1. a. Explain the evolutionary trend of scalable computing

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 minicomputer 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. Figure 1.1 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.

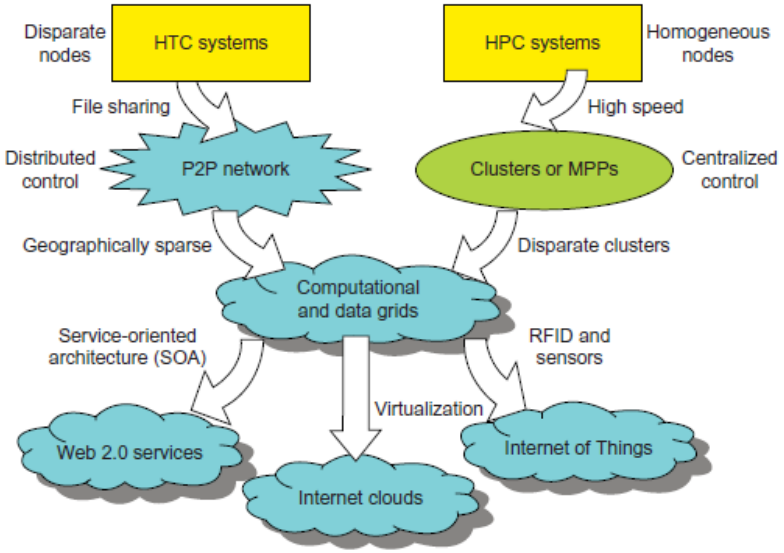


FIGURE 1.1

Evolutionary trend toward parallel, distributed, and cloud computing with clusters, MPPs, P2P networks, grids, clouds, web services, and the Internet of Things.

b. Differentiate between high performance computing and high throughput computing

HTC = High Throughput Computing
HPC = High Performance Computing

Below are some attributes of HTC and HPC that highlight the differences. HPC and HTC jobs can run on the same cluster architecture, but use the resources differently. In summary:
HTC jobs generally involve running multiple independent instances of software on multiple processors, at the same time. Serial systems are suitable for these requirements.
HPC jobs generally involve running a single instance of parallel software over many processors. Results at various instances throughout the computation are communicated among the processors, requiring a parallel environment.

More detail:

HTC: use when one needs to:
-run many jobs that are typically similar but not highly parallel;
-run the same program with varying inputs;
-run jobs that do not communicate with each other;
-execute on physically distributed resources using grid-enabled technologies;
-make use of many computing resources over long periods of time to accomplish a computational task.

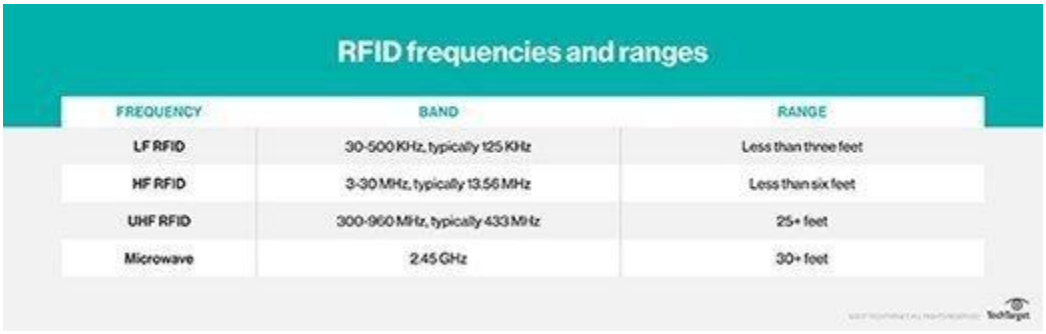
HPC: use when one needs to:
-run jobs where rapid communication of intermediate results is required to perform the computations;
-make intense use of large amounts of computing resources in relatively short time periods.

c. Define RFID, GPS and IoT with examples

An RFID system consists of three components: a scanning antenna and transceiver (often combined into one reader, also known as an interrogator) and a transponder, the RFID tag. An RFID tag consists of a microchip, memory and antenna The RFID reader is a network-connected device that can be permanently attached or portable. It uses radio frequency waves to transmit signals that activate the tag. Once activated, the tag sends a wave back to the antenna, where it is translated into data.

Types of RFID tags

There are two main types of RFID tags: active RFID and passive RFID. An active RFID tag has its own power source, often a battery. A passive RFID tag, on the other hand, does not require batteries; rather it receives its power from the reading antenna, whose electromagnetic wave induces a current in the RFID tag's antenna. There are also semi-passive RFID tags, meaning a battery runs the circuitry while communication is powered by the RFID reader



GPS, which stands for Global Positioning System, is a radio navigation system that allows land, sea, and airborne users to determine their exact location, velocity, and time 24 hours a day, in all weather conditions, anywhere in the world. The capabilities of today's system render other well-known navigation and positioning "technologies"—namely the magnetic compass, the sextant, the chronometer, and radio-based devices—impractical and obsolete. GPS is used to support a broad range of military, commercial, and consumer applications.

24 GPS satellites (21 active, 3 spare) are in orbit at 10,600 miles above the earth. The satellites are spaced so that from any point on earth, four satellites will be above the horizon. Each satellite contains a computer, an atomic clock, and a radio. With an understanding of its own orbit and the clock, the satellite continually broadcasts its changing position and time. (Once a day, each satellite checks its own sense of time and position with a ground station and makes any minor correction.) On the ground, any GPS receiver contains a computer that "triangulates" its own position by getting bearings from three of the four satellites. The result is provided in the form of a geographic position - longitude and latitude - to, for most receivers, within a few meters.

If the receiver is also equipped with a display screen that shows a map, the position can be shown on the map. If a fourth satellite can be received, the receiver/computer can figure out the altitude as well as the geographic position. If you are moving, your receiver may also be able to calculate your speed and direction of travel and give you estimated times of arrival to specified destinations. Some specialized GPS receivers can also store data for use in Geographic Information Systems (GIS) and map making.

The Internet of Things (IoT) refers to the use of intelligently connected devices and systems to leverage data gathered by embedded sensors and actuators in machines and other physical objects. In other **words**, the **IoT** (Internet of Things) can be called to any of the physical objects connected with network.

Applications:
Consumer Applications – Smart Home, Elder Case
Commercial Applications – Medical and Healthcare, Transportation, Building and home automation
Industrial Applications – Manufacturing, Agriculture
Infrastructure Applications – Metropolitan Scale Deployments, Energy Management, Environmental Monitoring

Consumer connected devices include smart TVs, smart **speakers**, toys, wearables and smart appliances. Smart meters, commercial security systems and smart city technologies -- such as those used to **monitor traffic** and weather conditions -- are examples of industrial and enterprise IoT devices.

