NAVODAYA INSTITUTE OF TECHNOLOGY, RAICHUR

DEPARMENT OF COMPUTER SCIENCE & ENGINEERING

CLOUD COMPUTING BCS601

MODULE-1

DISTRIBUTED SYSTEM MODELS AND ENABLING TECHNOLOGIES

1.1 SCALABLE COMPUTING OVER THE INTERNET

Over the past 60 years, computing technology has undergone a series of platform and environment changes. In this section, we assess evolutionary changes in machine architecture, operating system platform, network connectivity, and application workload. Instead of using a centralized computer to solve computational problems, a parallel and distributed computing system uses multiple computers to solve large-scale problems over the Internet. Thus, distributed computing becomes data-intensive and network-centric. This section identifies the applications of modern computer systems that practice parallel and distributed computing. These large-scale Internet applications have significantly enhanced the quality of life and information services in society today.

1.1.1 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 distribu- ted computing technologies [5,6,19,25]. We have to upgrade data centers using fast servers, storage systems, and high-bandwidth networks. The purpose is to advance network-based computing and web services with the emerging new technologies.

1.1.1.1 The Platform Evolution

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

From 1970 to 1990, we saw widespread use of personal computers built with VLSI microproces-

sors. 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

Scalable Computing over the Internet

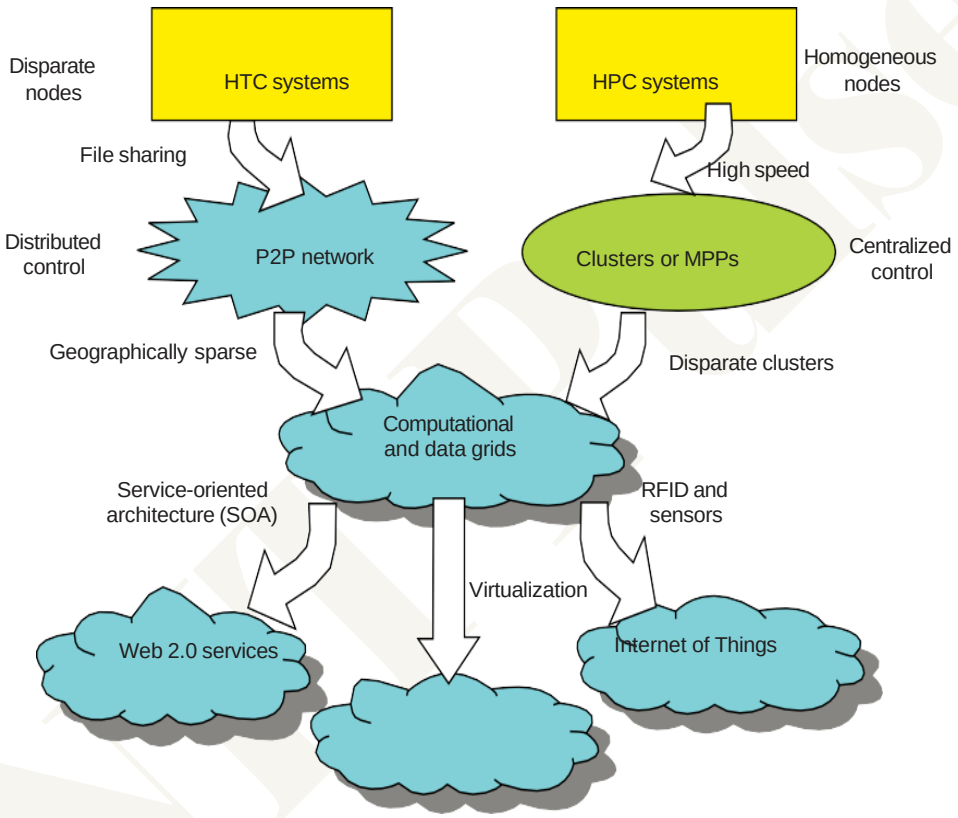

FIGURE 1.1

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

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. We will discuss clusters, MPPs, and grid systems in more detail in Chapters 2 and 7.

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 (a concept we will discuss further in Chapter 5). 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.

1.1.1.2 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.

1.1.1.3 High-Throughput Computing

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

1.1.1.4 Three New Computing Paradigms

As Figure 1.1 illustrates, with the introduction of SOA, Web 2.0 services become available. Advances in virtualization make it possible to see the growth of Internet clouds as a new computing paradigm. The maturity of *radio-frequency identification (RFID)*, *Global Positioning System (GPS)*, and sensor technologies has triggered the development of the *Internet of Things (IoT)*. These new paradigms are only briefly introduced here. We will study the details of SOA in Chapter 5; virtualization in Chapter 3; cloud computing in Chapters 4, 6, and 9; and the IoT along with cyber-physical systems (CPS) in Chapter 9.

When the Internet was introduced in 1969, Leonard Klienrock of UCLA declared: "As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of computer utilities, which like present electric and telephone utilities, will service individual homes and offices across the country." Many people have redefined the term "computer" since that time. In 1984, John Gage of Sun Microsystems created the slogan, "The net- work is the computer." In 2008, David Patterson of UC Berkeley said, "The data center is the compu- ter. There are dramatic differences between developing software for millions to use as a service versus distributing software to run on their PCs." Recently, Rajkumar Buyya of Melbourne University simply said: "The cloud is the computer."

This book covers clusters, MPPs, P2P networks, grids, clouds, web services, social networks, and the IoT. In fact, the differences among clusters, grids, P2P systems, and clouds may blur in the future. Some people view clouds as grids or clusters with modest changes through virtualization. Others feel the changes could be major, since clouds are anticipated to process huge data sets gener- ated by the traditional Internet, social networks, and the future IoT. In subsequent chapters, the distinctions and dependencies among all distributed and cloud systems models will become clearer and more transparent.

1.1.1.5 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 distributed, centralized, and parallel computing. The following list defines these terms more clearly; their architectural and operational differences are discussed further in subsequent chapters

- 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 [18,26].
- 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* [15,27]. Interprocessor communication is accomplished through shared memory or via message passing. A computer system capable of parallel computing is commonly known as a *parallel computer* [28]. Programs running in a parallel computer are called *parallel programs*. The process of writing *parallel programs* is often referred to as *parallel programming* [32].
- Distributed computing This is a field of computer science/engineering that studies distributed systems. A *distributed system* [8,13,37,46] 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* [11,19].

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. 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. This book covers all the aforementioned computing paradigms, placing more emphasis on distributed and cloud com- puting and their working systems, including the clusters, grids, P2P, and cloud systems.

1.1.1.6 Distributed System Families

Since the mid-1990s, technologies for building P2P networks and *networks of clusters* have been consolidated into many national projects designed to establish wide area computing infrastructures, known as *computational grids* or *data grids*. Recently, we have witnessed a surge in interest in exploring Internet cloud resources for data-intensive applications. Internet clouds are the result of moving desktop computing to service-oriented computing using server clusters and huge databases

at data centers. This chapter introduces the basics of various parallel and distributed families. Grids and clouds are disparity systems that place great emphasis on resource sharing in hardware, software, and data sets.

Design theory, enabling technologies, and case studies of these massively distributed systems are also covered in this book. Massively distributed systems are intended to exploit a high degree of parallelism or concurrency among many machines. In October 2010, the highest performing cluster machine was built in China with 86016 CPU processor cores and 3,211,264 GPU cores in a Tianhe-1A system. The largest computational grid connects up to hundreds of server clus- ters. A typical P2P network may involve millions of client machines working simultaneously. Experimental cloud computing clusters have been built with thousands of processing nodes. We devote the material in Chapters 4 through 6 to cloud computing. Case studies of HTC systems will be examined in Chapters 4 and 9, including data centers, social networks, and virtualized cloud platforms

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). 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.

1.1.2 Scalable Computing Trends and New Paradigms

Several predictable trends in technology are known to drive computing applications. In fact, designers and programmers want to predict the technological capabilities of future systems. For instance, Jim Gray's paper, "Rules of Thumb in Data Engineering," is an excellent example of how technology affects applications and vice versa. In addition, Moore's law indicates that processor speed doubles every 18 months. Although Moore's law has been proven valid over the last 30 years, it is difficult to say whether it will continue to be true in the future.

Gilder's law indicates that network bandwidth has doubled each year in the past. Will that trend continue in the future? The tremendous price/performance ratio of commodity hardware was driven by the desktop, notebook, and tablet computing markets. This has also driven the adoption and use of commodity technologies in large-scale computing. We will discuss the future of these computing trends in more detail in subsequent chapters. For now, it's important to understand how distributed

systems emphasize both resource distribution and concurrency or high *degree of parallelism (DoP)*. Let's review the degrees of parallelism before we discuss the special requirements for distributed computing.

1.1.2.1 Degrees of Parallelism

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

Data-level parallelism (DLP) was made popular through *SIMD (single instruction, multiple data)* and vector machines using vector or array types of instructions. DLP requires even more hard-ware support and compiler assistance to work properly. Ever since the introduction of multicore processors and *chip multiprocessors (CMPs)*, we have been exploring *task-level parallelism (TLP)*. A modern processor explores all of the aforementioned parallelism types. In fact, BLP, ILP, and DLP are well supported by advances in hardware and compilers. However, TLP is far from being very successful due to difficulty in programming and compilation of code for efficient execution on multicore CMPs. 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.

