Distributed Cloud Computing and Parallel Processing -Part 1

Reference: Distributed and Cloud Computing From Parallel Processing to the Internet of Things, Kai Hwang

Geoffrey C. Fox, and Jack J. Dongarra, Morgan Kaufmann

Scalable Computing Over the Internet

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

The Age of Internet Computing

- Billions of people use the Internet every day.
- As a result, supercomputer sites and large data centers must provide high-performance computing services to huge numbers of Internet users **concurrently**.
- Because of this high demand, the **Linpack Benchmark** for ***high-performance computing*** (HPC) applications is no longer optimal for measuring system performance.
- The emergence of computing clouds instead demands ***high-throughput computing*** (HTC) systems built with parallel and distributed computing technologies.

The Platform Evolution

- 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 I*nternet* clouds has proliferated.
- These systems are employed by both consumers and high-end web-scale computing and information services.

The Platform Evolution

- 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 computing nodes that are physically connected in close range to one another.

The Platform Evolution

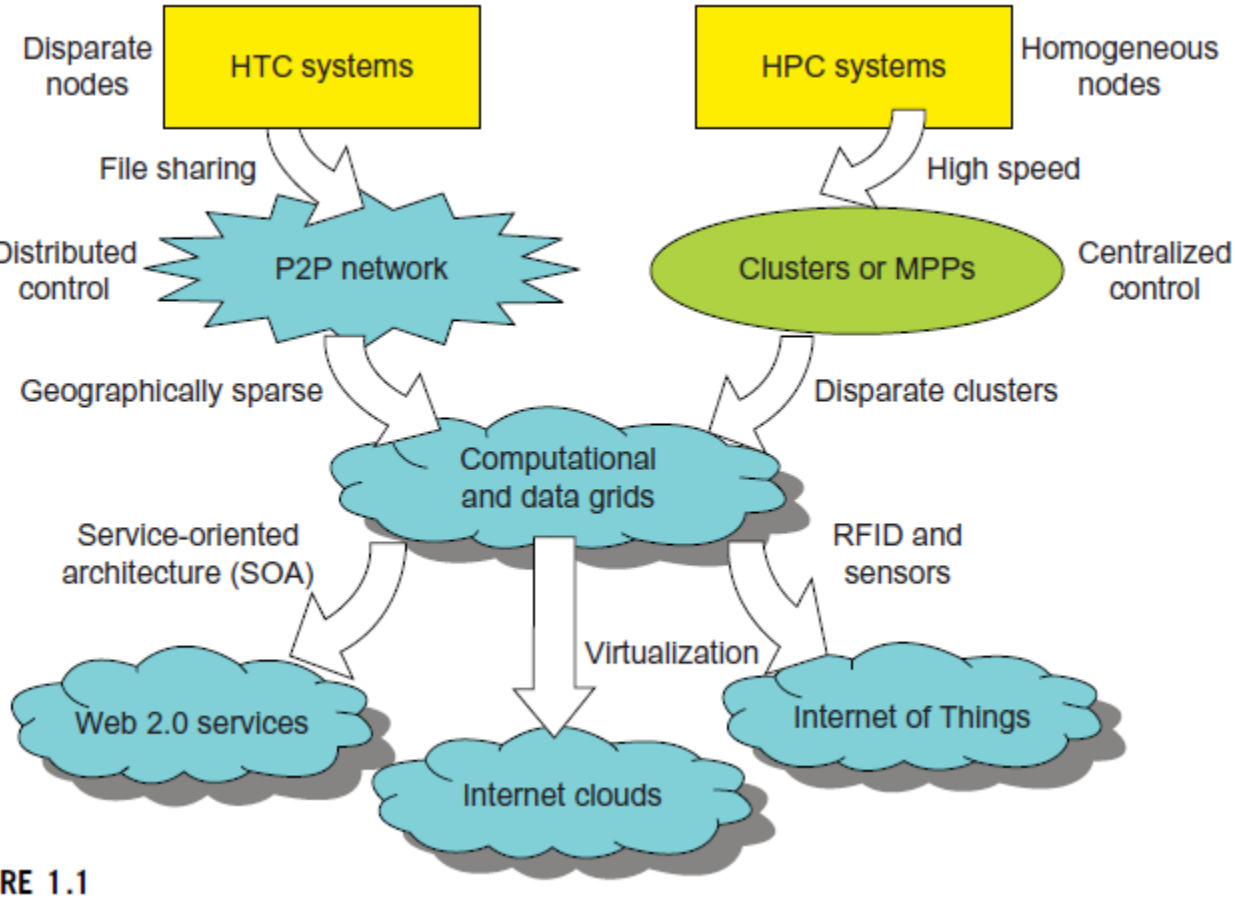

FIGURE 1.1

The Platform Evolution

- 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**.

High-Performance Computing

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

High-Throughput Computing

- The development of market-oriented high-end computing systems is undergoing a strategic change from an **HPC** paradigm to an **HTC** paradigm.
- This **HTC paradigm** pays more attention to **high-flux computing**. *The main application for high-flux computing is in Internet searches and web services by millions or more users simultaneously.*
  - **The performance goal thus shifts to measure high throughput or the number of tasks completed per unit of time.**
- **HTC technology** needs to not only improve in terms of *batch processing speed*, but also address the acute problems of *cost*, *energy savings*, *security*, and *reliability* at many data and enterprise computing centers.

Computing Paradigm Distinctions

- The **high-technology** community has argued for many years about the precise definitions of *centralized computing*, *parallel computing*, *distributed computing*, and *cloud computing*.
- In general, *distributed computing is the opposite of centralized computing.*
- The field of **parallel computing** overlaps with **distributed computing** to a great extent, and **cloud computing** overlaps with **distributed**, **centralized**, and **parallel computing**.

Computing Paradigm Distinctions

- **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.

Computing Paradigm Distinctions

- **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**.

Computing Paradigm Distinctions

- **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.

Computing Paradigm Distinctions

- **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.

Computing Paradigm Distinctions

- As an alternative to the preceding terms, some in the high-tech community prefer the term **concurrent computing** or **concurrent programming**.
- These terms typically refer to the **union of parallel computing and distributing computing**, although biased practitioners may interpret them differently.
- Ubiquitous computing refers to computing with pervasive devices at any place and time using wired or wireless communication.

Computing Paradigm Distinctions

- The **Internet of Things (IoT)** is a networked connection of everyday objects including computers, sensors, humans, etc.
- The **IoT** is supported by **Internet clouds** to achieve ubiquitous computing with any object at any place and time.
- Finally, the term **Internet computing** is even broader and covers all computing paradigms over the Internet.

Computing Paradigm Distinctions

- In the future, both HPC and HTC systems will demand **multicore** or **many-core** processors that can handle large numbers of computing threads per core.
- Both HPC and HTC systems emphasize **parallelism** and **distributed** computing.
- Future HPC and HTC systems must be able to satisfy this huge demand in computing power in terms of *throughput*, *efficiency, scalability*, and *reliability*.
- The **system efficiency** is decided by *speed*, *programming*, and *energy factors* (i.e., throughput per watt of energy consumed).

Computing Paradigm Distinctions

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

Scalable Computing Trends and New Paradigms

- **Degrees of Parallelism (DoP):**
  - Fifty years ago, when hardware was bulky and expensive, most computers were designed in a **bit-serial** fashion. In this scenario, ***bit-level parallelism*** (**BLP**) converts bit-serial processing to **word-level processing** gradually.
  - Over the years, users graduated from 4-bit microprocessors to 8-, 16-, 32-, and 64-bit CPUs.
  - This led us to the next wave of improvement, known as ***instruction-level parallelism*** (**ILP**), in which the processor executes multiple instructions simultaneously rather than only one instruction at a time.
  - For the past 30 years, we have practiced **ILP** through *pipelining*, *superscalar computing*, **VLIW** (very long instruction word) architectures, and *multithreading*.

Scalable Computing Trends and New Paradigms

- **ILP** requires *branch prediction*, *dynamic scheduling*, *speculation,* and *compiler support* to work efficiently.
- **Data-level parallelism** (DLP) was made popular through **SIMD** (single instruction, multiple data) and **vector machines** using vector or array types of instructions.
- **DLP** requires even more hardware support and compiler assistance to work properly.
- Ever since the introduction of **multicore** processors and **chip multiprocessors** (**CMP**s), we have been exploring **task-level parallelism** (TLP).

Scalable Computing Trends and New Paradigms

- A modern processor explores all of the aforementioned parallelism types. In fact, **BLP**, **ILP**, and **DLP** are well supported by advances in **hardware** and **compilers**.
- However, TLP is far from being very successful due to difficulty in programming and compilation of code for efficient execution on **multicore CMPs**.
- As we move from parallel processing to distributed processing, we will see an increase in computing granularity to **job-level parallelism** (**JLP**).
- It is fair to say that **coarse-grain** parallelism is built on top of **fine-grain** parallelism.