2. a. Explain centralized computing, parallel, distributed and cloud 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. Inter-processor 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

b. What are clusters of cooperative computers?

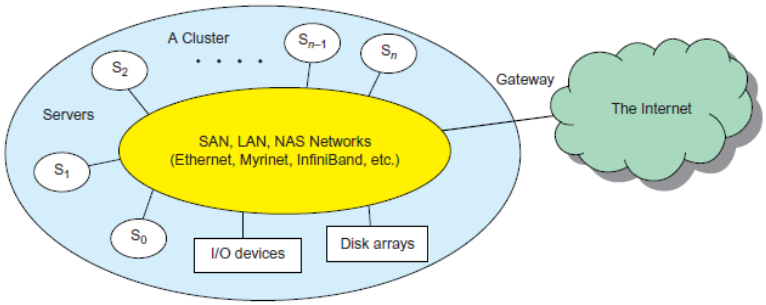


FIGURE 1.15
A cluster of servers interconnected by a high-bandwidth SAN or LAN with shared I/O devices and disk arrays; the cluster acts as a single computer attached to the Internet.

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.

3. a. Discuss major cluster design issues and their features.

Table 1.3 Critical Cluster Design Issues and Feasible Implementations

Features	Functional Characterization	Feasible Implementations
Availability and Support	Hardware and software support for sustained HA in cluster	Failover, failback, check pointing, rollback recovery, nonstop OS, etc.
Hardware Fault Tolerance	Automated failure management to eliminate all single points of failure	Component redundancy, hot swapping, RAID, multiple power supplies, etc.
Single System Image (SSI)	Achieving SSI at functional level with hardware and software support, middleware, or OS extensions	Hardware mechanisms or middleware support to achieve DSM at coherent cache level
Efficient Communications	To reduce message-passing system overhead and hide latencies	Fast message passing, active messages, enhanced MPI library, etc.
Cluster-wide Job Management	Using a global job management system with better scheduling and monitoring	Application of single-job management systems such as LSF, Codine, etc.
Dynamic Load Balancing	Balancing the workload of all processing nodes along with failure recovery	Workload monitoring, process migration, job replication and gang scheduling, etc.
Scalability and Programmability	Adding more servers to a cluster or adding more clusters to a grid as the workload or data set increases	Use of scalable interconnect, performance monitoring, distributed execution environment, and better software tools

b. Explain Peer-to-Peer network families with diagram

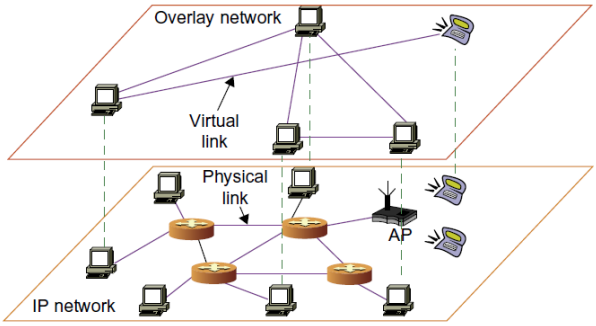


FIGURE 1.17

The structure of a P2P system by mapping a physical IP network to an overlay network built with virtual links.

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. 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 above 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.

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. 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 networks 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.

4. a. What is cloud computing? Explain cloud architecture

Cloud computing is shared pools of configurable computer system resources and higher-level services that can be rapidly provisioned with minimal management effort, often over the Internet. Cloud computing relies on sharing of resources to achieve coherence and economies of scale, similar to a public utility.

Third-party clouds enable organizations to focus on their core businesses instead of expending resources on computer infrastructure and maintenance. Advocates note that cloud computing allows companies to avoid or minimize up-front IT infrastructure costs. Proponents also claim that cloud computing allows enterprises to get their applications up and running faster, with improved manageability and less maintenance, and that it enables IT teams to more rapidly adjust resources to meet fluctuating and unpredictable demand. Cloud providers typically use a "pay-as-you-go" model, which can lead to unexpected operating expenses if administrators are not familiarized with cloud-pricing models.

The availability of high-capacity networks, low-cost computers and storage devices as well as the widespread adoption of hardware virtualization, service-oriented architecture, and autonomic and utility computing has led to growth in cloud computing

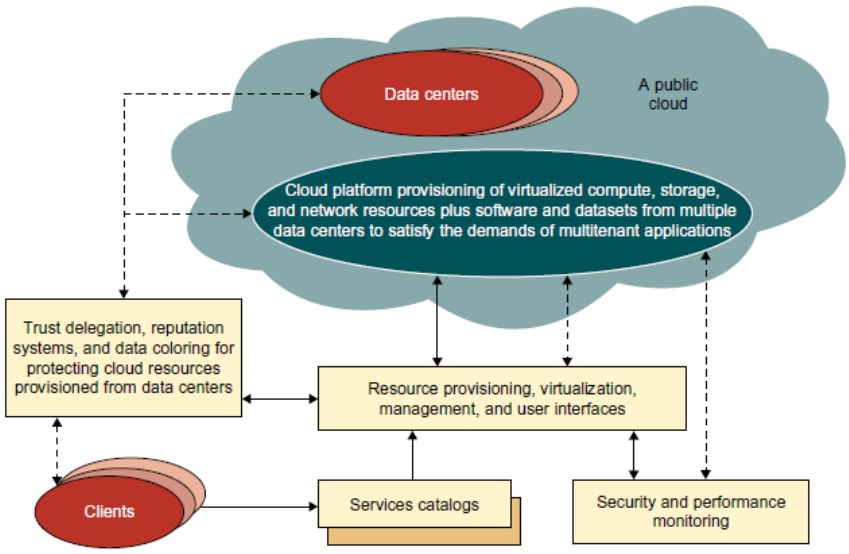


FIGURE 4.14

A security-aware cloud platform built with a virtual cluster of VMs, storage, and networking resources over the data-center servers operated by providers.

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. 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.