1.1.2.2 Innovative Applications

Both HPC and HTC systems desire transparency in many application aspects. For example, data access, resource allocation, process location, concurrency in execution, job replication, and failure recovery should be made transparent to both users and system management. Table 1.1 highlights a few key applications that have driven the development of parallel and distributed systems over the

Table 1.1 Applications of High-Performance and High-Throughput Systems

Domain	Specific Applications
Science and engineering	Scientific simulations, genomic analysis, etc. Earthquake prediction, global warming, weather forecasting, etc.
Business, education, services industry, and health care	Telecommunication, content delivery, e-commerce, etc. Banking, stock exchanges, transaction processing, etc. Air traffic control, electric power grids, distance education, etc. Health care, hospital automation, telemedicine, etc.
Internet and web services, and government applications	Internet search, data centers, decision-making systems, etc. Traffic monitoring, worm containment, cyber security, etc. Digital government, online tax return processing, social networking, etc.
Mission-critical applications	Military command and control, intelligent systems, crisis management, etc.

years. These applications spread across many important domains in science, engineering, business, education, health care, traffic control, Internet and web services, military, and government applications.

Almost all applications demand computing economics, web-scale data collection, system reliability, and scalable performance. For example, distributed transaction processing is often prac- ticed in the banking and finance industry. Transactions represent 90 percent of the existing market for reliable banking systems. Users must deal with multiple database servers in distributed transactions. Maintaining the consistency of replicated transaction records is crucial in real-time banking services. Other complications include lack of software support, network saturation, and security threats in these applications. We will study applications and software support in more detail in subsequent chapters.

1.1.2.3 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 dis- covery. Finally, these paradigms are composable with QoS and *SLAs (service-level agreements)*. These paradigms and their attributes realize the computer utility vision.

Utility computing focuses on a business model in which customers receive computing resources from a paid service provider. All grid/cloud platforms are regarded as utility service providers. However, cloud computing offers a broader concept than utility computing. Distributed cloud applications run on any available servers in some edge networks. Major technological challenges include all aspects of computer science and engineering. For example, users demand new network-efficient processors, scalable memory and storage schemes, distributed OSes, middleware for machine virtualization, new programming models, effective resource management, and application

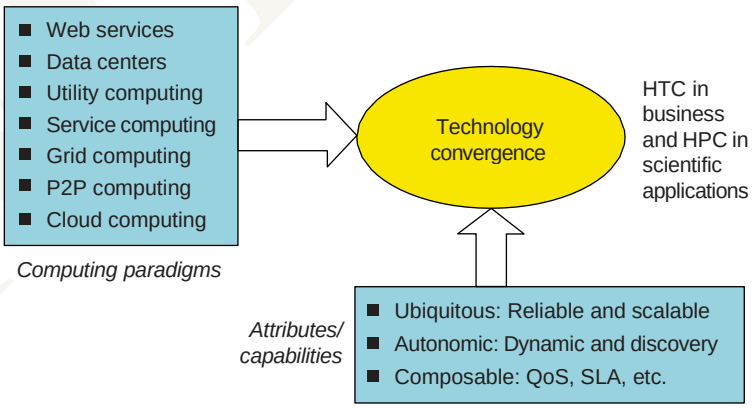

FIGURE 1.2

The vision of computer utilities in modern distributed computing systems.

(*Modified from presentation slide by Raj Buyya, 2010*)

program development. These hardware and software supports are necessary to build distributed systems that explore massive parallelism at all processing levels.

1.1.2.4 The Hype Cycle of New Technologies

Any new and emerging computing and information technology may go through a hype cycle, as illustrated in Figure 1.3. This cycle shows the expectations for the technology at five different stages. The expectations rise sharply from the trigger period to a high peak of inflated expectations. Through a short period of disillusionment, the expectation may drop to a valley and then increase steadily over a long enlightenment period to a plateau of productivity. The number of years for an emerging technology to reach a certain stage is marked by special symbols. The hollow circles indicate technologies that will reach mainstream adoption in two years. The gray circles represent technologies that will reach mainstream adoption in two to five years. The solid circles represent those that require five to 10 years to reach mainstream adoption, and the triangles denote those that require more than 10 years. The crossed circles represent technologies that will become obsolete before they reach the plateau.

The hype cycle in Figure 1.3 shows the technology status as of August 2010. For example, at that time *consumer-generated media* was at the disillusionment stage, and it was predicted to take less than two years to reach its plateau of adoption. *Internet micropayment systems* were forecast to take two to five years to move from the enlightenment stage to maturity. It was believed that *3D printing* would take five to 10 years to move from the rising expectation stage to mainstream adop- tion, and *mesh network sensors* were expected to take more than 10 years to move from the inflated expectation stage to a plateau of mainstream adoption.

Also as shown in Figure 1.3, the *cloud technology* had just crossed the peak of the expectation stage in 2010, and it was expected to take two to five more years to reach the productivity stage. However, *broadband over power line* technology was expected to become obsolete before leaving the valley of disillusionment stage in 2010. Many additional technologies (denoted by dark circles in Figure 1.3) were at their peak expectation stage in August 2010, and they were expected to take five to 10 years to reach their plateau of success. Once a technology begins to climb the slope of enlightenment, it may reach the productivity plateau within two to five years. Among these promis- ing technologies are the clouds, biometric authentication, interactive TV, speech recognition, predic- tive analytics, and media tablets.

1.1.3 The Internet of Things and Cyber-Physical Systems

In this section, we will discuss two Internet development trends: the Internet of Things [48] and cyber-physical systems. These evolutionary trends emphasize the extension of the Internet to every- day objects. We will only cover the basics of these concepts here; we will discuss them in more detail in Chapter 9.

1.1.3.1 The Internet of Things

The traditional Internet connects machines to machines or web pages to web pages. The concept of the IoT was introduced in 1999 at MIT [40]. The IoT refers to the networked interconnection of everyday objects, tools, devices, or computers. One can view the IoT as a wireless network of sen- sors that interconnect all things in our daily life. These things can be large or small and they vary

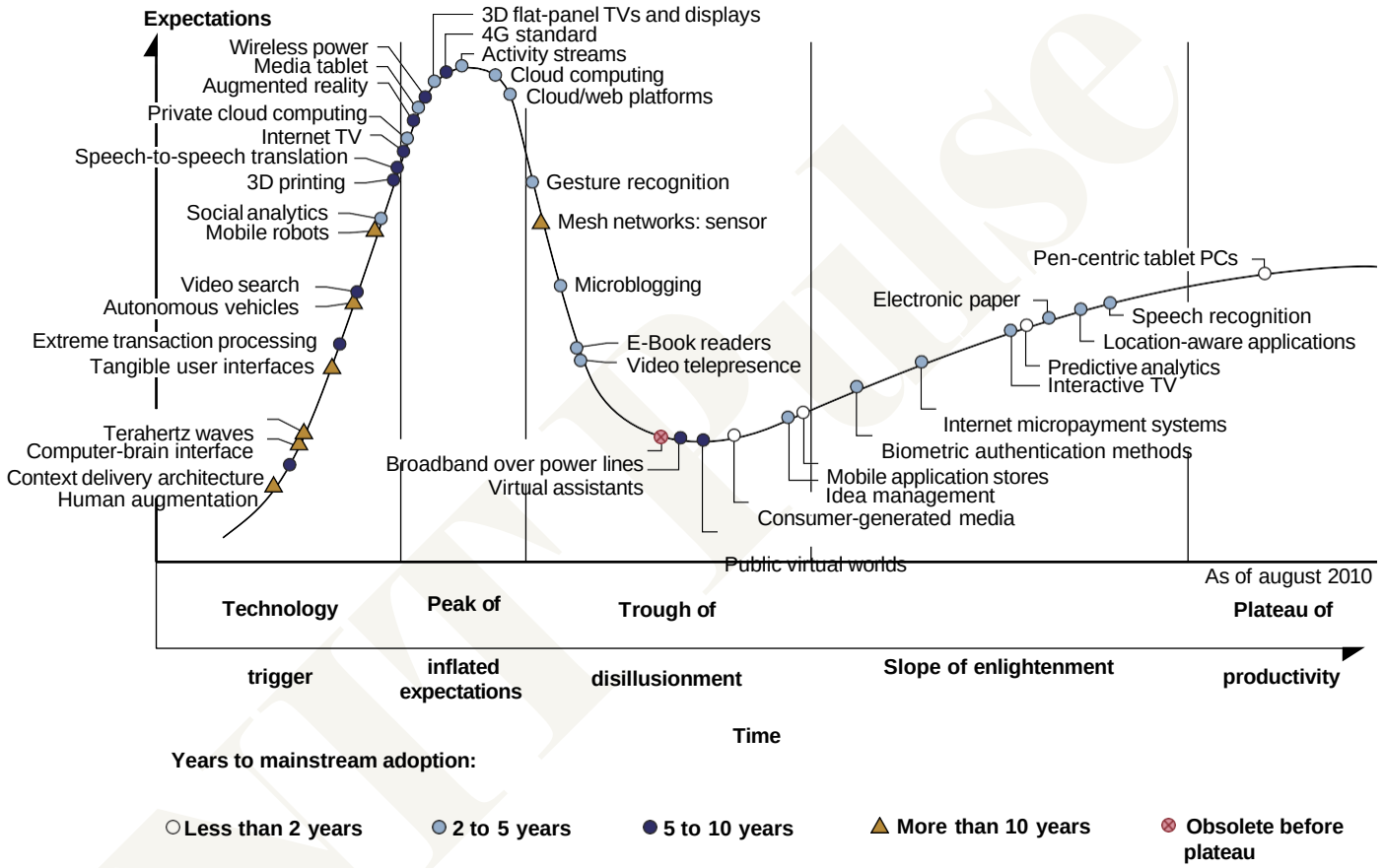
Expectations
3D flat-panel TVs and displays
4G standard
Activity streams
Wireless power
Media tablet
Augmented reality
Cloud computing
Cloud/web platforms
Private cloud computing
Internet TV
Speech-to-speech translation
3D printing
Gesture recognition
Social analytics
Mobile robots
Mesh networks: sensor
Pen-centric tablet PCs
Video search
Microblogging
Electronic paper
Autonomous vehicles
Speech recognition
Extreme transaction processing
E-Book readers
Location-aware applications
Tangible user interfaces
Video telepresence
Predictive analytics
Interactive TV
Terahertz waves
Internet micropayment systems
Computer-brain interface
Biometric authentication methods
Broadband over power lines
Mobile application stores
Context delivery architecture
Virtual assistants
Idea management
Human augmentation
Consumer-generated media
Public virtual worlds
As of august 2010
Technology trigger
Peak of inflated expectations
Trough of disillusionment
Slope of enlightenment
Plateau of productivity
Time
Years to mainstream adoption:
Less than 2 years
2 to 5 years
5 to 10 years
More than 10 years
Obsolete before plateau