The Trend toward Utility Computing

- **Figure 1.2** identifies major computing paradigms to facilitate the study of **distributed systems** and their applications.
- These paradigms share some common characteristics. First, they are all ubiquitous in daily life.
- **Reliability** and **scalability** are two major design objectives in these computing models.
- Second, they are aimed at autonomic operations that can be self-organized to support dynamic discovery.
- Finally, these paradigms are composable with **QoS** and **SLA**s (service-level agreements).
- These paradigms and their attributes realize the computer utility vision.

The Trend toward Utility Computing

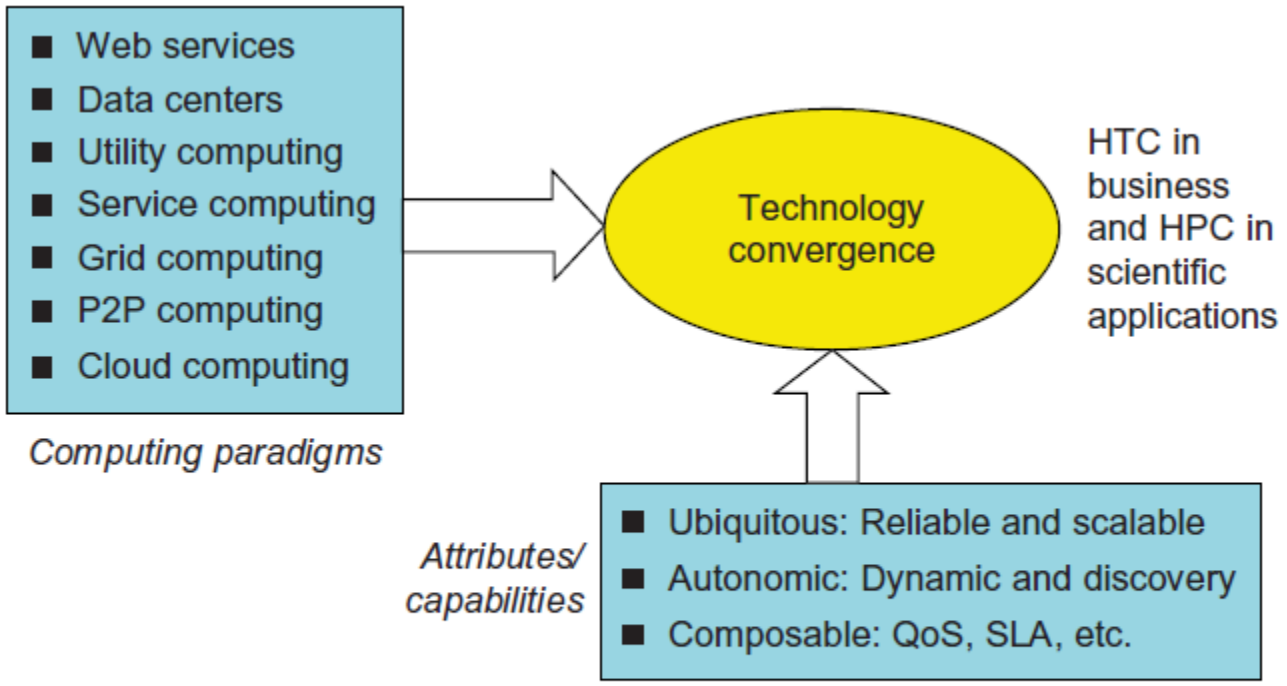

FIGURE 1.2

The vision of computer utilities in modern distributed computing systems.

Multicore CPUs and Multithreading Technologies

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

Multicore CPUs and Multithreading Technologies

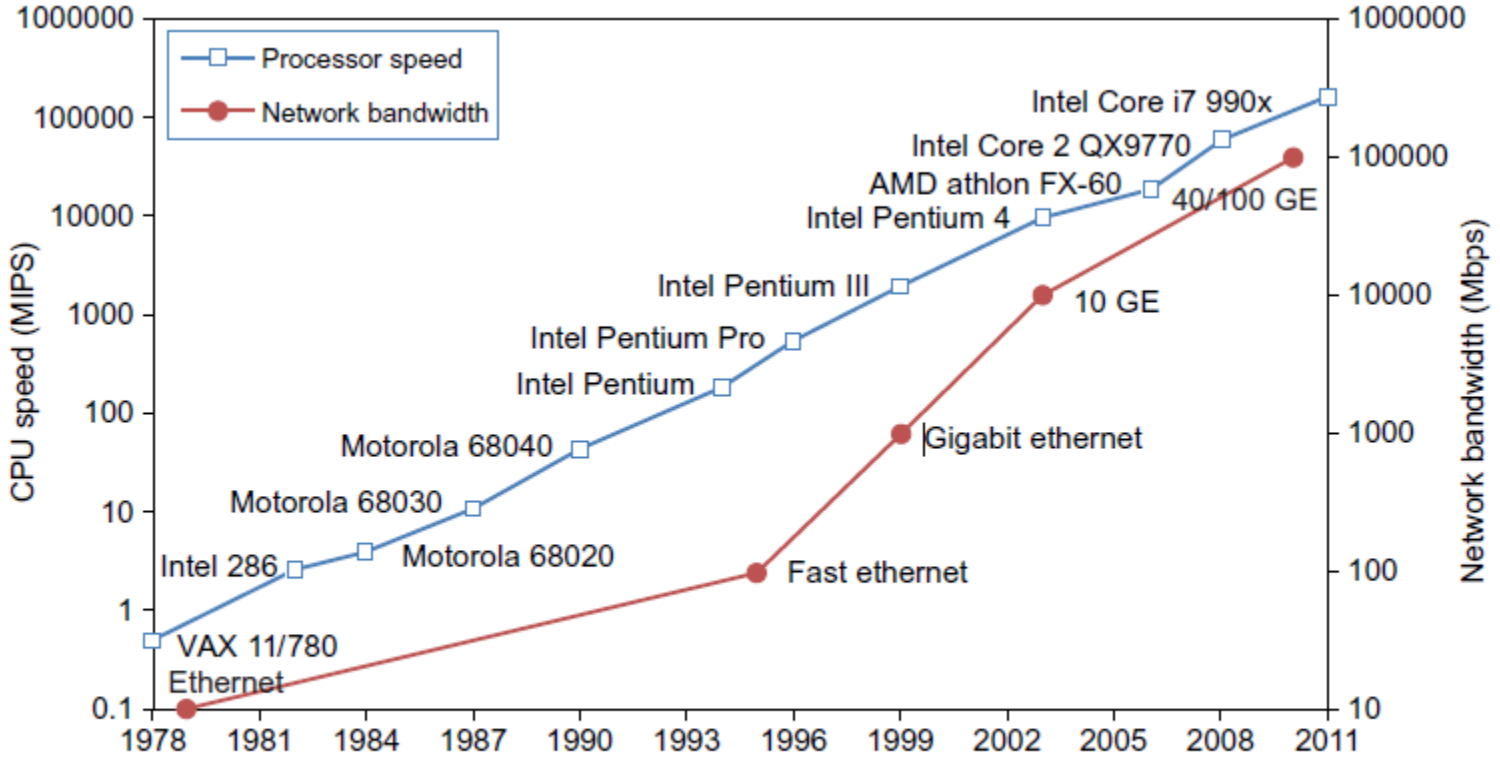

FIGURE 1.4

Improvement in processor and network technologies over 33 years.

Multicore CPUs and Multithreading Technologies

- Both **multi-core CPU** and **many-core GPU** processors can handle multiple instruction threads at different magnitudes today.
- **Figure 1.5** shows the architecture of a typical **multicore processor**.
- Each core is essentially a processor with its own private cache (**L1 cache**).
- Multiple cores are housed in the same chip with an L2 cache that is shared by all cores.
- In the future, multiple CMPs could be built on the same CPU chip with even the L3 cache on the chip.
- Multicore and multithreaded CPUs are equipped with many high-end processors, including the Intel i7, Xeon, AMD Opteron, Sun Niagara, IBM Power 6, and X cell processors.
- Each core could be also multithreaded. For example, the Niagara II is built with eight cores with eight threads handled by each core.