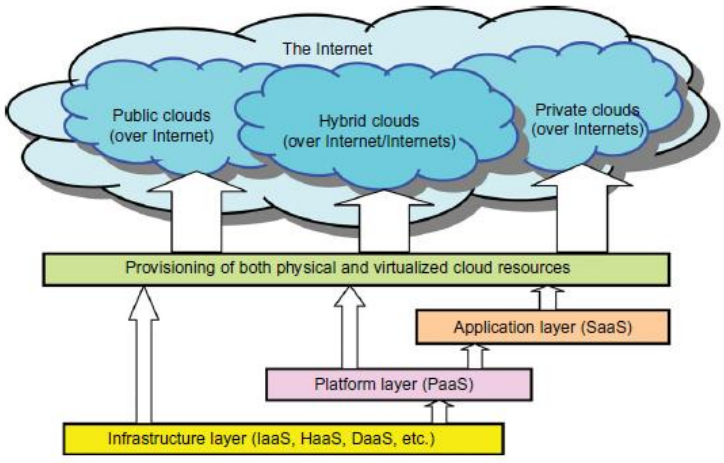


FIGURE 4.15

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

The architecture of a cloud is developed at three layers: infrastructure, platform, and application. 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. 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.

b. Explain performance metrics in distributed systems

HPC performance has been well defined over the years. The most popular metrics used are Gflops and Tflops in the past, Pflops on top systems now, and Eflops for the future. Parallel benchmarking programs are available to evaluate HPC systems in batch processing of a large-scale problem. Well-known benchmark programs include the Linpack Benchmark, NAS, Splash, and Parkbench. On the other hand, HTC performance on business servers, clusters, data centers, and cloud systems is much more complex a problem to evaluate because the systems are used by many clients simultaneously. Thus, HTC performance is attributed to many orthogonal factors. Some of them are measurable and some are not. The basic assumption is that many (millions or more) independent users may use the shared resources in the cloud or data centers simultaneously. The goal is to satisfy as many concurrent users as possible, even if each user program is a simple web service, social contact, or simple cloud storage. In this section, we attempt to present some performance evaluations and QoS assessment models for both HPC and HTC systems.

Basic Performance Attributes

The performance models applied to evaluate MPPs, data-center clusters, and virtualized clouds could be very different. To fit the layered structure of the cloud computing paradigm, the performance metrics of IaaS serve as a basis to model PaaS performance. Similarly, the attributes applicable to model PaaS performance serve as a foundation to evaluate SaaS performance. A good model should be able to cover all the layers of computing services. The model is generic in the sense that it applies to various cloud platforms under different workload distributions. To assess QoS in cloud services, the impacts on the top layer (SaaS) should remain transparent to the lower layers (PaaS and IaaS). A cloud platform should be built to serve many users simultaneously. Therefore, multitasking is a necessity to assess distributed system performance. Five basic performance metrics are introduced in this section: namely system throughput, multitasking scalability, availability measure, data security, and cost-effectiveness. This analytical model offers a first-order approximation to model the performance of any HPC or HTC system, including elastic clouds. One can always add more performance attributes to expand the model from the user's perspective to cover more dimensions or those concerns by service providers. In other words, refined performance models could be extended from basic attributes to include program behavior, environmental demand, QoS, and cost-effectiveness.

System Throughput and Efficiency

In general, the system throughput of a distributed system or a cloud platform measures the number of jobs that can be done per unit of time. The throughput measure is attributed to several key factors that affect the total execution time (T_{total}) of all jobs processed in a given time window. First, we have to examine all component times in T_{total}. Each job submitted may experience an initiation time to acquire resources. This overhead time includes the boot time of all machine instances and the scheduling time of incoming user jobs. The type of the application program limits horizontal scalability. In general, the overhead can be attributed to five components: infrastructure initiation delay, resource provisioning delay, interjob communication delay, OS overhead, and loading application software overhead. The OS and software overheads remain constant in running different jobs. The other three time factors may vary with problem size and system management policies. For simplicity in analysis, we lump all five time factors into a single overhead time kT_o, where T_o is their sum averaged over many jobs running in a fixed time interval. The constant k is a platform-dependent coefficient that varies with specific system configurations. This overhead applies to all three system models presented in this book. Let $T_e(n, m)$ be the effective execution time to complete n independent jobs in a cloud platform, where m accounts for the number of machine instances in a given system configuration. These machine instances could be cluster nodes (or processor cores) if physical servers are used, or VM counts if a virtualized cloud is used.

We compute the total job execution time (or makespan) with $T_{total} = kT_o + T_e(n, m)$. Then the system throughput (π) is defined by:

$\pi = n/T_{total} = n/[kT_o + T_e(n,m)]$
In the ideal case, the overhead kT_o is assumed to be very small or simply zero, compared with the magnitude of the execution time T_e. Hence, the ideal throughput is simply measured by $n/T_e(n, m)$.
The system efficiency (α) is defined by normalized throughput as follows: $\alpha = \pi/[n/T_e(n,m)] = T_e(n,m)/[kT_o + T_e(n,m)]$
System efficiency represents the effective utilization of all provisioned resources in a system. Note that the parameters n and m in the expression $T_e(n, m)$ vary with the workload (user number), system size (m), the parallel and distributed computing model applied.

5 a. What is virtualization? Explain implementation levels of virtualization

Virtualization is the creation of a virtual -- rather than actual -- version of something, such as an operating system, a server, a storage device or network resources. Virtualization describes a technology in which an application, guest operating system or data storage is abstracted away from the true underlying hardware or software. A key use of virtualization technology is server virtualization, which uses a software layer called a hypervisor to emulate the underlying hardware. This often includes the CPU's memory, I/O and network traffic. The guest operating system, normally interacting with true hardware, is now doing so with a software emulation of that hardware, and often the guest operating system has no idea it's on virtualized hardware. While the performance of this virtual system is not equal to the performance of the operating system running on true hardware, the concept of virtualization works because most guest operating systems and applications don't need the full use of the underlying hardware. This allows for greater flexibility, control and isolation by removing the dependency on a given hardware platform. While initially meant for server virtualization, the concept of virtualization has spread to applications, networks, data and desktops.

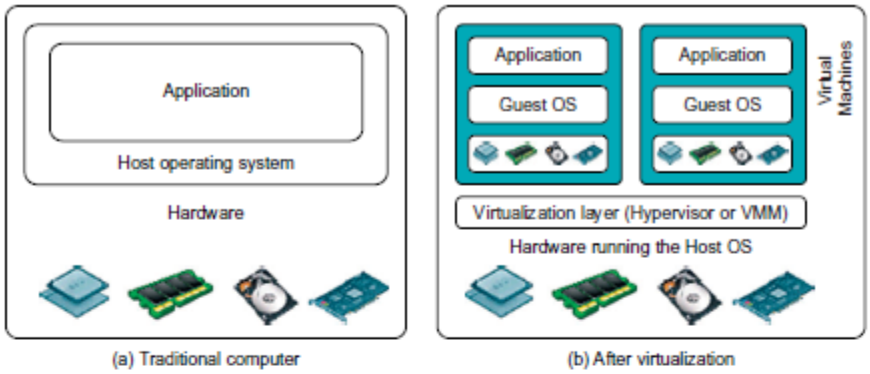


FIGURE 3.1
The architecture of a computer system before and after virtualization, where VMM stands for virtual machine monitor.