FIGURE 1.3

Hype cycle for Emerging Technologies, 2010.

12

Hype Cycle Disclaimer

The Hype Cycle is copyrighted 2010 by Gartner, Inc. and its affiliates and is reused with permission. Hype Cycles are graphical representations of the relative maturity of technologies, IT methodologies and management disciplines. They are intended solely as a research tool, and not as a specific guide to action. Gartner disclaims all warranties, express or implied, with respect to this research, including any warranties of merchantability or fitness for a particular purpose.

This Hype Cycle graphic was published by Gartner, Inc. as part of a larger research note and should be evaluated in the context of the entire report. The Gartner report is available at http://www.gartner.com/it/page.jsp?id=1447613.

source: Gartner Press Release "G's 2010 Hype Cycle Special Report Evaluates Maturity of 1,800 Technologies" 7 October 2010.)

with respect to time and place. The idea is to tag every object using RFID or a related sensor or electronic technology such as GPS.

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

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

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

1.1.3.2 Cyber-Physical Systems

A *cyber-physical system* (CPS) is the result of interaction between computational processes and the physical world. A CPS integrates "cyber" (heterogeneous, asynchronous) with "physical" (concur- rent and information-dense) objects. A CPS merges the "3C" technologies of *computation, commu- nication*, and *control* into an intelligent closed feedback system between the physical world and the information world, a concept which is actively explored in the United States. The IoT emphasizes various networking connections among physical objects, while the CPS emphasizes exploration of *virtual reality* (VR) applications in the physical world. We may transform how we interact with the physical world just like the Internet transformed how we interact with the virtual world. We will study IoT, CPS, and their relationship to cloud computing in Chapter 9.

1.2 TECHNOLOGIES FOR NETWORK-BASED SYSTEMS

With the concept of scalable computing under our belt, it's time to explore hardware, software, and network technologies for distributed computing system design and applications. In particular, we will focus on viable approaches to building distributed operating systems for handling massive par- allelism in a distributed environment.

1.2.1 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.

1.2.1.1 Advances in CPU Processors

Today, advanced CPUs or microprocessor chips assume a multicore architecture with dual, quad, six, or more processing cores. These processors exploit parallelism at ILP and TLP levels. Processor speed growth is plotted in the upper curve in Figure 1.4 across generations of microprocessors or CMPs. We see growth from 1 MIPS for the VAX 780 in 1978 to 1,800 MIPS for the Intel Pentium 4 in 2002, up to a 22,000 MIPS peak for the Sun Niagara 2 in 2008. As the figure shows, Moore's law has proven to be pretty accurate in this case. The clock rate for these processors increased from 10 MHz for the Intel 286 to 4 GHz for the Pentium 4 in 30 years.

However, the clock rate reached its limit on CMOS-based chips due to power limitations. At the time of this writing, very few CPU chips run with a clock rate exceeding 5 GHz. In other words, clock rate will not continue to improve unless chip technology matures. This limitation is attributed primarily to excessive heat generation with high frequency or high voltages. The ILP is highly exploited in modern CPU processors. ILP mechanisms include multiple-issue superscalar architecture, dynamic branch prediction, and speculative execution, among others. These ILP techniques demand hardware and compiler support. In addition, DLP and TLP are highly explored in *graphics processing units* (GPUs) that adopt a many-core architecture with hundreds to thousands of simple cores.

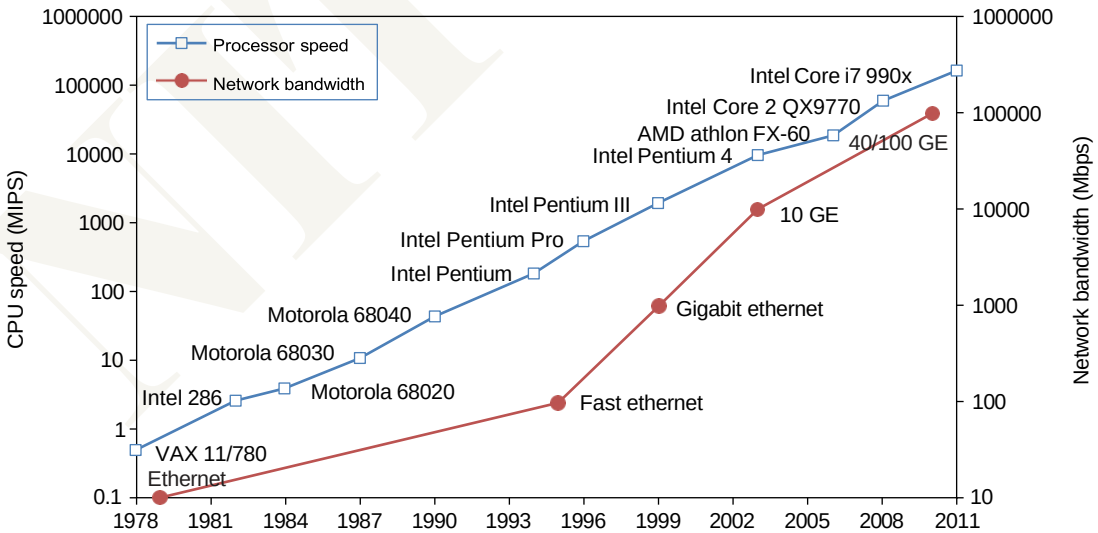

FIGURE 1.4

Improvement in processor and network technologies over 33 years.

(*Courtesy of Xiaosong Lou and Lizhong Chen of University of Southern California, 2011*)

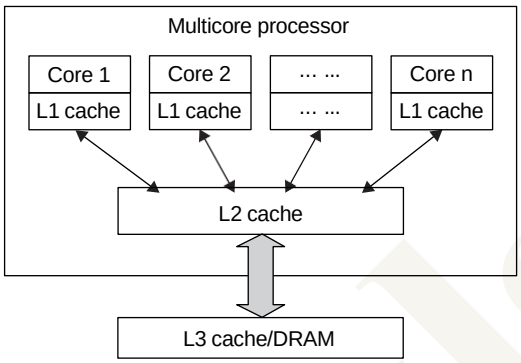

FIGURE 1.5

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.

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 multi- threaded 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. This implies that the maximum ILP and TLP that can be exploited in Niagara is 64 (8 × 8 = 64). In 2011, the Intel Core i7 990x has reported 159,000 MIPS execution rate as shown in the upper- most square in Figure 1.4.

1.2.1.2 Multicore CPU and Many-Core GPU Architectures

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

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.