Multicore CPUs and Multithreading Technologies

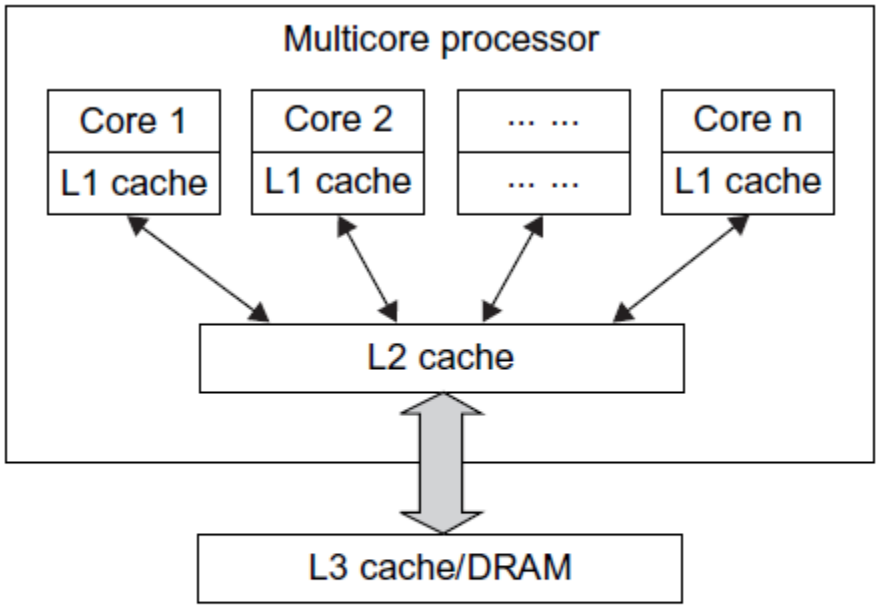

FIGURE 1.5

Schematic of a modern multicore CPU chip using a hierarchy of caches, where L1 cache is private to each core, on-chip L2 cache is shared and L3 cache or DRAM Is off the chip.

Multicore CPU and Many-Core GPU Architectures

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

Multicore CPU and Many-Core GPU Architectures

- Many RISC processors have been replaced with multicore x-86 processors and many-core GPUs in the Top 500 systems.
- This trend indicates that x-86 upgrades will dominate in data centers and supercomputers.
- The GPU also has been applied in large clusters to build supercomputers in MPPs.
- In the future, the processor industry is also keen to develop asymmetric or heterogeneous chip multiprocessors that can house both fat CPU cores and thin GPU cores on the same chip.

Multithreading Technology

- Consider in **Figure 1.6** the dispatch of **five independent threads** of instructions to four pipelined data paths (functional units) in each of the following five processor categories, from left to right: a four-issue *superscalar processor*, a *fine-grain multithreaded processor, a coarse-grain multithreaded processor*, a *two-core* CMP, and a *simultaneous multithreaded* (SMT) processor.
- The superscalar processor is single-threaded with four functional units.
- Each of the three multithreaded processors is four-way multithreaded over four functional data paths.
- In the dual-core processor, assume two processing cores, each a single-threaded two-way superscalar processor.

Multithreading Technology

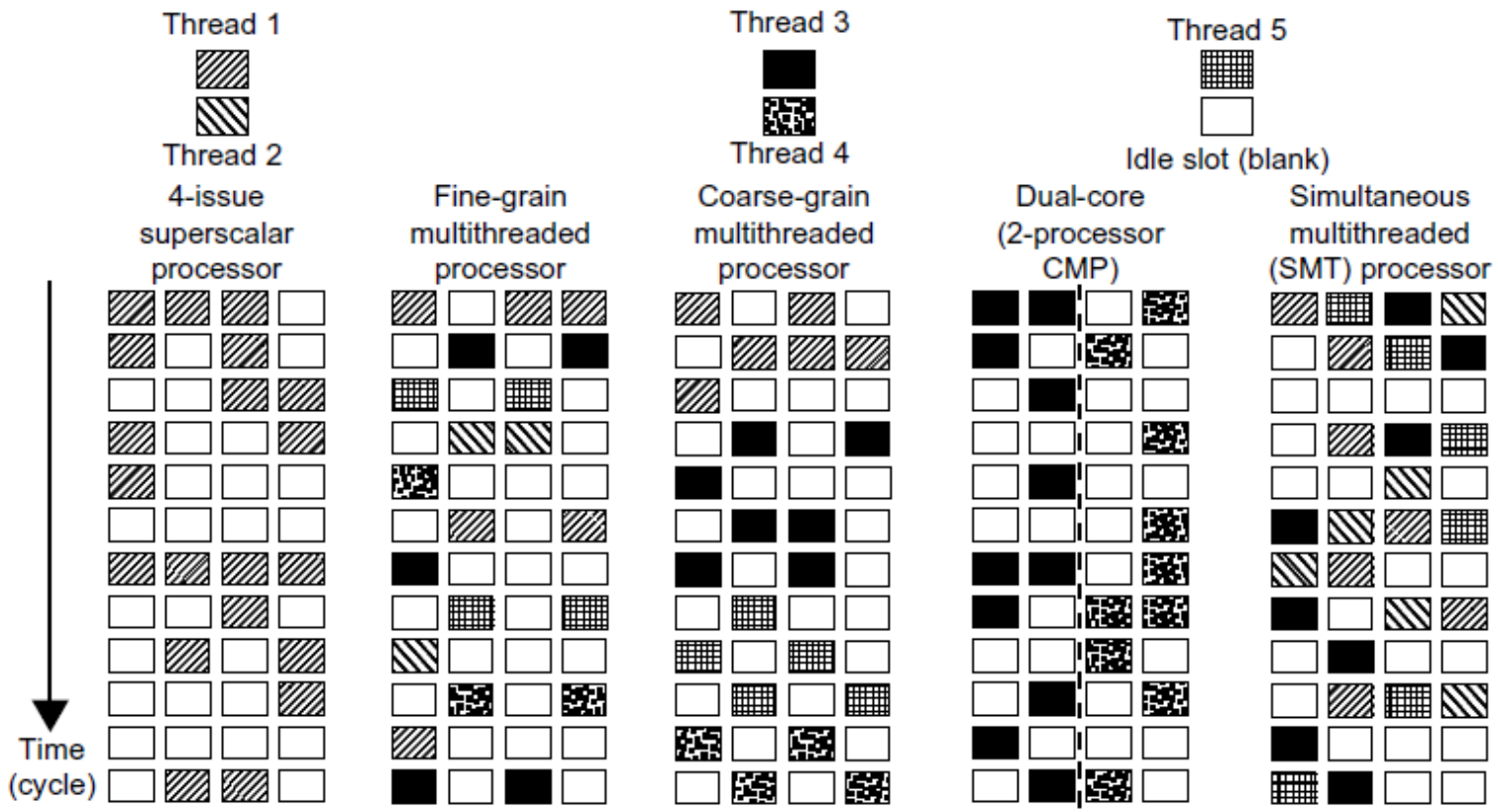

FIGURE 1.6

Five micro-architectures in modern CPU processors, that exploit ILP and TLP supported by multicore and multithreading technologies.

Multithreading Technology

- Instructions from different threads are distinguished by specific shading patterns for instructions from five independent threads.
- Typical instruction scheduling patterns are shown here. Only instructions from the same thread are executed in a superscalar processor.
- Fine-grain multithreading switches the execution of instructions from different threads per cycle.
- Course-grain multithreading executes many instructions from the same thread for quite a few cycles before switching to another thread. The multicore CMP executes instructions from different threads completely.
- The SMT allows simultaneous scheduling of instructions from different threads in the same cycle.

GPU Computing to Exa-scale and Beyond

- A **GPU** is a graphics coprocessor or accelerator mounted on a computer's graphics card or video card.
- A GPU offloads the CPU from tedious graphics tasks in video editing applications. The world's first GPU, the GeForce 256, was marketed by NVIDIA in 1999.
- These GPU chips can process a minimum of 10 million polygons per second, and are used in nearly every computer on the market today.
- Some GPU features were also integrated into certain CPUs.
- Traditional CPUs are structured with only a few cores. For example, the Xeon X5670 CPU has six cores.
- However, **a modern GPU chip can be built with hundreds of processing cores**.

GPU Computing to Exa-scale and Beyond

- Unlike CPUs, **GPU**s have a throughput architecture that exploits **massive parallelism** by executing many **concurrent threads;**
  - slowly, instead of executing a single long thread in a conventional microprocessor very quickly.
- Lately, parallel GPUs or GPU clusters have been garnering a lot of attention against the use of CPUs with limited parallelism.
  - General-purpose computing on GPUs, known as **GPGPU**s, have appeared in the HPC field.
  - NVIDIA's **CUDA** model was for HPC using GPGPUs.

How GPUs Work?

- Early GPUs functioned as coprocessors attached to the CPU.
- Today, the **NVIDIA GPU** has been upgraded to **128 cores** on a single chip.
  - Furthermore, **each core on a GPU can handle eight threads of instructions**.
  - This translates to having **up to 1,024 threads executed concurrently on a single GPU.**
- This is **true massive parallelism**, compared to only a few threads that can be handled by a conventional CPU.
- The CPU is optimized for **latency caches**, while the GPU is optimized to deliver much higher throughput with explicit management of **on-chip memory**.

How GPUs Work?

- Modern GPUs are not restricted to accelerated graphics or video coding.
- They are used in **HPC** systems to power supercomputers with **massive parallelism** at **multicore** and **multithreading** levels.
- **GPU**s are designed to handle large numbers of **floating-point operations in parallel.**
  - In a way, the GPU offloads the CPU from all data-intensive calculations, not just those that are related to video processing.
  - Conventional GPUs are widely used in mobile phones, game consoles, embedded systems, PCs, and servers.
- The NVIDIA CUDA Tesla or Fermi is used in GPU clusters or in HPC systems for parallel processing of massive floating-pointing data.