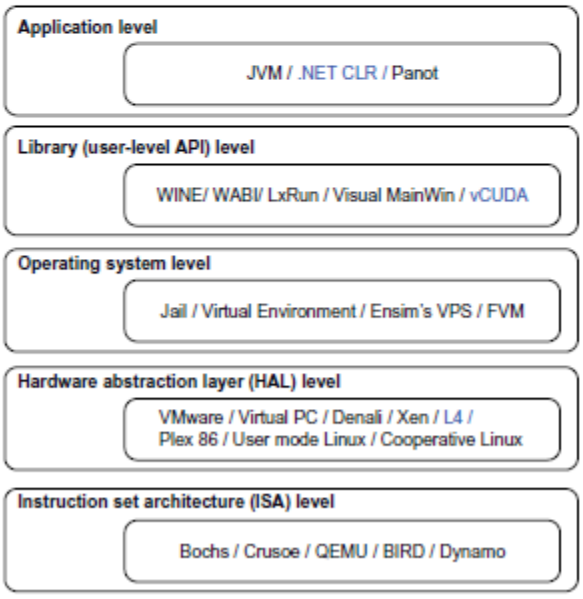


FIGURE 3.2
Virtualization ranging from hardware to applications in five abstraction levels.

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 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.

b. Define VMM with diagram of abstraction levels

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

6. a. Explain OS level virtualization

Operating system virtualization (OS virtualization) is a server virtualization technology that involves tailoring a standard operating system so that it can run different applications handled by multiple users on a single computer at a time. The operating systems do not interfere with each other even though they are on the same computer.

In OS virtualization, the operating system is altered so that it operates like several different, individual systems. The virtualized environment accepts commands from different users running different applications on the same machine. The users and their requests are handled separately by the virtualized operating system.

Operating system virtualization provides application-transparent virtualization to users by decoupling applications from the OS. The OS virtualization technique offers granular control at the application level by facilitating the transparent migration of individual applications. The finer granularity migration offers greater flexibility, resulting in reduced overhead.

OS virtualization can also be used to migrate critical applications to another running operating system instance. Patches and updates to the underlying operating system are done in a timely way, and have little or no impact on the availability of application services. The processes in the OS virtualized environment are isolated and their interactions with the underlying OS instance are monitored.

b. Differentiate between 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 physi-cal 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 functional-ity, 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 hyper-visor 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.

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 organi-zation 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

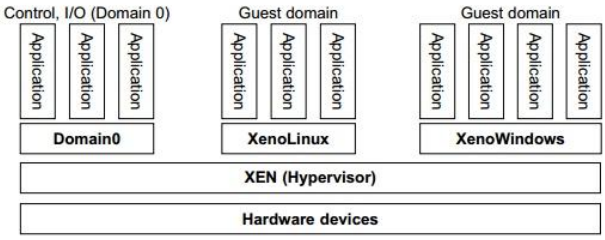


FIGURE 3.5

The Xen architecture's special domain 0 for control and I/O, and several guest domains for user applications.

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, con-figuration 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).

7. a. Define public, private and hybrid clouds

The Public Cloud

In the Public Cloud space, Windows Azure, Amazon Cloud Services and Rackspace are big players. Amazon elastic compute cloud (EC2) for example, provides the infrastructure and services over the public internet and are hosted at the cloud vendor's premises. The general public, SMEs or large enterprise groups can leverage this cloud model. Here the infrastructure is owned by the company that provides the cloud services. In a public cloud, the infrastructure and services are provisioned from a remote location hosted at the cloud provider's datacenter and the customer has no control and limited visibility over where the service is hosted. But they can use those services anytime anywhere as needed. In the Public Cloud, the core computing infrastructure is shared among several organizations. That said, each organization's data, applications, and infrastructure are separated and can only be accessed by the authorized personnel.

The Public Cloud offers advantages such as low cost of ownership, automated deployments, scalability and also reliability. The Public Cloud is well suited for the following:

- Data storage
- Data Archival
- Application Hosting
- Latency intolerant or mission critical web tiers
- On demand hosting for microsite and application.
- Auto-scaling environment for large applications.

The Private Cloud

A Private Cloud, as the name suggests, is a cloud infrastructure that is meant for use exclusively by a single organization. The cloud is then owned, managed and operated exclusively by the organization or by a third-party vendor or both together. In this cloud model, the infrastructure is provisioned on the organization premise but may be hosted in a third-party data center. However, in most cases a Private Cloud infrastructure is implemented and hosted in an on-premise data center using a virtualization layer. Private cloud environments offer greater configurability support to any application and even support those legacy applications that suffer from performance issues in Public Clouds.

While the Private Cloud offers the greatest level of control and security, it does demand that the organization purchase and maintain all the infrastructure and acquire and retain the skill to do so. This makes the Private Cloud significantly more expensive and a not-so-viable option for small or mid-sized organizations.

Choosing a Private Cloud makes sense for:

- Organization that demand strict security, latency, regulatory and data privacy levels.
- Organizations that are highly regulated and need data hosted privately and securely.
- Organizations that are large enough to support the costs that go into running a next-gen cloud data center.
- Organizations that need high-performance access to a filesystem such as in media companies.
- Hosting applications that have predictable usage patterns and demand low storage costs.
- Organizations that demand greater adaptability, configurability, and flexibility.
- Hosting business critical data and applications.

The Hybrid Cloud

So, what does an organization do when it wants to leverage the cloud both for its efficiency and cost saving but also wants security, privacy, and control? It looks at the Hybrid Cloud which almost serves as a mid-way point between Public and Private Cloud. The Hybrid Cloud uses a combination of at least one Private and one Public Cloud. The Private Cloud can be on-premise or even a virtual private cloud located outside the organization's data center. A Hybrid Cloud can also consist of multiple Private and Public Clouds and may use many active servers, physical or virtualized, which are not a part of the Private Cloud. With the Hybrid Cloud, organizations can keep each business aspect in the most efficient cloud format possible. However, with the Hybrid Cloud, organizations have to manage multiple security platforms and aspects and also ensure that all the cloud properties can communicate seamlessly with one another.