1.2.1.3 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

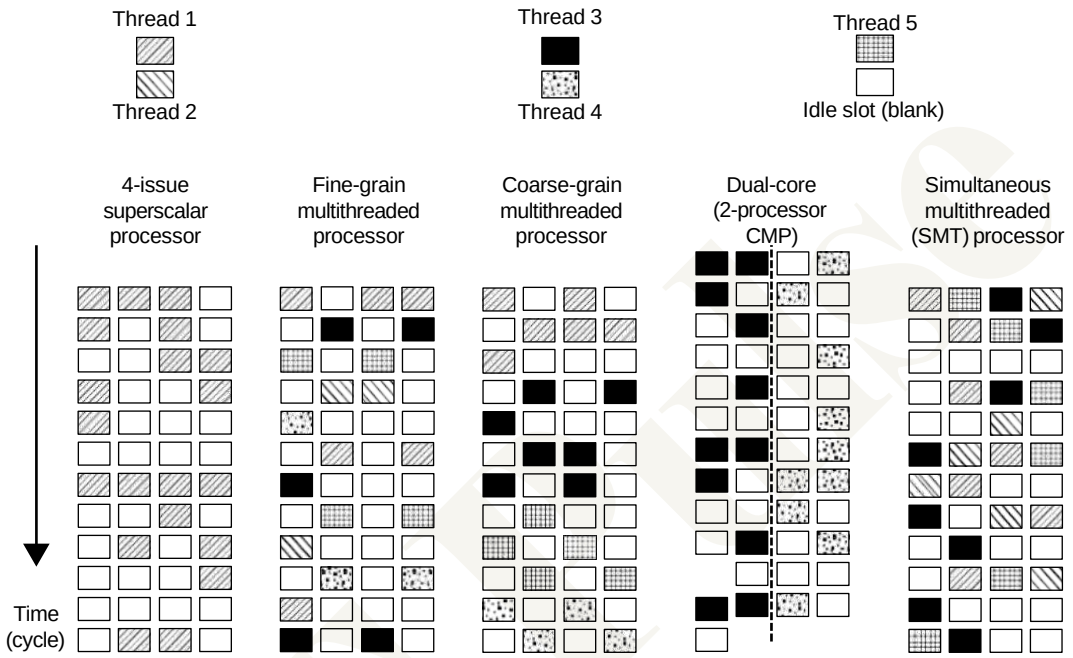

FIGURE 1.6

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

four-issue superscalar processor, a fine-grain multithreaded processor, a coarse-grain multi- threaded 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.

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 com- pletely. The SMT allows simultaneous scheduling of instructions from different threads in the same cycle.

These execution patterns closely mimic an ordinary program. The blank squares correspond to no available instructions for an instruction data path at a particular processor cycle. More blank cells imply lower scheduling efficiency. The maximum ILP or maximum TLP is difficult to achieve at each processor cycle. The point here is to demonstrate your understanding of typical instruction scheduling patterns in these five different micro-architectures in modern processors.

1.2.2 GPU Computing to Exascale 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.

Unlike CPUs, GPUs have a throughput architecture that exploits massive parallelism by executing many concurrent threads slowly, instead of executing a single long thread in a conven- tional 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 GPGPUs, have appeared in the HPC field. NVIDIA's CUDA model was for HPC using GPGPUs. Chapter 2 will discuss GPU clusters for massively parallel computing in more detail [15,32].

1.2.2.1 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.

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. GPUs 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 sys- tems, PCs, and servers. The NVIDIA CUDA Tesla or Fermi is used in GPU clusters or in HPC sys- tems for parallel processing of massive floating-pointing data.

1.2.2.2 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. We will study the use of CUDA GPUs in large-scale cluster computing in Chapter 2.

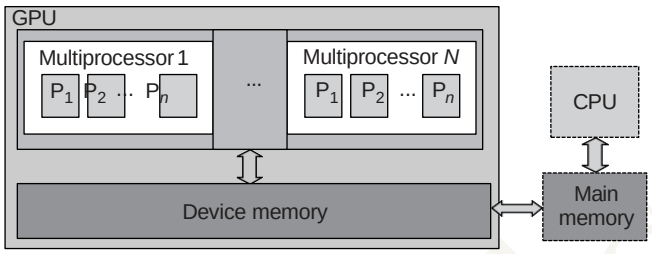

FIGURE 1.7

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

(*Courtesy of B. He, et al., PACT'08 [23]*)

Example 1.1 The NVIDIA Fermi GPU Chip with 512 CUDA Cores

In November 2010, three of the five fastest supercomputers in the world (the Tianhe-1a, Nebulae, and Tsubame) used large numbers of GPU chips to accelerate floating-point computations. Figure 1.8 shows the architecture of the Fermi GPU, a next-generation GPU from NVIDIA. This is a *streaming multiprocessor (SM)* module. Multiple SMs can be built on a single GPU chip. The Fermi chip has 16 SMs implemented with 3 billion transistors. Each SM comprises up to 512 *streaming processors (SPs)*, known as *CUDA cores*. The Tesla GPUs used in the Tianhe-1a have a similar architecture, with 448 CUDA cores.

The Fermi GPU is a newer generation of GPU, first appearing in 2011. The Tesla or Fermi GPU can be used in desktop workstations to accelerate floating-point calculations or for building large-scale data centers. The architecture shown is based on a 2009 white paper by NVIDIA [36]. There are 32 CUDA cores per SM. Only one SM is shown in Figure 1.8. Each CUDA core has a simple pipelined integer ALU and an FPU that can be used in parallel. Each SM has 16 load/store units allowing source and destination addresses to be calculated for 16 threads per clock. There are four *special function units (SFUs)* for executing transcendental instructions.

All functional units and CUDA cores are interconnected by an *NoC (network on chip)* to a large number of SRAM banks (L2 caches). Each SM has a 64 KB L1 cache. The 768 KB unified L2 cache is shared by all SMs and serves all load, store, and texture operations. *Memory controllers* are used to connect to 6 GB of off-chip DRAMs. The SM schedules threads in groups of 32 parallel threads called *warps*. In total, 256/512 *FMA (fused multiply and add)* operations can be done in parallel to produce 32/64-bit floating-point results. The 512 CUDA cores in an SM can work in parallel to deliver up to 515 Gflops of double-precision results, if fully utilized. With 16 SMs, a single GPU has a peak speed of 82.4 Tflops. Only 12 Fermi GPUs have the potential to reach the Pflops performance.

In the future, thousand-core GPUs may appear in Exascale (Eflops or 10^{18} flops) systems. This reflects a trend toward building future MPPs with hybrid architectures of both types of processing chips. In a DARPA report published in September 2008, four challenges are identified for exascale computing: (1) energy and power, (2) memory and storage, (3) concurrency and locality, and (4) system resiliency. Here, we see the progress of GPUs along with CPU advances in power

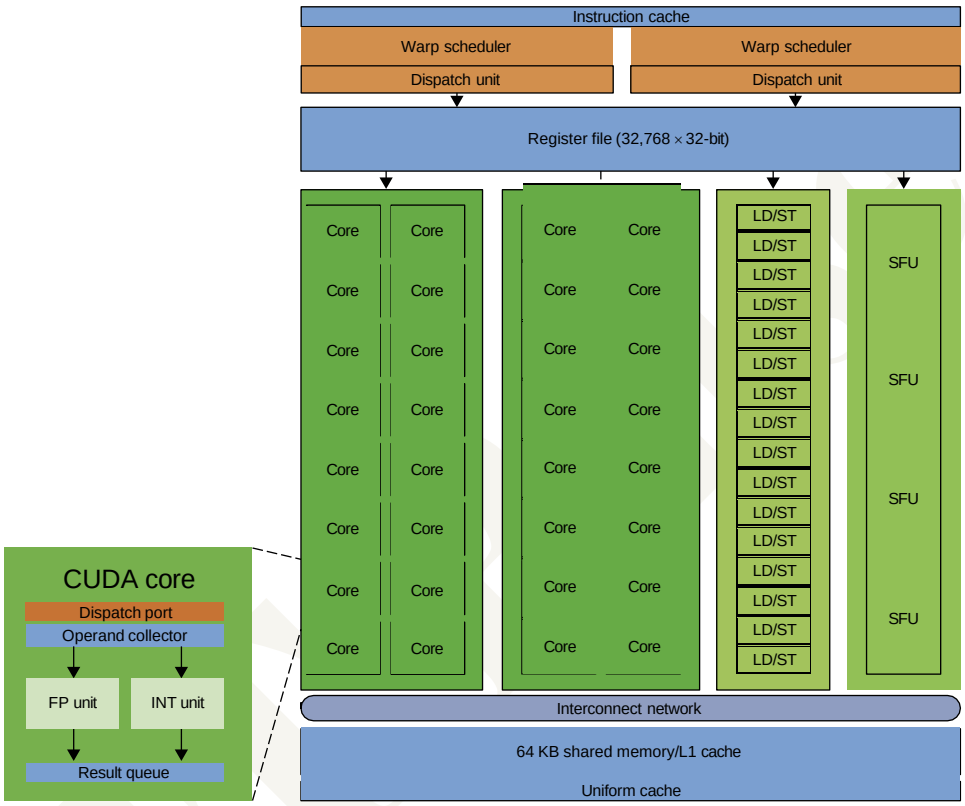

FIGURE 1.8

NVIDIA Fermi GPU built with 16 streaming multiprocessors (SMs) of 32 CUDA cores each; only one SM Is shown. More details can be found also in [49].

(*Courtesy of NVIDIA, 2009 [36] 2011*)

efficiency, performance, and programmability [16]. In Chapter 2, we will discuss the use of GPUs to build large clusters.