GPU Programming Model

- **Figure 1.7** shows the interaction between a CPU and GPU in performing parallel execution of floating-point operations concurrently.
- **The CPU is the conventional multicore processor with limited parallelism to exploit**.
- **The GPU has a many-core architecture that has hundreds of simple processing cores organized as multiprocessors. Each core can have one or more threads.**
- Essentially, the CPU's floating-point kernel computation role is largely offloaded to the many-core GPU.
- The CPU instructs the GPU to perform massive data processing.
- The bandwidth must be matched between the on-board main memory and the on-chip GPU memory.
  - This process is carried out in NVIDIA's CUDA programming using the GeForce 8800 or Tesla and Fermi GPUs.

GPU Programming Model

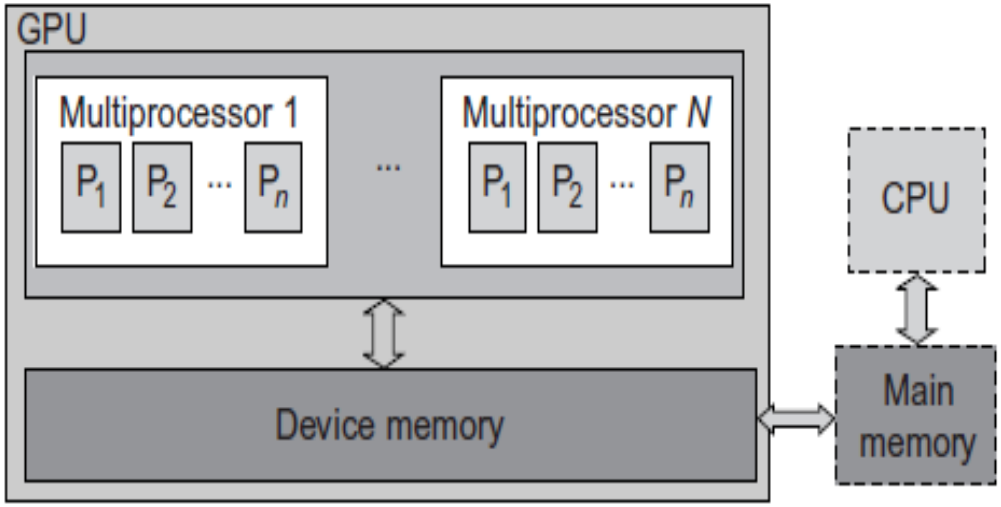

FIGURE 1.7

The use of a GPU along with a CPU for massively parallel execution in hundreds or thousands of processing cores.

System-Area Interconnects

- The nodes in small clusters are mostly interconnected by an **Ethernet switch** or **a local area network** (LAN).
- As **Figure 1.11** shows, a LAN typically is used to connect client hosts to big servers.
- A **storage area network** (**SAN**) connects servers to network storage such as disk arrays.
- **Network attached storage** (**NAS**) connects client hosts directly to the disk arrays.
- All three types of networks often appear in a **large cluster built** with commercial network components.
- If no large distributed storage is shared, a small cluster could be built with a multiport Gigabit Ethernet switch plus copper cables to link the end machines.
  - All three types of networks are commercially available.

System-Area Interconnects

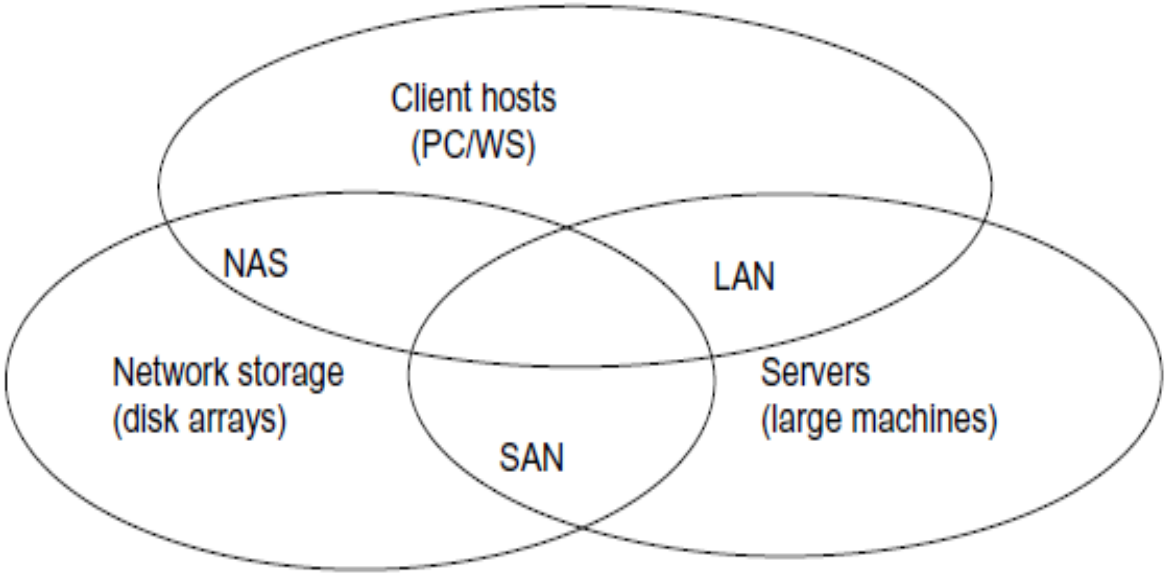

FIGURE 1.11

Three interconnection networks for connecting servers, client hosts, and storage devices; the LAN connects client hosts and servers, the SAN connects servers with disk arrays, and the NAS connects clients with large storage systems in the network environment.

Virtual Machines and Virtualization Middleware

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

Virtual Machines and Virtualization Middleware

- Today, to build large clusters, grids, and clouds, we need to access large amounts of computing, storage, and networking resources in a **virtualized** manner.
- We need to aggregate those resources, and hopefully, offer a single system image.
- In particular, a **cloud of provisioned resources** must rely on *virtualization of processors*, *memory*, and I/O facilities dynamically.
- **Figure 1.12** illustrates the architectures of **three VM** configurations.

Virtual Machines and Virtualization Middleware

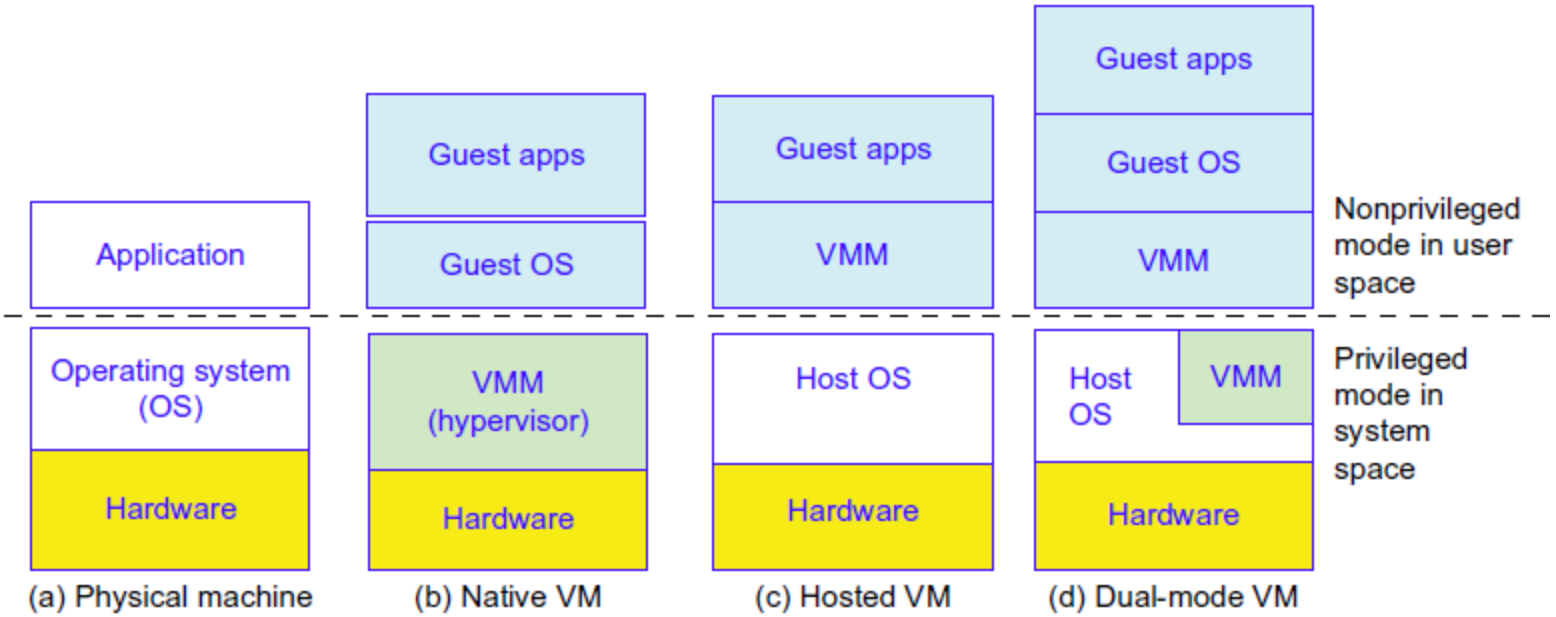

FIGURE 1.12

Three VM architectures in (b), (c), and (d), compared with the traditional physical machine shown in (a).