A Hybrid Cloud is best suited for:

- Large organizations that want the flexibility and scalability as offered by the public cloud.
- Organizations that offer services for vertical markets- customer interactions can be hosted in the Public Cloud while company data can be hosted in the Private Cloud.
- Organizations that demand greater operational flexibility and scalability. For them, mission critical data can be hosted on the Private Cloud and application development and testing can take place in the Public Cloud.

b. Explain data-centre networking structure

Types of Data center network

Three-tier DCN

The legacy three-tier DCN architecture follows a multi-rooted tree based network topology composed of three layers of network switches, namely access, aggregate, and core layers. The servers in the lowest layers are connected directly to one of the edge layer switches. The aggregate layer switches interconnects multiple access layer switches together. All of the aggregate layer switches are connected to each other by core layer switches. Core layer switches are also responsible for connecting the data center to the Internet. The three-tier is the common network architecture used in data centers. However, three-tier architecture is unable to handle the growing demand of cloud computing. The higher layers of the three-tier DCN are highly oversubscribed. Moreover, scalability is another major issue in three-tier DCN. Major problems faced by the three-tier architecture include, scalability, fault tolerance, energy efficiency, and cross-sectional bandwidth. The three-tier architecture uses enterprise-level network devices at the higher layers of topology that are very expensive and power hungry.

Fat tree DCN

Fat tree DCN architecture handles the oversubscription and cross section bandwidth problem faced by the legacy three-tier DCN architecture. Fat tree DCN employs commodity network switches based architecture using Clos topology. The network elements in fat tree topology also follows hierarchical organization of network switches in access, aggregate, and core layers. However, the number of network switches is much larger than the three-tier DCN. The architecture is composed of *k* pods, where each pod contains, $(k/2)^2$ servers, k/2 access layer switches, and k/2 aggregate layer switches in the topology. The core layers contain $(k/2)^2$ core switches where each of the core switches is connected to one aggregate layer switch in each of the pods. The fat tree topology offers 1:1 oversubscription ratio and full bisection bandwidth. The fat tree architecture uses a customized addressing scheme and routing algorithm. The scalability is one of the major issues in fat tree DCN architecture and maximum number of pods is equal to the number of ports in each switch.

DCell

DCell is a server-centric hybrid DCN architecture where one server is directly connected to many other servers. A server in the DCell architecture is equipped with multiple Network Interface Cards (NICs). The DCell follows a recursively built hierarchy of cells. A $cell_0$ is the basic unit and building block of DCell topology arranged in multiple levels, where a higher level cell contains multiple lower layer cells. The $cell_0$ is building block of DCell topology, which contains *n* servers and one commodity network switch. The network switch is only used to connect the server within a $cell_0$. A $cell_1$ contain $k=n+1$ $cell_0$ cells, and similarly a $cell_2$ contains k * n + 1 $dcell_1$. The DCell is a highly scalable architecture where a four level DCell with only six servers in $cell_0$ can accommodate around 3.26 million servers. Besides very high scalability, the DCell architecture depicts very high structural robustness. However, cross section bandwidth and network latency is a major issue in DCell DCN architecture.

c. List out and explain 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.

8. a. Explain AWS, GAE and Ms Azure as cloud platforms

AWS vs Azure vs Google

The competition is heating up in the public cloud space as vendors regularly drop prices and offer new features. In this article, we will shine a light on the competition between the three giants of the cloud: Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft's Azure. While AWS has a significant head start on the others, Google and Microsoft are far from out of the race. As of today, March 23rd, 2016 Google is planning 12 new cloud data centers in next 18 months. They've both got the power, money, technology, and marketing to attract individual and enterprise customers. Let's compare these three big players by service category: compute, storage, networking, and pricing structure.

AWS vs Azure vs Google: cloud players

AWS vs Azure vs Google: Compute

AWS's EC2 (Elastic Compute Cloud) provides Amazon's core compute service, allowing users to configure virtual machines using either pre-configured or custom AMIs (machine images). You select the size, power, memory capacity, and number of VMs and choose from among different regions and availability zones within which to launch. EC2 also allows load balancing (ELB) and auto scaling. ELB distributes loads across instances for better performance, and auto-scaling allow users to automatically scale available EC2 capacity up or down.
In 2012, Google introduced their computing cloud service: Google Compute Engine (GCE). Google Compute Engine lets users launch virtual machines, much like AWS, into regions and availability groups. However, GCE didn't become available for everyone until 2013. Since then Google has added its own enhancements, like load balancing, extended support for Operating Systems, live migration of VMs, faster persistent disks, and instances with more cores.
Also in 2012, Microsoft introduced their compute service as a preview, but didn't make it generally available until May 2013. Azure users choose a VHD (Virtual Hard Disk), which is equivalent to Amazon's AMI, to create a VM. A VHD can be either predefined by Microsoft, by third parties, or be user-defined. With each VM, you need to specify the number of cores and amount of memory.
Table1 shows Big Three compute options:

	Instance Families	Instances types	Regions	Zones
AWS	7	38	Yes	Yes
GCE	4	18	Yes	Yes
Azure	4	33	Yes	

Table1: AWS vs Azure vs Google: Compute

AWS vs Azure vs Google: Storage and databases

AWS provides ephemeral (temporary) storage that is allocated once an instance is started and is destroyed when the instance is terminated. It provides Block Storage that is equivalent to hard disks, in that it can either be attached to any instance or kept separate. AWS also offers object storage with their S3 Service, and archiving services with Glacier. AWS fully supports relational and NoSQL databases and Big Data.
Google's Cloud Platform similarly provides both temporary storage and persistent disks. For Object storage, GCP has Google Cloud Storage. GCP supports relational DBs through Google Cloud SQL. Technologies pioneered by Google, like Big Query, Big Table, and

Hadoop, are naturally fully supported. Google's Nearline offers archiving as cheap as Glacier, but with virtually no latency on recovery. Azure uses temporary storage (D drive) and Page Blobs (Microsoft's Block Storage option) for VM-based volumes. Block Blobs and Files serve for Object Storage. Azure supports both relational and NoSQL databases, and Big Data, through Windows Azure Table and HDInsight.
Table2 shows a comparison of the three clouds in storage and DBs.