1.2.2.3 Power Efficiency of the GPU

Bill Dally of Stanford University considers power and massive parallelism as the major benefits of GPUs over CPUs for the future. By extrapolating current technology and computer architecture, it was estimated that 60 Gflops/watt per core is needed to run an exaflops system (see Figure 1.10). Power constrains what we can put in a CPU or GPU chip. Dally has estimated that the CPU chip consumes about 2 nJ/instruction, while the GPU chip requires 200 pJ/instruction, which is 1/10 less than that of the CPU. The CPU is optimized for latency in caches and memory, while the GPU is optimized for throughput with explicit management of on-chip memory.

Figure 1.9 compares the CPU and GPU in their performance/power ratio measured in Gflops/ watt per core. In 2010, the GPU had a value of 5 Gflops/watt at the core level, compared with less

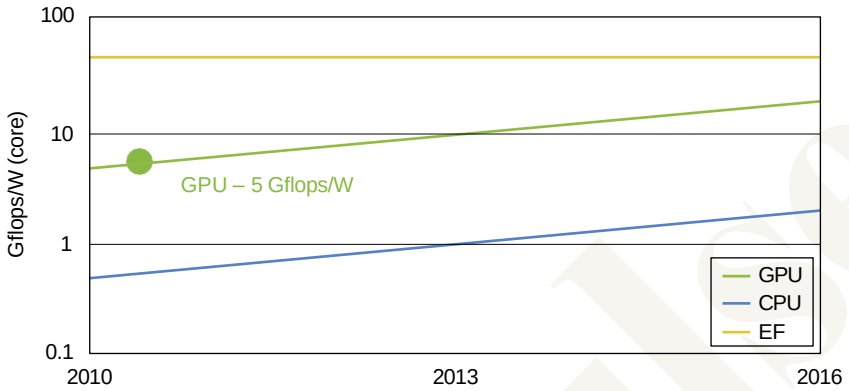

FIGURE 1.9

The GPU performance (middle line, measured 5 Gflops/W/core in 2011), compared with the lower CPU performance (lower line measured 0.8 Gflops/W/core in 2011) and the estimated 60 Gflops/W/core performance in 2011 for the Exascale (EF in upper curve) in the future.

(*Courtesy of Bill Dally [15]*)

than 1 Gflop/watt per CPU core. This may limit the scaling of future supercomputers. However, the GPUs may close the gap with the CPUs. Data movement dominates power consumption. One needs to optimize the storage hierarchy and tailor the memory to the applications. We need to promote self-aware OS and runtime support and build locality-aware compilers and auto-tuners for GPU- based MPPs. This implies that both power and software are the real challenges in future parallel and distributed computing systems.

1.2.3 Memory, Storage, and Wide-Area Networking

1.2.3.1 Memory Technology

The upper curve in Figure 1.10 plots the growth of DRAM chip capacity from 16 KB in 1976 to 64 GB in 2011. This shows that memory chips have experienced a 4x increase in capacity every three years. Memory access time did not improve much in the past. In fact, the memory wall problem is getting worse as the processor gets faster. For hard drives, capacity increased from 260 MB in 1981 to 250 GB in 2004. The Seagate Barracuda XT hard drive reached 3 TB in 2011. This represents an approximately 10x increase in capacity every eight years. The capacity increase of disk arrays will be even greater in the years to come. Faster processor speed and larger memory capacity result in a wider gap between processors and memory. The memory wall may become even worse a problem limiting the CPU performance in the future.

1.2.3.2 Disks and Storage Technology

Beyond 2011, disks or disk arrays have exceeded 3 TB in capacity. The lower curve in Figure 1.10 shows the disk storage growth in 7 orders of magnitude in 33 years. The rapid growth of flash memory and *solid-state drives* (SSDs) also impacts the future of HPC and HTC systems. The mor- tality rate of SSD is not bad at all. A typical SSD can handle 300,000 to 1 million write cycles per

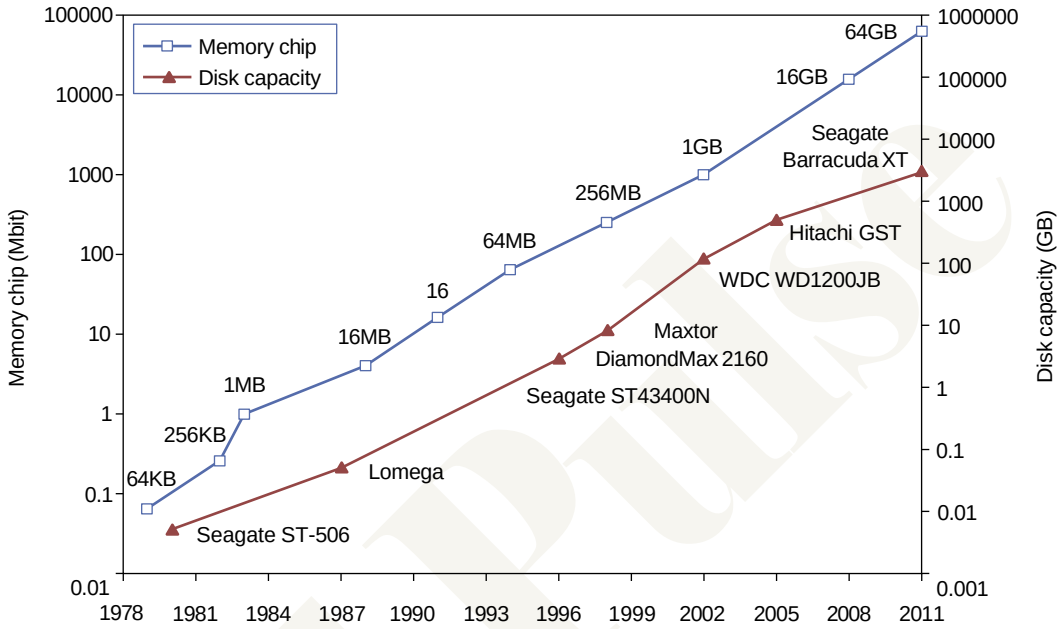

FIGURE 1.10

Improvement in memory and disk technologies over 33 years. The Seagate Barracuda XT disk has a capacity of 3 TB in 2011.

(*Courtesy of Xiaosong Lou and Lizhong Chen of University of Southern California, 2011*)

block. So the SSD can last for several years, even under conditions of heavy write usage. Flash and SSD will demonstrate impressive speedups in many applications.

Eventually, power consumption, cooling, and packaging will limit large system development. Power increases linearly with respect to clock frequency and quadratic ally with respect to voltage applied on chips. Clock rate cannot be increased indefinitely. Lowered voltage supplies are very much in demand. Jim Gray once said in an invited talk at the University of Southern California, "*Tape units are dead, disks are tape units, flashes are disks, and memory are caches now*." This clearly paints the future for disk and storage technology. In 2011, the SSDs are still too expensive to replace stable disk arrays in the storage market.

1.2.3.3 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 commercialy available.22 CHAPTER 1 Distributed System Models and Enabling Technologies

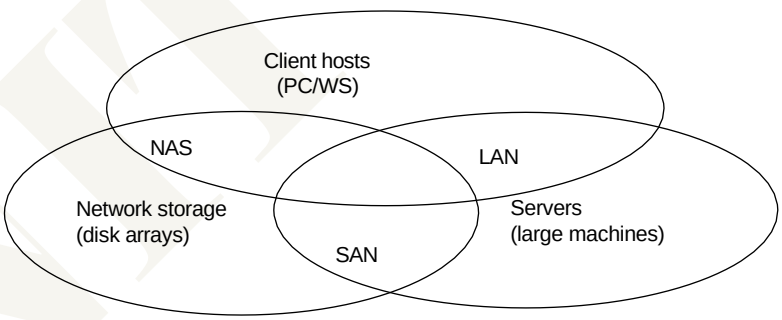

FIGURE 1.11

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.

1.2.3.4 Wide-Area Networking

The lower curve in Figure 1.10 plots the rapid growth of Ethernet bandwidth from 10 Mbps in 1979 to 1 Gbps in 1999, and 40 ~ 100 GE in 2011. It has been speculated that 1 Tbps network links will become available by 2013. According to Berman, Fox, and Hey [6], network links with 1,000, 1,000, 100, 10, and 1 Gbps bandwidths were reported, respectively, for international, national,

organization, optical desktop, and copper desktop connections in 2006.

An increase factor of two per year on network performance was reported, which is faster than Moore's law on CPU speed doubling every 18 months. The implication is that more computers will be used concurrently in the future. High-bandwidth networking increases the capability of building massively distributed systems. The IDC 2010 report predicted that both InfiniBand and Ethernet will be the two major interconnect choices in the HPC arena. Most data centers are using Gigabit Ethernet as the interconnect in their server clusters.

1.2.4 Virtual Machines and Virtualization Middleware

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

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. We will cover virtualization in Chapter 3. However, the basic concepts of virtualized resources, such as VMs, virtual storage, and virtual networking and their virtualization software or middleware, need to be introduced first. Figure 1.12 illustrates the architectures of three VM configurations.