VM Primitive Operations

- The **VMM** provides the VM abstraction to the guest OS. With full virtualization, the VMM exports a VM abstraction identical to the physical machine so that a standard OS such as Windows 2000 or Linux can run just as it would on the physical hardware.
- Low-level VMM operations are indicated by Mendel Rosenblum [41] and illustrated in **Figure 1.13**.
  - **First**, the VMs can be multiplexed between hardware machines, as shown in Figure 1.13(a).
  - **Second**, a VM can be suspended and stored in stable storage, as shown in Figure 1.13(b).
  - T**hird,** a suspended VM can be resumed or provisioned to a new hardware platform, as shown in Figure 1.13(c).
  - **Finally**, a VM can be migrated from one hardware platform to another, as shown in Figure 1.13(d).

VM Primitive Operations

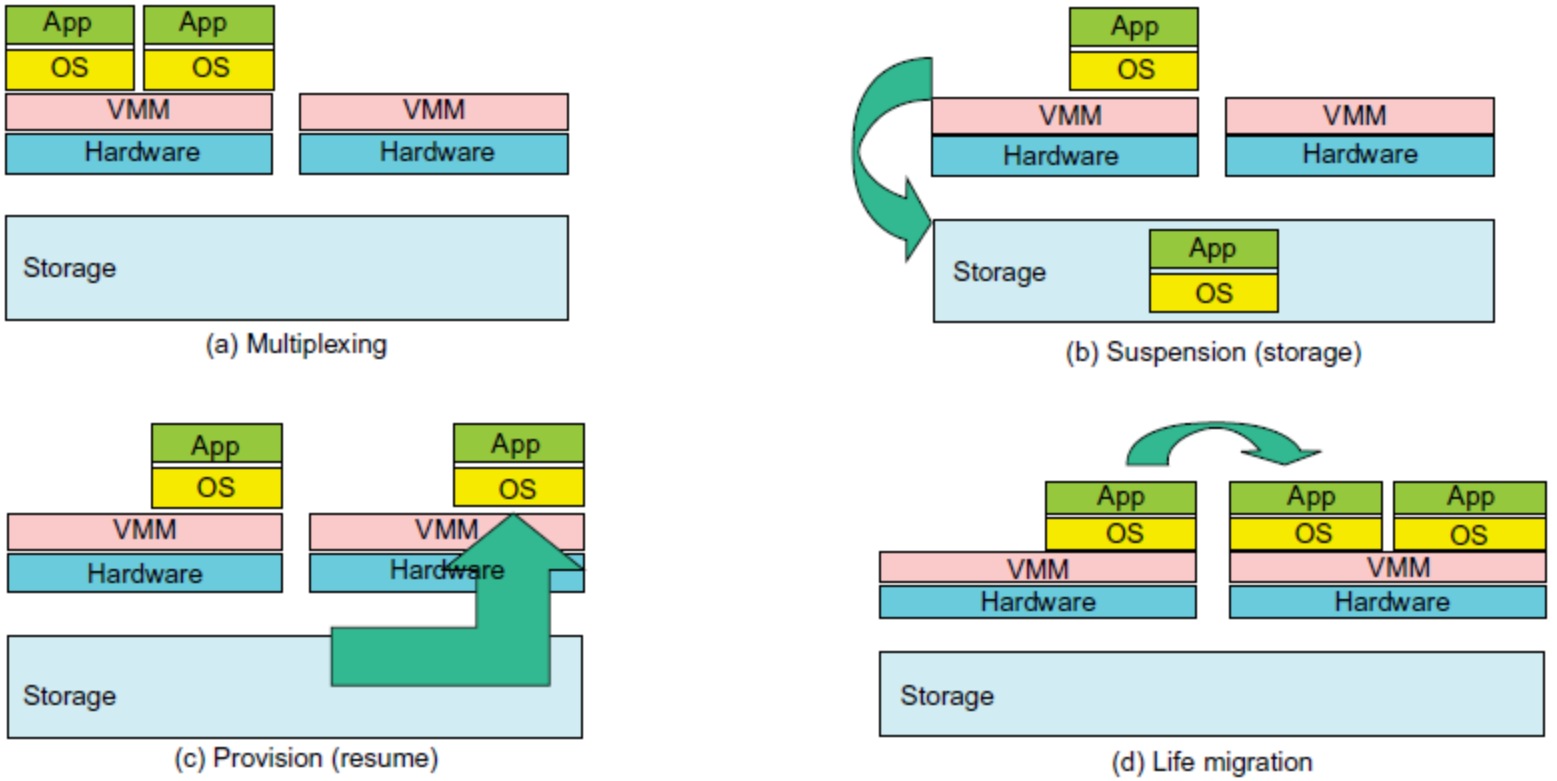

FIGURE 1.13

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

VM Primitive Operations

- These VM operations enable a VM to be provisioned to any available hardware platform.
- They also enable flexibility in porting distributed application executions.
- Furthermore, the VM approach will significantly enhance the utilization of server resources.
- Multiple server functions can be consolidated on the same hardware platform to achieve higher system efficiency.
- This will eliminate server sprawl via deployment of systems as VMs, which move transparency to the shared hardware.
- With this approach, **VMware** claimed that server utilization could be increased from its current 5–15 percent to 60–80 percent.

System Models for Distributed and Cloud Computing

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

System Models for Distributed and Cloud Computing

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

Table 1.2 Classification of Parallel and Distributed Computing Systems

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

Cluster Architecture

- A **computing cluster** consists of interconnected stand-alone computers which work cooperatively as a single integrated computing resource.
- In the past, clustered computer systems have demonstrated impressive results in handling heavy workloads with large data sets.
- **Figure 1.15** shows the architecture of a typical server cluster built around a low-latency, high bandwidth interconnection network.
  - This network can be as simple as a SAN (e.g., Myrinet) or a LAN (e.g., Ethernet). To build a larger cluster with more nodes, the interconnection network can be built with multiple levels of Gigabit Ethernet, Myrinet, or InfiniBand switches.

Cluster Architecture

- 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.

Cluster Architecture

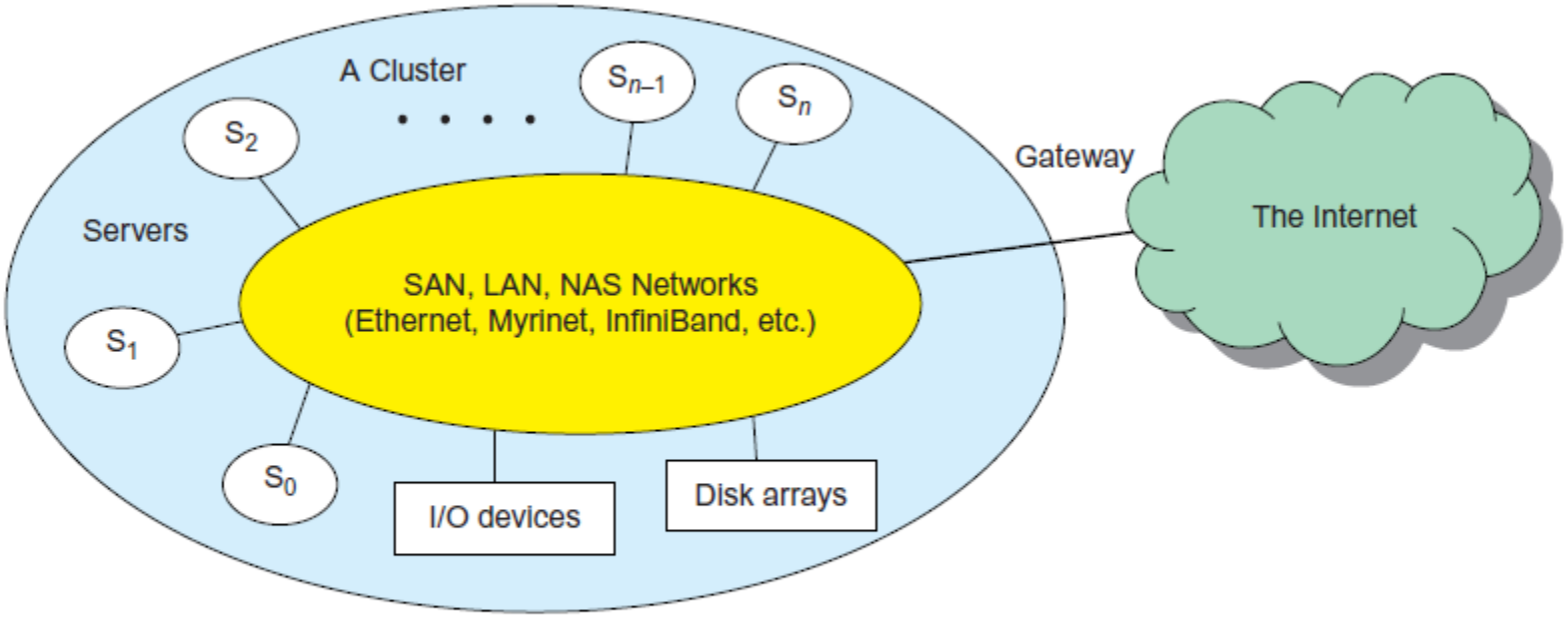

FIGURE 1.15

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.