	Ephemeral (Temporary)	Block Storage	Object Storage	Relational DB	Archiving	NoSQL and Big Data
AWS	Yes	EBS	S3	RDS	Glacier	DynamoDB, EMR, Kinesis, Redshift
GCP	Yes	Persistent disks	Google Cloud Storage	Google Cloud SQL	Nearline	Cloud Datastore, Big Query, Hadoop
Azure	Temporary Storage – D Drive	Page Blobs	Block Blobs and Files	Relational DBs		Windows Azure Table, HDInsight

Table 2: AWS vs Azure vs Google: Storage and databases

AWS vs Azure vs Google: Networking

Amazon's Virtual Private Clouds (VPCs) and Azure's Virtual Network (VNET) allow users to group VMs into isolated networks in the cloud. Using VPCs and VNETs, users can define a network topology, create subnets, route tables, private IP address ranges, and network gateways. There's not much to choose between AWS vs Azure on this: they both have solutions to extend your on-premise data center into the public (or hybrid) cloud. Each Google Compute Engine instance belongs to a single network, which defines the address range and gateway address for all instances connected to it. Firewall rules can be applied to an instance, and it can receive a public IP address.
AWS is unique in providing Route 53, a DNS web service.
Table 3 compares the three clouds from a networking point of view.

	Virtual network	Public IP	Hybrid Cloud	DNS	Firewall/ACL
AWS	VPC	Yes	Yes	Route 53	Yes
GCP	subnet	Yes			Yes
Azure	VNet	Yes	Yes		Yes

Table 3: AWS vs Azure vs Google: Networking
AWS vs Azure vs Google: Pricing Structure

AWS charges customers by rounding up the number of hours used, so the minimum use is one hour. AWS instances can be purchased using any one of three models:

- on demand – customers pay for what they use without any upfront cost
- reserved – customers reserve instances for 1 or 3 years with an upfront cost that is based on the utilization
- spot – customers bid for the extra capacity available

GCP charges for instances by rounding up the number of *minutes* used, with a minimum of 10 minutes. Google recently announced new *sustained-use pricing* for compute services that will offer a simpler and more flexible approach to AWS's reserved instances. Sustained-use pricing will discount the on-demand baseline hourly rate automatically as a particular instance is used for a larger percentage of the month.
Azure charges customers by rounding up the number of minutes used for on demand. Azure also offers short-term commitments with discounts.
Table 4 shows the comparison in Pricing and Models between the three public clouds.

	Pricing	Models
AWS	Per hour – rounded up	On demand, reserved, spot
GCP	Per minute – rounded up (minimum 10 minutes)	On demand – sustained use
Azure	Per minute – rounded up commitments (pre-paid or monthly)	On demand – short term commitments (pre-paid or monthly)

Table 4: AWS vs Azure vs Google: Pricing and Models

b. Define physical and cyber security

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.

9. a. List out and explain traditional features in cluster, grid, parallel computing environments

Table 6.3 Traditional Features in Cluster, Grid, and Parallel Computing Environments

Cluster management: ROCKS and packages offering a range of tools to make it easy to bring up clusters

Data management: Included metadata support such as RDF triple stores (Semantic web success and can be built on MapReduce as in SHARD); SQL and NOSQL included in

Grid programming environment: Varies from link-together services as in Open Grid Services Architecture (OGSA) to GridRPC (Ninf, GridSolve) and SAGA

OpenMP/threading: Can include parallel compilers such as Cilk; roughly shared memory technologies. Even transactional memory and fine-grained data flow come here

Portals: Can be called (science) gateways and see an interesting change in technology from portlets to HUBzero and now in the cloud: Azure Web Roles and GAE

Scalable parallel computing environments: MPI and associated higher level concepts including ill-fated HP FORTRAN, PGAS (not successful but not disgraced), HPCS languages (X-10, Fortress, Chapel), patterns (including Berkeley dwarves), and functional languages such as F# for distributed memory

Virtual organizations: From specialized grid solutions to popular Web 2.0 capabilities such as Facebook

Workflow: Supports workflows that link job components either within or between grids and clouds; relate to LIMS Laboratory Information Management Systems.

b. Explain Mapreduce, Hadoop in cloud programming environment

MapReduce is a software framework which supports parallel and distributed computing on large data sets. This software framework abstracts the data flow of running a parallel program on a distributed computing system by providing users with two interfaces in the form of two functions: Map and Reduce. Users can override these two functions to interact with and manipulate the data flow of running their programs. Figure 6.1 illustrates the logical data flow from the Map to the Reduce function in MapReduce frameworks. In this framework, the "value" part of the data, (key, value), is the actual data, and the "key" part is only used by the MapReduce controller to control the data flow

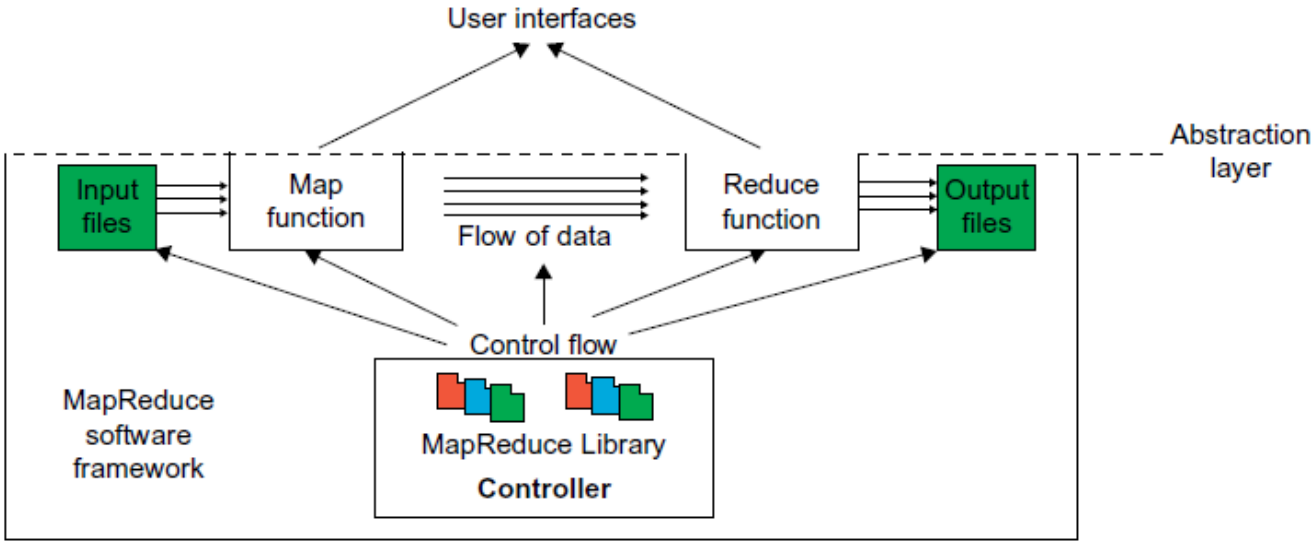


FIGURE 6.1

MapReduce framework: Input data flows through the Map and Reduce functions to generate the output result under the control flow using MapReduce software library. Special user interfaces are used to access the Map and Reduce resources.

c. What are GFS in software environment?

