations, large servers, a mesh of processors, and Linux clusters to satisfy a chain of computational needs. The grid is built across various IP broadband networks including LANs and WANs already used by enterprises or organizations over the Internet. The grid is presented to users as integrated resources pool as shown in the upper half of the figure. Special instruments may be involved such as using the radio telescope in SETI@Home search of life in the galaxy and the austrophysics@Swineburne for pulsars. At the server end, the grid is a network. At the client end, we see wired or wireless terminal devices. The grid integrates the computing, communication, contents, and transactions as rented services. Enterprises and consumers form the user base, which then defines the usage trends and service characteristics.
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. Grid systems are classified in essentially two categories: computational or data grids and P2P grids. Computing or data grids are built primarily at the national level.

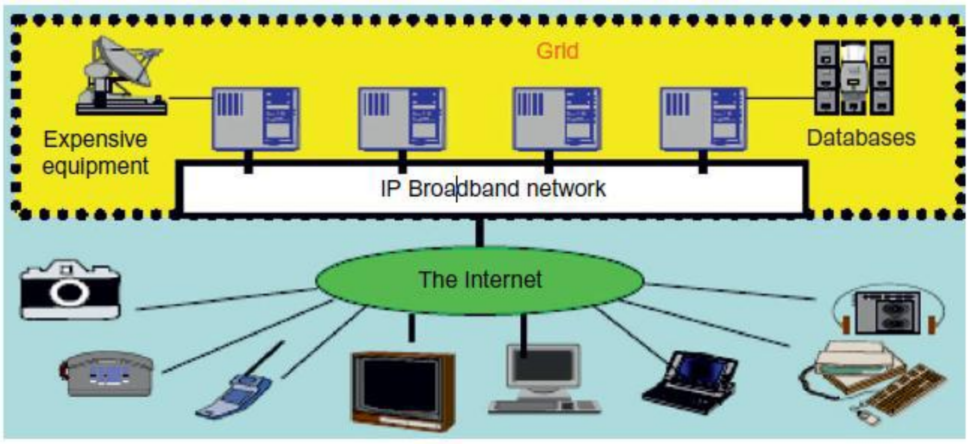


FIGURE 1.16
Computational grid or data grid providing computing utility, data, and information services through resource sharing and cooperation among participating organizations.