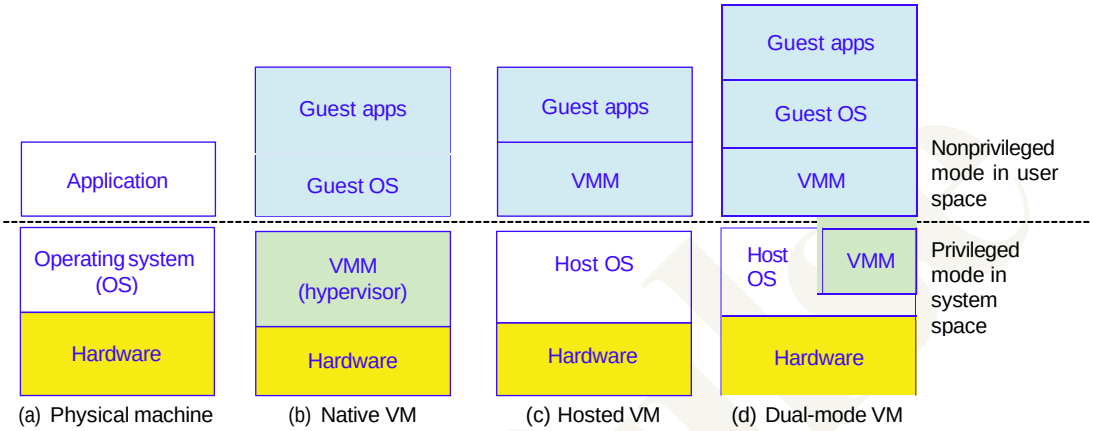

FIGURE 1.12

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

(*Courtesy of M. Abde-Majeed and S. Kulkarni, 2009 USC*)

1.2.4.1 Virtual Machines

In Figure 1.12, the host machine is equipped with the physical hardware, as shown at the bottom of the figure. An example is an x-86 architecture desktop running its installed Windows OS, as shown in part (a) of the figure. The VM can be provisioned for any hardware system. The VM is built with virtual resources managed by a guest OS to run a specific application. Between the VMs and the host platform, one needs to deploy a middleware layer called a *virtual machine monitor (VMM)*. Figure 1.12(b) shows a native VM installed with the use of a VMM called a *hypervisor* in privi- leged mode. For example, the hardware has x-86 architecture running the Windows system.

The guest OS could be a Linux system and the hypervisor is the XEN system developed at Cambridge University. This hypervisor approach is also called *bare-metal VM*, because the hypervisor handles the bare hardware (CPU, memory, and I/O) directly. Another architecture is the host VM shown in Figure 1.12(c). Here the VMM runs in nonprivileged mode. The host OS need not be modified. The VM can also be implemented with a dual mode, as shown in Figure 1.12(d). Part of the VMM runs at the user level and another part runs at the supervisor level. In this case, the host OS may have to be modified to some extent. Multiple VMs can be ported to a given hardware system to support the virtualization process. The VM approach offers hardware independence of the OS and applications. The user application running on its dedicated OS could be bundled together as a *virtual appliance* that can be ported to any hardware platform. The VM could run on an OS different from that of the host computer.

1.2.4.2 VM Primitive Operations

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

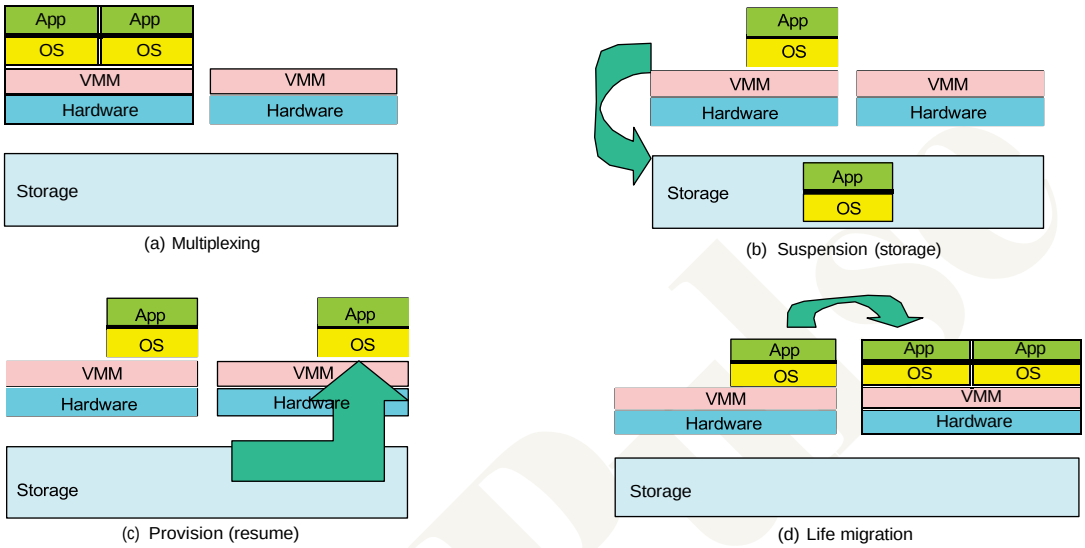

FIGURE 1.13

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

(*Courtesy of M. Rosenblum, Keynote address, ACM ASPLOS 2006 [41]*)

- 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).
- Third, 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).

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.

1.2.4.3 Virtual Infrastructures

Physical resources for compute, storage, and networking at the bottom of Figure 1.14 are mapped to the needy applications embedded in various VMs at the top. Hardware and software are then separated. Virtual infrastructure is what connects resources to distributed applications. It is a dynamic mapping of system resources to specific applications. The result is decreased costs and increased efficiency and responsiveness. Virtualization for server consolidation and containment is a good example of this. We will discuss VMs and virtualization support in Chapter 3. Virtualization support for clusters, clouds, and grids is covered in Chapters 3, 4, and 7, respectively.

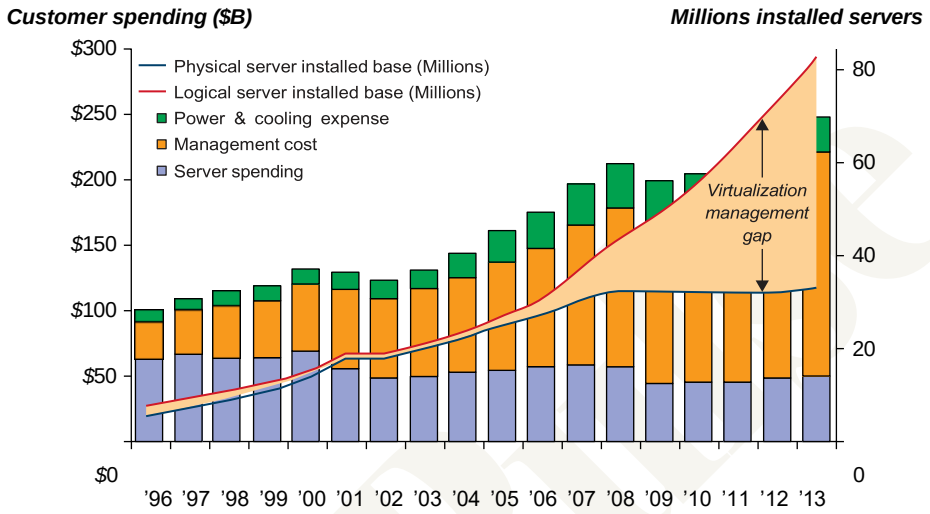
Customer spending ($B)
Millions installed servers
$300
$250
$200
$150
$100
$50
$0
80
60
40
20
0
Physical server installed base (Millions)
Logical server installed base (Millions)
Power & cooling expense
Management cost
Server spending
Virtualization management gap
'96 '97 '98 '99 '00 '01 '02 '03 '04 '05 '06 '07 '08 '09 '10 '11 '12 '13

FIGURE 1.14

Growth and cost breakdown of data centers over the years.

(*Source: IDC Report, 2009*)

1.2.5 Data Center Virtualization for Cloud Computing

In this section, we discuss basic architecture and design considerations of data centers. Cloud architecture is built with commodity hardware and network devices. Almost all cloud platforms choose the popular x86 processors. Low-cost terabyte disks and Gigabit Ethernet are used to build data centers. Data center design emphasizes the performance/price ratio over speed performance alone. In other words, storage and energy efficiency are more important than shear speed performance. Figure 1.13 shows the server growth and cost breakdown of data centers over the past 15 years. Worldwide, about 43 million servers are in use as of 2010. The cost of utilities exceeds the cost of hardware after three years.

1.2.5.1 Data Center Growth and Cost Breakdown

A large data center may be built with thousands of servers. Smaller data centers are typically built with hundreds of servers. The cost to build and maintain data center servers has increased over the years. According to a 2009 IDC report (see Figure 1.14), typically only 30 percent of data center costs goes toward purchasing IT equipment (such as servers and disks), 33 percent is attributed to the chiller, 18 percent to the *uninterruptible power supply (UPS)*, 9 percent to *computer room air conditioning (CRAC)*, and the remaining 7 percent to power distribution, lighting, and transformer costs. Thus, about 60 percent of the cost to run a data center is allocated to management and main- tenance. The server purchase cost did not increase much with time. The cost of electricity and cool- ing did increase from 5 percent to 14 percent in 15 years.