Peer-to-Peer Network Families

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

Peer-to-Peer Network Families

- 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 neede**d.
- In other words, no peer machine has a global view of the entire P2P system.
  - The system is self-organizing with distributed control.

Peer-to-Peer Network Families

- **Figure 1.17** 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.

Peer-to-Peer Network Families

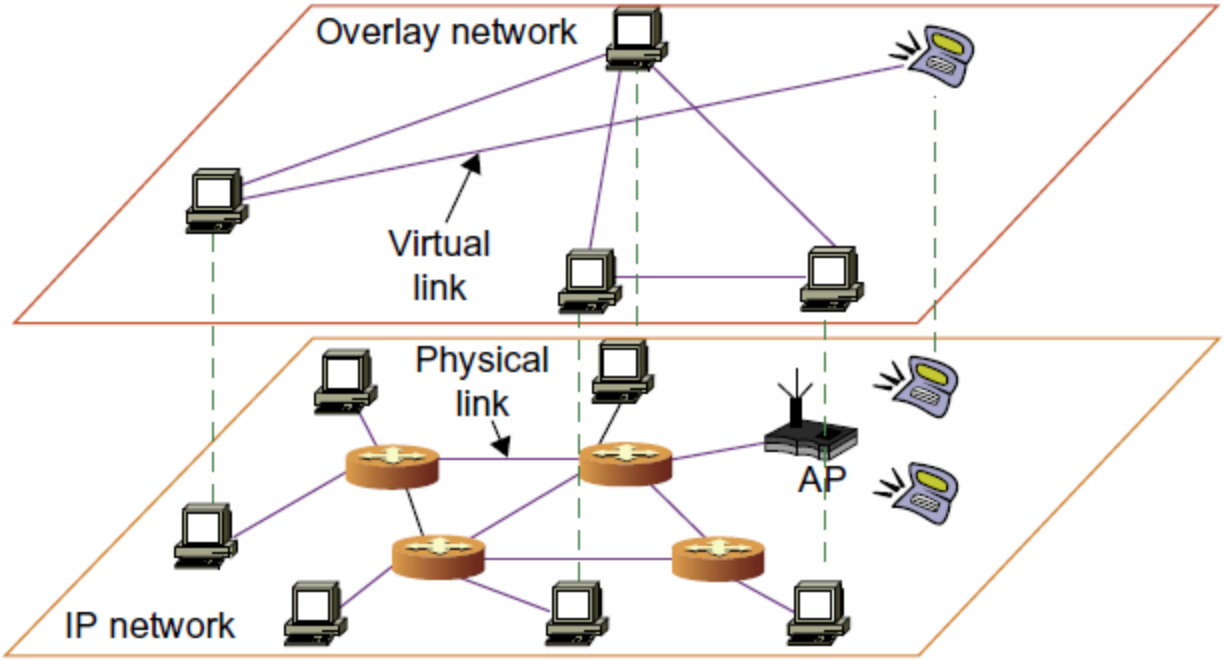

FIGURE 1.17

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

Cloud Computing over the Internet

- **Cloud computing** has been defined differently by many users and designers.
  - For example, IBM, a major player in cloud computing, has defined it as follows: “A cloud is a pool of virtualized computer resources”.
- A cloud can host a variety of different workloads, including batch-style backend jobs and interactive and user-facing applications.”
  - Based on this definition, a cloud allows workloads to be deployed and scaled out quickly through rapid provisioning of virtual or physical machines.

Cloud Computing over the Internet

- The cloud supports redundant, self-recovering, highly scalable programming models that allow workloads to recover from many unavoidable hardware/software failures.
- Finally, the cloud system should be able to monitor resource use in real time to enable rebalancing of allocations when needed.
- **Cloud computing** applies a virtualized platform with elastic resources on demand by provisioning hardware, software, and data sets dynamically (see **Figure 1.18**).

Internet Clouds

- **The idea is to move desktop computing to a service-oriented platform using server clusters and huge databases at data centers.**
  - Cloud computing leverages its low cost and simplicity to benefit both users and providers.
- **Machine virtualization** has enabled such cost-effectiveness. Cloud computing intends to satisfy many user applications simultaneously. The cloud ecosystem must be designed to be secure, trustworthy, and dependable.
  - Some computer users think of the cloud as a centralized resource pool.
  - Others consider the cloud to be a server cluster which practices distributed computing over all the servers used.

Internet Clouds

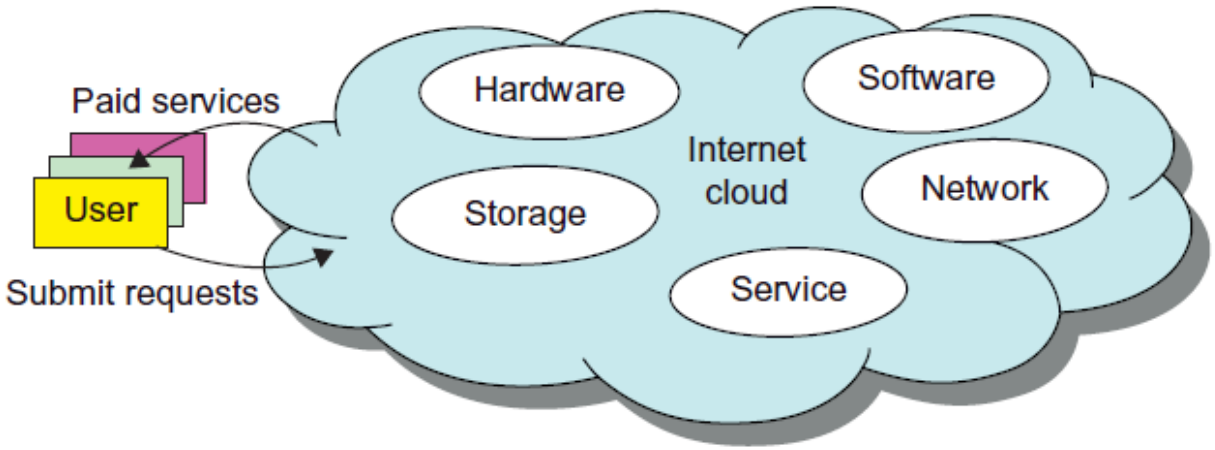

FIGURE 1.18

Virtualized resources from data centers to form an Internet cloud, provisioned with hardware, software, storage, network, and services for paid users to run their applications.

The Cloud Landscape

- Traditionally, a **distributed computing system** tends to be owned and operated by an autonomous administrative domain (e.g., a research laboratory or company) for on-premises computing needs.
- However, these traditional systems have encountered several performance bottlenecks:
  - *constant system maintenance*, *poor utilization,* and *increasing costs associated with hardware/software upgrades*.
- **Cloud computing** as an on-demand computing paradigm resolves or relieves us from these problems.
- **Figure 1.19** depicts the cloud landscape and major cloud players, based on three cloud service models.