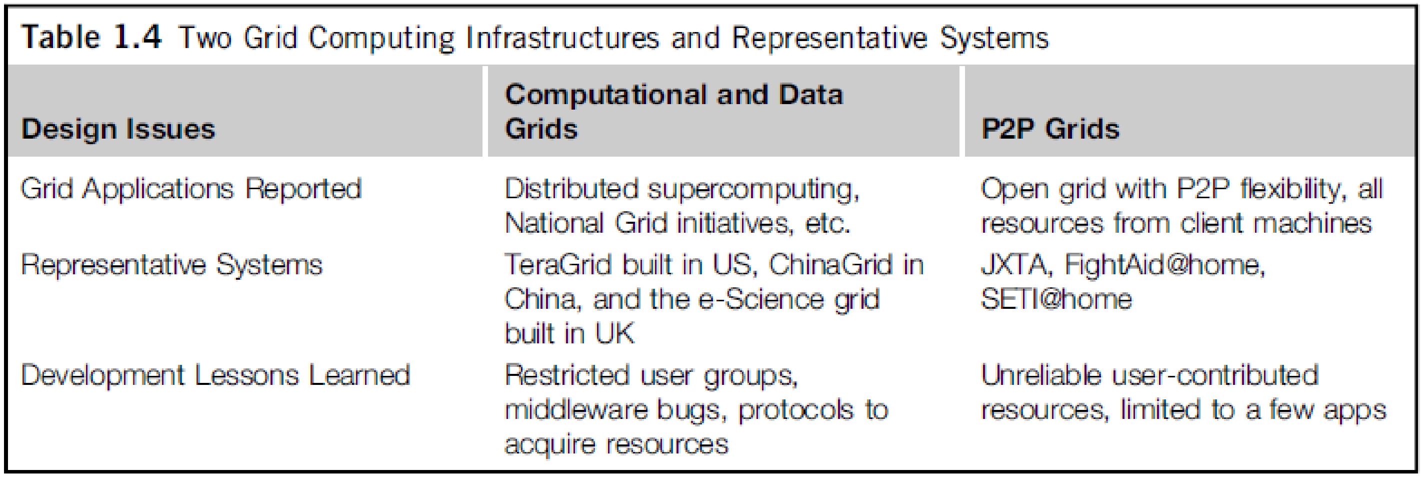


Table 1.4 Two Grid Computing Infrastructures and Representative Systems

Design Issues	Computational and Data Grids	P2P Grids
Grid Applications Reported	Distributed supercomputing, National Grid initiatives, etc.	Open grid with P2P flexibility, all resources from client machines
Representative Systems	TeraGrid built in US, ChinaGrid in China, and the e-Science grid built in UK	JXTA, FightAid@home, SETI@home
Development Lessons Learned	Restricted user groups, middleware bugs, protocols to acquire resources	Unreliable user-contributed resources, limited to a few apps

25. Explain various levels of implementing virtualization

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.

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.

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.

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.

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 a sprocess-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. 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.

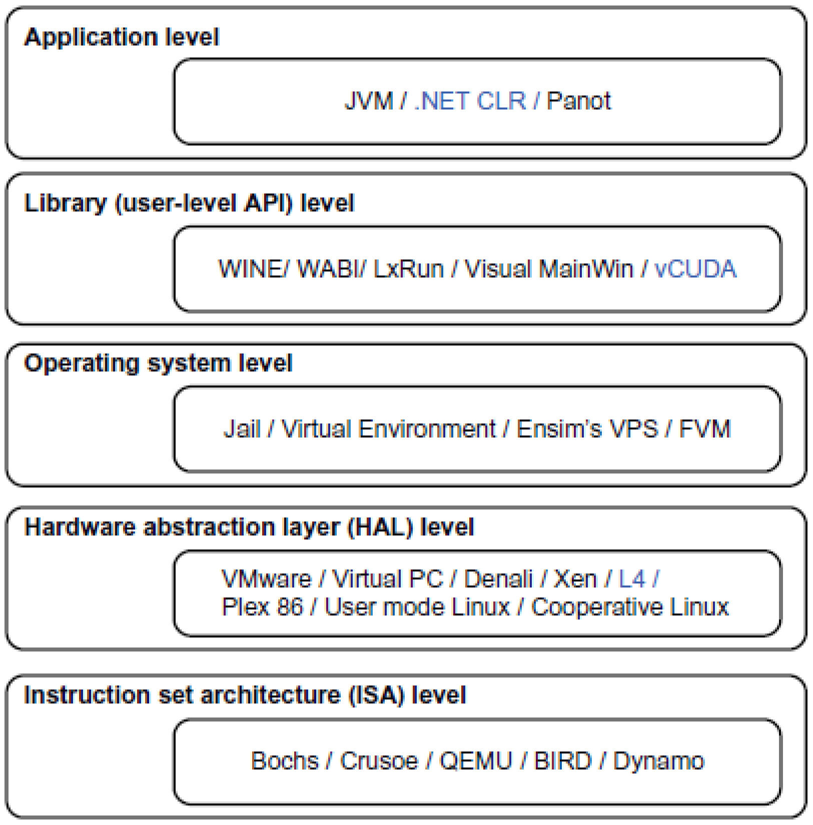

Application level
JVM / .NET CLR / Panot
Library (user-level API) level
WINE/ WABI/ LxRun / Visual MainWin / vCUDA
Operating system level
Jail / Virtual Environment / Ensim's VPS / FVM
Hardware abstraction layer (HAL) level
VMware / Virtual PC / Denali / Xen / L4 /
Plex 86 / User mode Linux / Cooperative Linux
Instruction set architecture (ISA) level
Bochs / Crusoe / QEMU / BIRD / Dynamo

FIGURE 3.2

Virtualization ranging from hardware to applications in five abstraction levels.