Google File System (GFS)

GFS was built primarily as the fundamental storage service for Google's search engine. As the size of the web data that was crawled and saved was quite substantial, Google needed a distributed file system to redundantly store massive amounts of data on cheap and

unreliable computers. None of the traditional distributed file systems can provide such functions and hold such large amounts of data. In addition, GFS was designed for Google applications, and Google applications were built for GFS. In traditional file system design, such a philosophy is not attractive, as there should be a clear interface between applications and the file system, such as a POSIX interface. There are several assumptions concerning GFS. One is related to the characteristic of the cloud computing hardware infrastructure (i.e., the high component failure rate). As servers are composed of inexpensive commodity components, it is the norm rather than the exception that concurrent failures will occur all the time. Another concerns the file size in GFS. GFS typically will hold a large number of huge files, each 100MB or larger, with files that are multiple GB in size quite common. Thus, Google has chosen its file data block size to be 64MB instead of the 4 KB in typical traditional file systems. The I/O pattern in the Google application is also special. Files are typically written once, and the write operations are often the appending data blocks to the end of files. Multiple appending operations might be concurrent. There will be a lot of large streaming reads and only a little random access. As for large streaming reads, highly sustained throughput is much more important than low latency. Thus, Google made some special decisions regarding the design of GFS. As noted earlier, a 64 MB block size was chosen. Reliability is achieved by using replications (i.e., each chunk or data block of a file is replicated across more than three chunk servers). A single master coordinates access as well as keeps the metadata. This decision simplified the design and management of the whole cluster. Developers do not need to consider many difficult issues in distributed systems, such as distributed consensus. There is no data cache in GFS as large streaming reads and writes represent neither time nor space locality. GFS provides a similar, but not identical, POSIX file system accessing interface. The distinct difference is that the application can even see the physical location of file blocks. Such a scheme can improve the upper-layer applications. The customized API can simplify the problem and focus on Google applications. The customized API adds snapshot and record append operations to facilitate the building of Google applications. Figure 6.18 shows the GFS architecture. It is quite obvious that there is a single master in the whole cluster. Other nodes act as the chunk servers for storing data, while the single master stores the metadata. The file system namespace and locking facilities are managed by the master. The master periodically communicates with the chunk servers to collect management information as well as give instructions to the chunk servers to do work such as load balancing or fail recovery. The master has enough information to keep the whole cluster in a healthy state. With a single master, many complicated distributed algorithms can be avoided and the design of the system can be simplified. However, this design does have a potential weakness, as the single GFS master could be the performance bottleneck and the single point of failure. To mitigate this, Google uses a shadow master to replicate all the data on the master, and the design guarantees that all the data operations are performed directly between the client and the chunk server. The control messages are transferred between the master and the clients and they can be cached for future use. With the current quality of commodity servers, the single master can handle a cluster of more than 1,000 nodes.

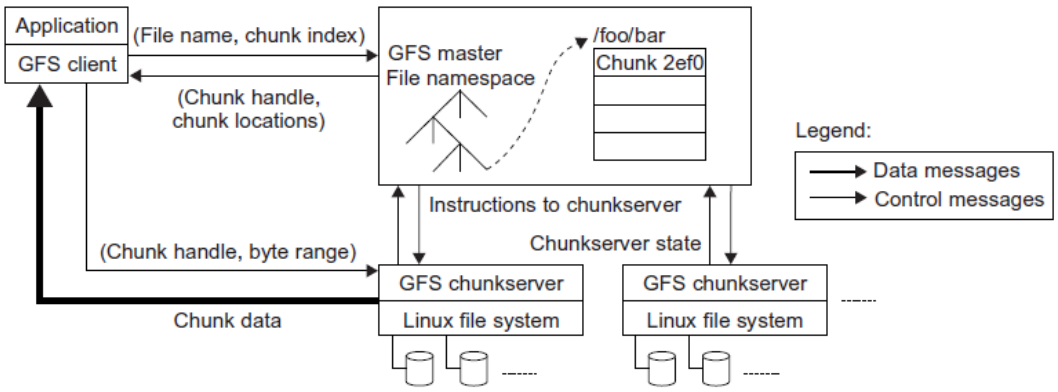


FIGURE 6.18

Architecture of Google File System (GFS).

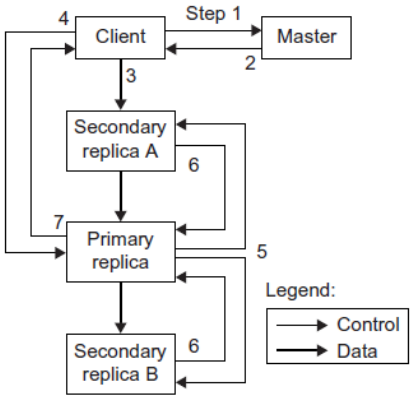


FIGURE 6.19

Data mutation sequence in GFS.

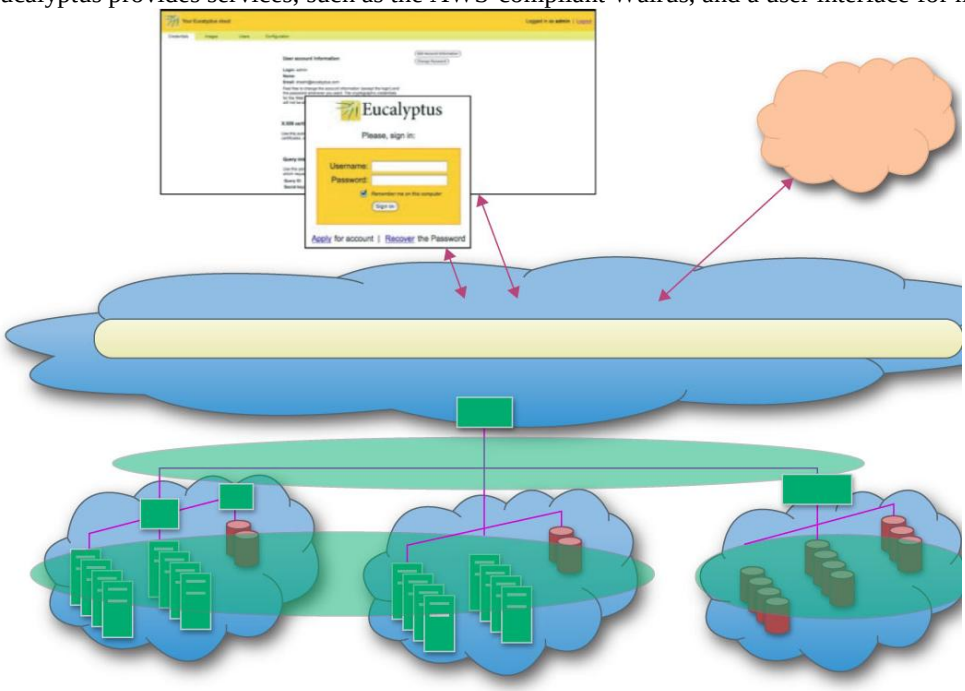
10. a. Explain Amazon EBS as cloud programming