The Cloud Landscape

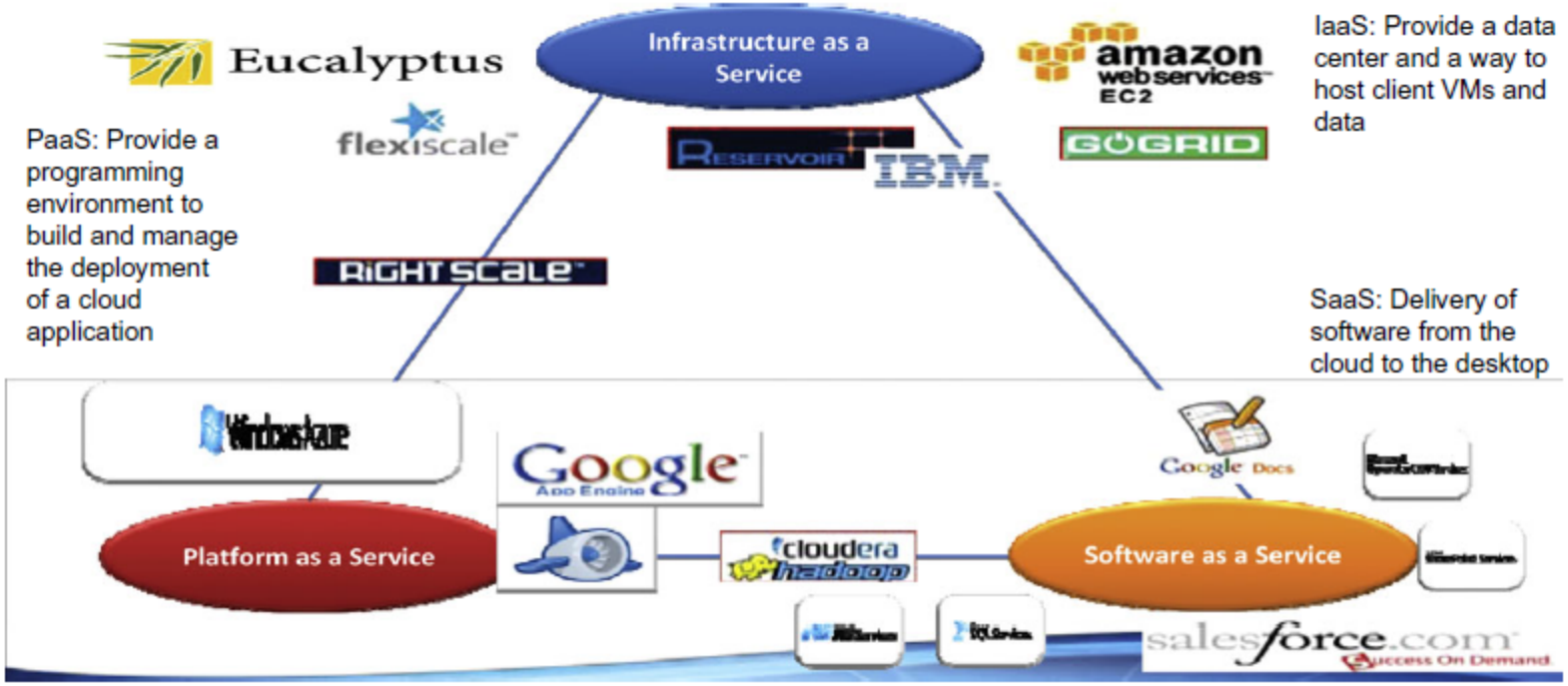

FIGURE 1.19

Three cloud service models in a cloud landscape of major providers.

The Cloud Landscape

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

The Cloud Landscape

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

The Cloud Landscape

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

The Cloud Landscape

- **Internet clouds** offer four deployment modes: *private*, *public*, *managed*, and *hybrid*.
- These modes demand different levels of security implications.
  - The different SLAs imply that the security responsibility is shared among all the cloud providers, the cloud resource consumers, and the third party cloud-enabled software providers.
- Advantages of cloud computing have been advocated by many IT experts, industry leaders, and computer science researchers.

The Cloud Landscape

- The following list highlights eight reasons to adapt the cloud for upgraded Internet applications and web services:
  1. Desired location in areas with protected space and higher energy efficiency.
  2. Sharing of peak-load capacity among a large pool of users, improving overall utilization.
  3. Separation of infrastructure maintenance duties from domain-specific application development.
  4. Significant reduction in cloud computing cost, compared with traditional computing paradigms.
  5. Cloud computing programming and application development.
  6. Service and data discovery and content/service distribution.
  7. Privacy, security, copyright, and reliability issues.
  8. Service agreements, business models, and pricing policies.

Software Environments for Distributed Systems and Clouds

- This section introduces popular software environments for using distributed and cloud computing systems:
  - Service-Oriented Architecture (SOA)
  - Layered Architecture for Web Services and Grids
  - Web Services and Tools
  - The Evolution of SOA
  - Grids versus Clouds

Service-Oriented Architecture (SOA)

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

Service-Oriented Architecture (SOA)

- **SOA** starts with entity interfaces and inter-entity communication, which rebuild the top four OSI layers but at the entity and not the bit level. **Figure 1.20** shows the layered architecture for distributed entities used in web services and grid systems.

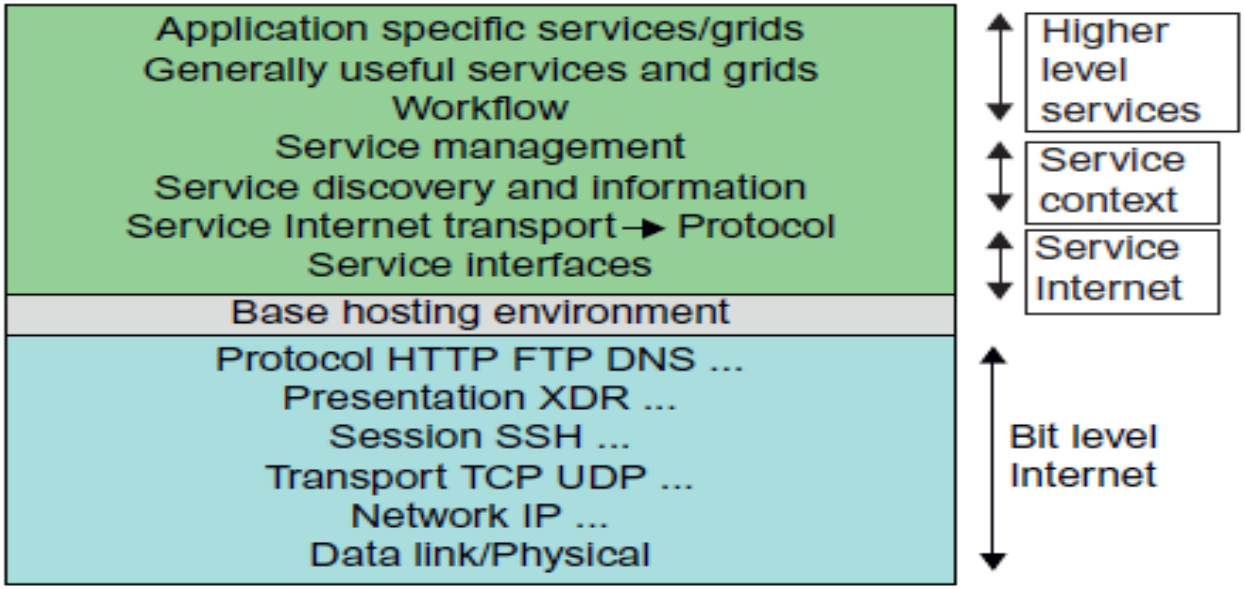

FIGURE 1.20

Layered achitecture for web services and the grids.

Layered Architecture for Web Services and Grids

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

Layered Architecture for Web Services and Grids

- In the case of **fault tolerance**, the features in the ***Web Services Reliable Messaging*** (WSRM) framework mimic the OSI layer capability (as in TCP fault tolerance) modified to match the different abstractions (such as messages versus packets, virtualized addressing) at the entity levels.
- Security is a critical capability that either uses or re-implements the capabilities seen in concepts such as ***Internet Protocol Security*** (IPsec) and secure sockets in the OSI layers.

Web Services and Tools

- **Loose coupling** and support of heterogeneous implementations make Web services more attractive than distributed objects.
- **Figure 1.20** corresponds to two choices of service architecture: web services or REST systems.
- Both web services and REST systems have very distinct approaches to building reliable interoperable systems.
- In web services, one aims to fully specify all aspects of the service and its environment.
- This specification is carried with communicated messages using *Simple Object Access Protocol* (SOAP).

Web Services and Tools

- The hosting environment then becomes a universal **distributed operating system** with fully distributed capability carried by **SOAP** messages.
  - This approach has mixed success as it has been hard to agree on key parts of the protocol and even harder to efficiently implement the protocol by software such as Apache Axis.
- In the REST approach, one adopts simplicity as the universal principle and delegates most of the difficult problems to application (implementation-specific) software.

Web Services and Tools

- In a web services language, REST has minimal information in the header, and the message body (that is opaque to generic message processing) carries all the needed information.
  - REST architectures are clearly more appropriate for rapid technology environments.
- However, the ideas in web services are important and probably will be required in mature systems at a different level in the stack (as part of the application).
  - Note that REST can use XML schemas but not those that are part of SOAP; “XML over HTTP” is a popular design choice in this regard.

The Evolution of SOA

- As shown in **Figure 1.21**, ***service-oriented architecture*** (**SOA**) has evolved over the years.
  - SOA applies to building *grids*, *clouds*, *grids of clouds*, *clouds of grids*, *clouds of clouds* (also known as *interclouds)*, and *systems of systems* in general.
- A large number of sensors provide data-collection services, denoted in the figure as **SS** (**sensor service**).
  - A sensor can be a **ZigBee** device, a **Bluetooth device**, a WiFi access point, a personal computer, a GPA, or a wireless phone, among other things.

The Evolution of SOA

- **Raw data** is collected by sensor services.
  - All the SS devices interact with large or small computers, many forms of grids, databases, the compute cloud, the storage cloud, the filter cloud, the discovery cloud, and so on.
- **Filter services** (**fs** in the figure 1.21) are used to eliminate unwanted raw data, in order to respond to specific requests from the web, the grid, or web services.