1.2.5.2 Low-Cost Design Philosophy

High-end switches or routers may be too cost-prohibitive for building data centers. Thus, using high-bandwidth networks may not fit the economics of cloud computing. Given a fixed budget,

commodity switches and networks are more desirable in data centers. Similarly, using commodity x86 servers is more desired over expensive mainframes. The software layer handles network traffic balancing, fault tolerance, and expandability. Currently, nearly all cloud computing data centers use Ethernet as their fundamental network technology.

1.2.5.3 Convergence of Technologies

Essentially, cloud computing is enabled by the convergence of technologies in four areas: (1) hard- ware virtualization and multi-core chips, (2) utility and grid computing, (3) SOA, Web 2.0, and WS mashups, and (4) atonomic computing and data center automation. Hardware virtualization and mul- ticore chips enable the existence of dynamic configurations in the cloud. Utility and grid computing technologies lay the necessary foundation for computing clouds. Recent advances in SOA, Web 2.0, and mashups of platforms are pushing the cloud another step forward. Finally, achievements in autonomic computing and automated data center operations contribute to the rise of cloud computing.

Jim Gray once posted the following question: "*Science faces a data deluge. How to manage and analyze information?*" This implies that science and our society face the same challenge of data deluge. Data comes from sensors, lab experiments, simulations, individual archives, and the web in all scales

and formats. Preservation, movement, and access of massive data sets require generic tools supporting high-performance, scalable file systems, databases, algorithms, workflows, and visualization. With science becoming data-centric, a new paradigm of scientific discovery is becoming based on data-intensive technologies.

On January 11, 2007, the *Computer Science and Telecommunication Board (CSTB)* recom- mended fostering tools for data capture, data creation, and data analysis. A cycle of interaction exists among four technical areas. First, cloud technology is driven by a surge of interest in data deluge. Also, cloud computing impacts e-science greatly, which explores multicore and parallel computing technologies. These two hot areas enable the buildup of data deluge. To support data- intensive computing, one needs to address workflows, databases, algorithms, and virtualization issues.

By linking computer science and technologies with scientists, a spectrum of e-science or e-research applications in biology, chemistry, physics, the social sciences, and the humanities has generated new insights from interdisciplinary activities. Cloud computing is a transformative approach as it promises much more than a data center model. It fundamentally changes how we interact with information. The cloud provides services on demand at the infrastructure, platform, or software level. At the platform level, MapReduce offers a new programming model that transpar- ently handles data parallelism with natural fault tolerance capability. We will discuss MapReduce in more detail in Chapter 6.

Iterative MapReduce extends MapReduce to support a broader range of data mining algorithms commonly used in scientific applications. The cloud runs on an extremely large cluster of commod- ity computers. Internal to each cluster node, multithreading is practiced with a large number of cores in many-core GPU clusters. Data-intensive science, cloud computing, and multicore comput- ing are converging and revolutionizing the next generation of computing in architectural design and programming challenges. They enable the pipeline: Data becomes information and knowledge, and in turn becomes machine wisdom as desired in SOA.

1.3 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 man- ner. With today's networking technology, a few LAN switches can easily connect hundreds of machines as a working cluster. A WAN can connect many local clusters to form a very large cluster of clusters. In this sense, one can build a massive system with millions of computers connected to edge networks.

Massive systems are considered highly scalable, and can reach web-scale connectivity, either physically or logically. In Table 1.2, massive systems are classified into four groups: *clusters*, *P2P networks*, *computing grids*, and *Internet clouds* over huge data centers. In terms of node number, these four system classes may involve hundreds, thousands, or even millions of computers as participating nodes. These machines work collectively, cooperatively, or collaboratively at various levels. The table entries characterize these four system classes in various technical and application aspects.

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

From the application perspective, clusters are most popular in supercomputing applications. In 2009, 417 of the Top 500 supercomputers were built with cluster architecture. It is fair to say that clusters have laid the necessary foundation for building large-scale grids and clouds. P2P networks appeal most to business applications. However, the content industry was reluctant to accept P2P technology for lack of copyright protection in ad hoc networks. Many national grids built in the past decade were underutilized for lack of reliable middleware or well-coded applications. Potential advantages of cloud computing include its low cost and simplicity for both providers and users.

1.3.1 Clusters of Cooperative Computers

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

1.3.1.1 Cluster Architecture

Figure 1.15 shows the architecture of a typical server cluster built around a low-latency, high-bandwidth interconnection network. This network can be as simple as a SAN (e.g., Myrinet) or a LAN (e.g., Ethernet). To build a larger cluster with more nodes, the interconnection network can be built with multiple levels of Gigabit Ethernet, Myrinet, or InfiniBand switches. Through hierarchical construction using a SAN, LAN, or WAN, one can build scalable clusters with an increasing number of nodes. The cluster is connected to the Internet via a virtual private network (VPN) gateway. The gateway IP address locates the cluster. The system image of a computer is decided by the way the OS manages the shared cluster resources. Most clusters have loosely coupled node computers. All resources of a server node are managed by their own OS. Thus, most clusters have multiple system images as a result of having many autonomous nodes under different OS control.

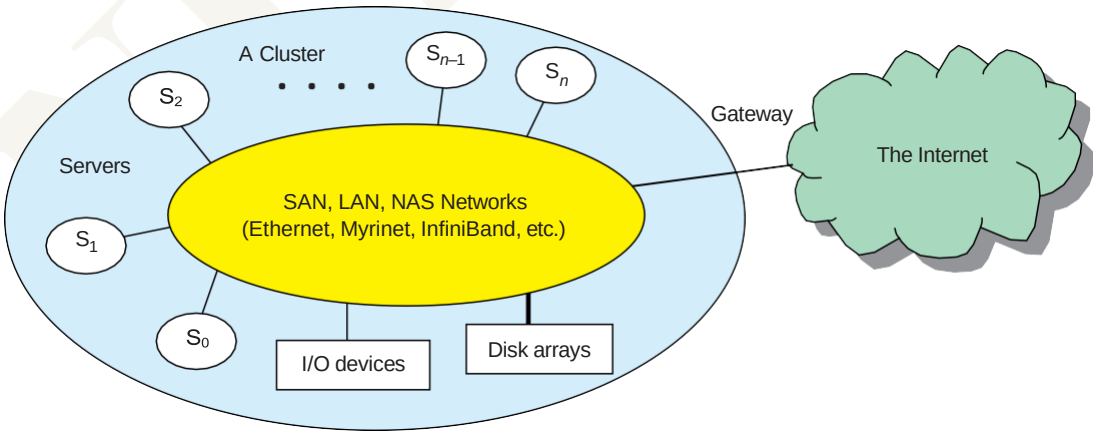

FIGURE 1.15

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.

1.3.1.2 Single-System Image

Greg Pfister [38] has indicated that an ideal cluster should merge multiple system images into a *single-system image (SSI)*. Cluster designers desire a *cluster operating system* or some middle- ware to support SSI at various levels, including the sharing of CPUs, memory, and I/O across all cluster nodes. An SSI is an illusion created by software or hardware that presents a collection of resources as one integrated, powerful resource. SSI makes the cluster appear like a single machine to the user. A cluster with multiple system images is nothing but a collection of inde- pendent computers.

1.3.1.3 Hardware, Software, and Middleware Support

In Chapter 2, we will discuss cluster design principles for both small and large clusters. Clusters exploring massive parallelism are commonly known as MPPs. Almost all HPC clusters in the Top 500 list are also MPPs. The building blocks are computer nodes (PCs, workstations, servers, or SMP), special communication software such as PVM or MPI, and a network interface card in each computer node. Most clusters run under the Linux OS. The computer nodes are interconnected by a high-bandwidth network (such as Gigabit Ethernet, Myrinet, InfiniBand, etc.).

Special cluster middleware supports are needed to create SSI or *high availability (HA)*. Both sequential and parallel applications can run on the cluster, and special parallel environments are needed to facilitate use of the cluster resources. For example, distributed memory has multiple images. Users may want all distributed memory to be shared by all servers by forming *distribu- ted shared memory (DSM)*. Many SSI features are expensive or difficult to achieve at various cluster operational levels. Instead of achieving SSI, many clusters are loosely coupled machines. Using virtualization, one can build many virtual clusters dynamically, upon user demand. We will discuss virtual clusters in Chapter 3 and the use of virtual clusters for cloud computing in Chapters 4, 5, 6, and 9.

1.3.1.4 Major Cluster Design Issues

Unfortunately, a cluster-wide OS for complete resource sharing is not available yet. Middleware or OS extensions were developed at the user space to achieve SSI at selected functional levels. Without this middleware, cluster nodes cannot work together effectively to achieve cooperative computing. The software environments and applications must rely on the middleware to achieve high performance. The cluster benefits come from scalable performance, efficient message passing, high system availability, seamless fault tolerance, and cluster-wide job management, as summarized in Table 1.3. We will address these issues in Chapter 2.

1.3.2 Grid Computing Infrastructures

In the past 30 years, users have experienced a natural growth path from Internet to web and grid computing services. Internet services such as the *Telnet* command enables a local computer to con- nect to a remote computer. A web service such as HTTP enables remote access of remote web pages. Grid computing is envisioned to allow close interaction among applications running on dis- tant computers simultaneously. *Forbes Magazine* has projected the global growth of the IT-based economy from $1 trillion in 2001 to $20 trillion by 2015. The evolution from Internet to web and grid services is certainly playing a major role in this growth.