The Elastic Block Store (EBS) provides the volume block interface for saving and restoring the virtual images of EC2 instances. Traditional EC2 instances will be destroyed after use. The status of EC2 can now be saved in the EBS system after the machine is shut down. Users can use EBS to save persistent data and mount to the running instances of EC2. Note that S3 is "Storage as a Service" with a messaging interface. EBS is analogous to a distributed file system accessed by traditional OS disk access mechanisms. EBS allows you to create storage volumes from 1 GB to 1 TB that can be mounted as EC2 instances. Multiple volumes can be mounted to the same instance. These storage volumes behave like raw, unformatted block devices, with user-supplied device names and a block device interface. You can create a file system on top of Amazon EBS volumes, or use them in any other way you would use a block device (like a hard drive). Snapshots are provided so that the data can be saved incrementally. This can improve performance when saving and restoring data. In terms of pricing, Amazon provides a similar pay-per-use schema as EC2 and S3. Volume storage charges are based on the amount of storage users allocate until it is released, and is priced at $0.10 per GB/month. EBS also charges $0.10 per 1 million I/O requests made to the storage (as of October 6, 2010). The equivalent of EBS has been offered in open source clouds such Nimbus.

b. Explain emerging cloud software environments

Open Source Eucalyptus and Nimbus

Eucalyptus is a product from Eucalyptus Systems (www.eucalyptus.com) that was developed out of a research project at the University of California, Santa Barbara. Eucalyptus was initially aimed at bringing the cloud computing paradigm to academic supercomputers and clusters. Eucalyptus provides an AWS-compliant EC2-based web service interface for interacting with the cloud service. Additionally, Eucalyptus provides services, such as the AWS-compliant Walrus, and a user interface for managing users and images.



OpenNebula, Sector/Sphere, and OpenStack

OpenNebula is an open source toolkit which allows users to transform existing infrastructure into an IaaS cloud with cloud-like interfaces. The architecture of OpenNebula has been designed to be flexible and modular to allow integration with different storage and network infrastructure configurations, and
hypervisor technologies. Here, the core is a centralized component that manages the VM full life cycle, including setting up networks dynamically for groups of VMs and managing their storage requirements, such as VM disk image deployment or on-the-fly software environment creation. Another important component is the capacity manager or scheduler. It governs the functionality provided by the core. The default capacity scheduler is a requirement/rank matchmaker. However, it is also possible to develop more complex scheduling policies, through a lease model and advance reservations . The last main components are the access drivers. They provide an abstraction of the underlying infrastructure to expose the basic functionality of the monitoring, storage, and virtualization services available in the cluster. Therefore, OpenNebula is not tied to any specific environment and can provide a uniform management layer regardless of the virtualization platform. Additionally, OpenNebula offers management interfaces to integrate the core's functionality within other data-center management tools, such as accounting or monitoring frameworks. To this end, OpenNebula implements the libvirt API, an open interface for VM management, as well as a command-line interface (CLI). A subset of this functionality is exposed to external users through a cloud interface. OpenNebula is able to adapt to organizations with changing resource needs, including addition or failure of physical resources. Some essential features to support changing environments are live migration and VM snapshots

Manjrasoft Aneka Cloud and Appliances

Aneka (www.manjrasoft.com/) is a cloud application platform developed by Manjrasoft, based in Melbourne, Australia. It is designed to support rapid development and deployment of parallel and distributed applications on private or public clouds. It provides a rich set of APIs for transparently exploiting distributed resources and expressing the business logic of applications by using preferred programming abstractions. System administrators can leverage a collection of tools to monitor and control the deployed infrastructure. It can be deployed on a public cloud such as Amazon EC2 accessible through the Internet to its subscribers, or a private cloud constituted by a set of nodes with restricted access. Aneka acts as a workload distribution and management platform for accelerating applications in both Linux and Microsoft .NET framework environments. Some of the key advantages of Aneka over other workload distribution solutions include:

• Support of multiple programming and application environments
• Simultaneous support of multiple runtime environments
• Rapid deployment tools and framework
• Ability to harness multiple virtual and/or physical machines for accelerating application provisioning based on users' Quality of Service/service-level agreement (QoS/SLA) requirements
• Built on top of the Microsoft .NET framework, with support for Linux environments through Mono Aneka offers three types of capabilities which are essential for building, accelerating, and managing clouds and their applications:

1. Build Aneka includes a new SDK which combines APIs and tools to enable users to rapidly develop applications. Aneka also allows users to build different runtime environments such as enterprise/private cloud by harnessing compute resources in network or enterprise data centers, Amazon EC2, and hybrid clouds by combining enterprise private clouds managed by Aneka with resources from Amazon EC2 or other enterprise clouds built and managed using XenServer.
2. Accelerate Aneka supports rapid development and deployment of applications in multiple runtime environments running different operating systems such as Windows or Linux/UNIX. Aneka uses physical machines as much as possible to achieve maximum utilization in local environments. Whenever users set QoS parameters such as deadlines, and if the enterprise resources are insufficient to meet the deadline, Aneka supports dynamic leasing of extra capabilities from public clouds such as EC2 to complete the task within the deadline
3. Manage Management tools and capabilities supported by Aneka include a GUI and APIs to set up, monitor, manage, and maintain remote and global Aneka compute clouds. Aneka also has an accounting mechanism and manages priorities and scalability based on SLA/QoS which enables dynamic provisioning. Here are three important programming models supported by Aneka for both cloud and traditional parallel applications:

1. Thread programming model, best solution to adopt for leveraging the computing capabilities of multicore nodes in a cloud of computers
2. Task programming model, which allows for quickly prototyping and implementing an independent bag of task applications
3. MapReduce programming model.