The Evolution of SOA

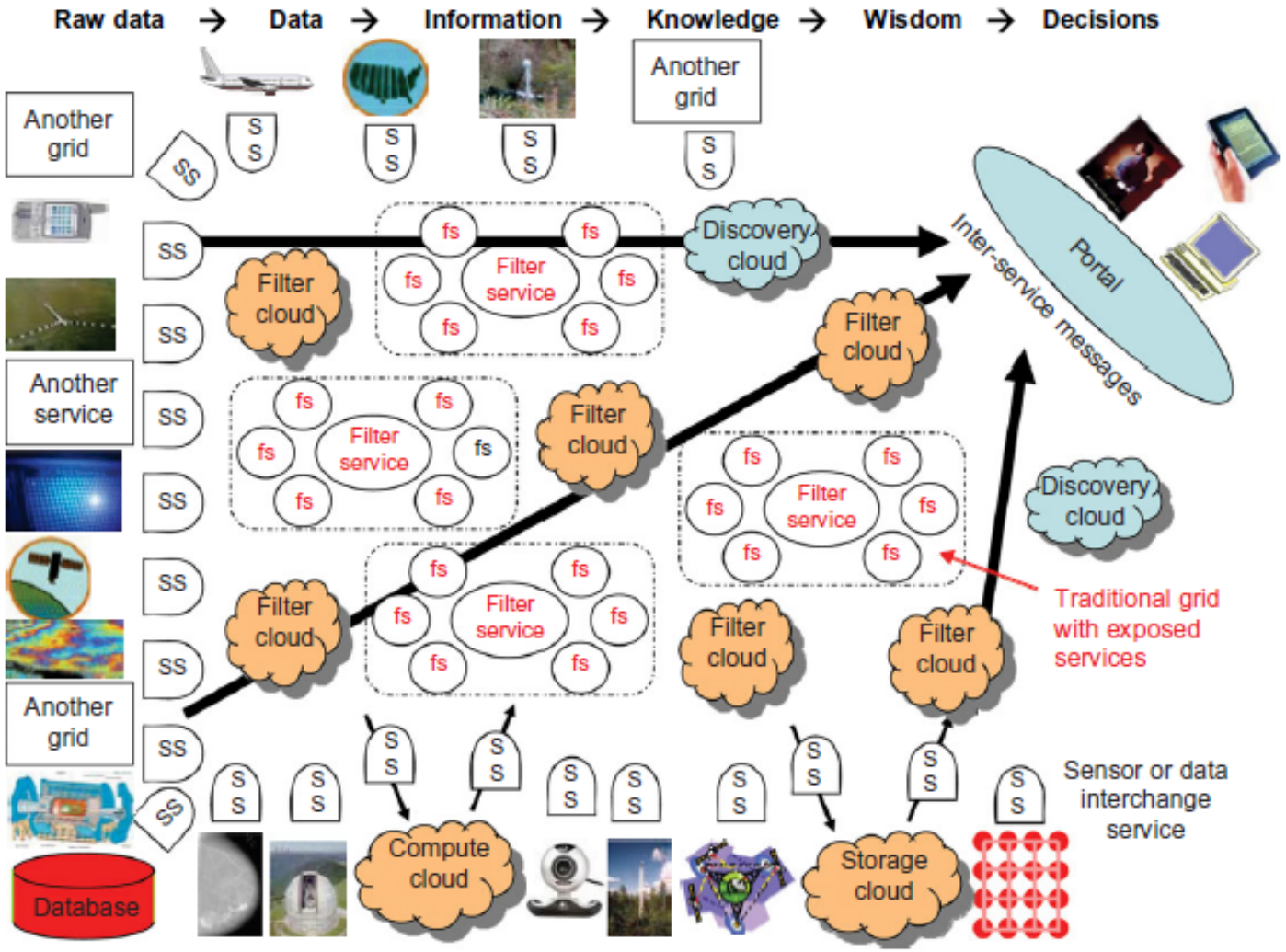

FIGURE 1.21

Grids versus Clouds

- The boundary between **grids** and **clouds** are getting blurred in recent years.
- For web services, workflow technologies are used to coordinate or orchestrate services with certain specifications used to define critical business process models such as two-phase transactions.
- In general, a grid system applies static resources, while a cloud emphasizes elastic resources.
- For some researchers, the differences between grids and clouds are limited only in dynamic resource allocation based on virtualization and autonomic computing.
- **One can build a grid out of multiple clouds**.
- This type of grid can do a better job than a pure cloud, because it can explicitly support negotiated resource allocation.
- Thus one may end up building with a system of systems: such as a cloud of clouds, a grid of clouds, or a cloud of grids, or inter-clouds as a basic SOA architecture.

Trends toward Distributed Operating Systems

- The computers in most **distributed systems** are **loosely coupled**.
  - This is mainly due to the fact that all node machines run with an **independent operating system**.
- To promote **resource sharing** and fast communication among node machines, it is best to have a **distributed OS** that manages all resources coherently and efficiently.
- Such a system is most likely to be a **closed system**, and it will likely rely on **message passing** and **RPC**s for internode communications.
  - It should be pointed out that a **distributed OS** is crucial for upgrading the performance, efficiency, and flexibility of distributed applications.

Features of 3 distributed OS

Distributed OS Functionality	AMOEBA Developed at Vrije University [46]	DCE as OSF/1 by Open Software Foundation [7]	MOSIX for Linux Clusters at Hebrew University [3]
History and Current System Status	Written in C and tested in the European community; version 5.2 released in 1995	Built as a user extension on top of UNIX, VMS, Windows, OS/2, etc.	Developed since 1977, now called MOSIX2 used in HPC Linux and GPU clusters
Distributed OS Architecture	Microkernel-based and location-transparent, uses many servers to handle files, directory, replication, run, boot, and TCP/IP services	Middleware OS providing a platform for running distributed applications; The system supports RPC, security, and threads	A distributed OS with resource discovery, process migration, runtime support, load balancing, flood control, configuration, etc.
OS Kernel, Middleware, and Virtualization Support	A special microkernel that handles low-level process, memory, I/O, and communication functions	DCE packages handle file,time, directory, security services, RPC, and authentication at middleware or user space	MOSIX2 runs with Linux 2.6; extensions for use in multiple clusters and clouds with provisioned VMs
Communication Mechanisms	Uses a network-layer FLIP protocol and RPC to implement point-to-point and group communication	RPC supports authenticated communication and other security services in user programs	Using PVM, MPI in collective communications, priority process control, and queuing services

Parallel and Distributed Programming Models

- In this section, we will explore four programming models for distributed computing with expected scalable performance and application flexibility.
- **Table 1.7** summarizes three of these models, along with some software tool sets developed in recent years.

Parallel and Distributed Programming Models

Table 1.7 Parallel and Distributed Programming Models and Tool Sets

Model	Description	Features
MPI	A library of subprograms that can be called from C or FORTRAN to write parallel programs running on distributed computer systems [6,28,42]	Specify synchronous or asynchronous point-to-point and collective communication commands and I/O operations in user programs for message-passing execution
MapReduce	A web programming model for scalable data processing on large clusters over large data sets, or in web search operations [16]	*Map* function generates a set of intermediate key/value pairs; *Reduce* function merges all intermediate values with the same key
Hadoop	A software library to write and run large user applications on vast data sets in business applications (http://hadoop.apache.org/core)	A scalable, economical, efficient, and reliable tool for providing users with easy access of commercial clusters

Message-Passing Interface (MPI)

- This is the primary programming standard used to develop **parallel** and **concurrent** programs to run on a **distributed system**.
  - **MPI** is essentially a library of subprograms that can be called from C or FORTRAN to write parallel programs running on a distributed system.
- The idea is to embody clusters, grid systems, and P2P systems with upgraded web services and utility computing applications.
  - Besides MPI, distributed programming can be also supported with low-level primitives such as the **Parallel Virtual Machine** (PVM).
  - Both MPI and PVM are described in Hwang and Xu.

MapReduce

- **MapReduce** is a web programming model for scalable data processing on large clusters over large data sets.
  - The model is applied mainly in web-scale search and cloud computing applications.
- The user specifies a **Map function** to generate a set of intermediate key/value pairs.
- Then the user applies a **Reduce function** to merge all intermediate values with the same intermediate key.

MapReduce

- **MapReduce** is highly scalable to explore high degrees of parallelism at different job levels.
- A typical MapReduce computation process can handle terabytes of data on tens of thousands or more client machines:
  - Hundreds of MapReduce programs can be executed simultaneously; in fact, thousands of MapReduce jobs are executed on Google's clusters every day.

Hadoop Library

- **Hadoop** offers a software platform that was originally developed by a **Yahoo! group.**
- The package enables users to write and run applications over vast amounts of **distributed data**.
- Users can easily scale Hadoop to store and process petabytes of data in the web space.
  - Also, Hadoop is economical in that it comes with an open source version of MapReduce that minimizes overhead in task spawning and massive data communication.
- It is efficient, as it processes data with a high degree of parallelism across a large number of commodity nodes, and it is reliable in that it automatically keeps multiple data copies to facilitate redeployment of computing tasks upon unexpected system failures.