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

1.3.2.1 Computational Grids

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

Figure 1.16 shows an example computational grid built over multiple resource sites owned by different organizations. The resource sites offer complementary computing resources, including workstations, large servers, a mesh of processors, and Linux clusters to satisfy a chain of computational needs. The grid is built across various IP broadband networks including LANs and WANs already used by enterprises or organizations over the Internet. The grid is presented to users as an integrated resource pool as shown in the upper half of the figure.

Special instruments may be involved such as using the radio telescope in SETI@Home search of life in the galaxy and the austrophysics@Swineburne for pulsars. At the server end, the grid is a network. At the client end, we see wired or wireless terminal devices. The grid integrates the computing, communication, contents, and transactions as rented services. Enter- prises and consumers form the user base, which then defines the usage trends and service char- acteristics. Many national and international grids will be reported in Chapter 7, the NSF

TeraGrid in US, EGEE in Europe, and ChinaGrid in China for various distributed scientific grid applications.

1.3.2.2 Grid Families

Grid technology demands new distributed computing models, software/middleware support, network protocols, and hardware infrastructures. National grid projects are followed by industrial grid plat- form development by IBM, Microsoft, Sun, HP, Dell, Cisco, EMC, Platform Computing, and others. New *grid service providers* (GSPs) and new grid applications have emerged rapidly, similar to the growth of Internet and web services in the past two decades. In Table 1.4, grid systems are classified in essentially two categories: *computational or data grids* and *P2P grids*. Computing or data grids are built primarily at the national level. In Chapter 7, we will cover grid applications and lessons learned.

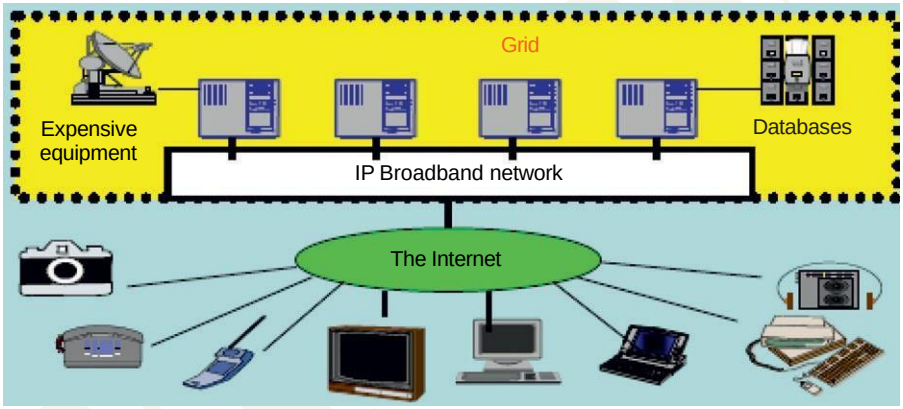

FIGURE 1.16

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

(*Courtesy of Z. Xu, Chinese Academy of Science, 2004*)

Table 1.4 Two Grid Computing Infrastructures and Representative Systems

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

1.3.3 Peer-to-Peer Network Families

An example of a well-established distributed system is the *client-server architecture*. In this sce- nario, 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 sys- tems are introduced at the physical level and overlay networks at the logical level.

1.3.3.1 P2P Systems

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 distrib- uted control.

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 parti- cipating 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 physi- cal network varies in size and topology dynamically due to the free membership in the P2P network.

1.3.3.2 Overlay Networks

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

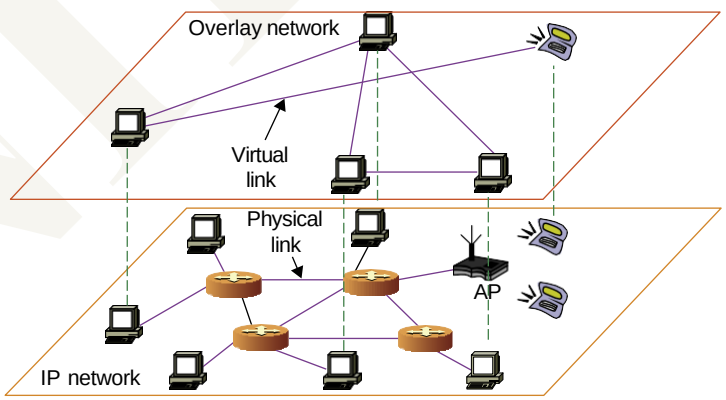

FIGURE 1.17

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

(*Courtesy of Zhenyu Li, Institute of Computing Technology, Chinese Academy of Sciences, 2010*)

formed by mapping each physical machine with its ID, logically, through a virtual mapping as shown in Figure 1.17. 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.

1.3.3.3 P2P Application Families

Based on application, P2P networks are classified into four groups, as shown in Table 1.5. The first family is for distributed file sharing of digital contents (music, videos, etc.) on the P2P network. This includes many popular P2P networks such as Gnutella, Napster, and BitTorrent, among others. Collaboration P2P networks include MSN or Skype chatting, instant messaging, and collaborative design, among others. The third family is for distributed P2P computing in specific applications. For example, SETI@home provides 25 Tflops of distributed computing power, collectively, over 3 million Internet host machines. Other P2P platforms, such as JXTA, .NET, and FightingAID@home, support naming, discovery, communication, security, and resource aggregation in some P2P applications. We will discuss these topics in more detail in Chapters 8 and 9.

1.3.3.4 P2P Computing Challenges

P2P computing faces three types of heterogeneity problems in hardware, software, and network requirements. There are too many hardware models and architectures to select from; incompat- ibility exists between software and the OS; and different network connections and protocols

Table 1.5 Major Categories of P2P Network Families [46]

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

make it too complex to apply in real applications. We need system scalability as the workload increases. System scaling is directly related to performance and bandwidth. P2P networks do have these properties. Data location is also important to affect collective performance. Data locality, network proximity, and interoperability are three design objectives in distributed P2P applications.

P2P performance is affected by routing efficiency and self-organization by participating peers. Fault tolerance, failure management, and load balancing are other important issues in using overlay networks. Lack of trust among peers poses another problem. Peers are strangers to one another. Security, privacy, and copyright violations are major worries by those in the industry in terms of applying P2P technology in business applications [35]. In a P2P network, all clients provide resources including computing power, storage space, and I/O bandwidth. The distributed nature of P2P net- works also increases robustness, because limited peer failures do not form a single point of failure.

By replicating data in multiple peers, one can easily lose data in failed nodes. On the other hand, disadvantages of P2P networks do exist. Because the system is not centralized, managing it is difficult. In addition, the system lacks security. Anyone can log on to the system and cause damage or abuse. Further, all client computers connected to a P2P network cannot be considered reliable or virus-free. In summary, P2P networks are reliable for a small number of peer nodes. They are only useful for applications that require a low level of security and have no concern for data sensitivity. We will discuss P2P networks in Chapter 8, and extending P2P technology to social networking in Chapter 9.

1.3.4 Cloud Computing over the Internet

Gordon Bell, Jim Gray, and Alex Szalay [5] have advocated: "Computational science is changing to be data-intensive. Supercomputers must be balanced systems, not just CPU farms but also petascale I/O and networking arrays." In the future, working with large data sets will typically mean sending the computations (programs) to the data, rather than copying the data to the workstations. This reflects the trend in IT of moving computing and data from desktops to large data centers, where there is on-demand provision of software, hardware, and data as a service. This data explosion has promoted the idea of cloud computing.

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. 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.

1.3.4.1 Internet Clouds

Cloud computing applies a virtualized platform with elastic resources on demand by provisioning hardware, software, and data sets dynamically (see Figure 1.18). 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

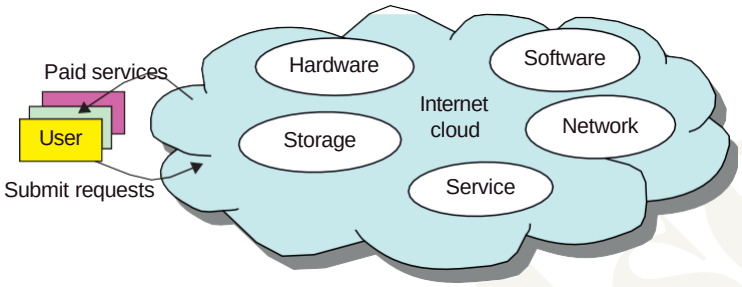

FIGURE 1.18

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.

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.

1.3.4.2 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. Chapters 4, 6, and 9 provide details regarding these cloud service offerings. Chapter 3 covers the relevant virtualization tools.

- Infrastructure as a Service (IaaS) This model puts together infrastructures demanded by users—namely servers, storage, networks, and the data center fabric. The user can deploy and run on multiple VMs running guest OSes on specific applications. The user does not manage or control the underlying cloud infrastructure, but can specify when to request and release the needed resources.
- Platform as a Service (PaaS) This model enables the user to deploy user-built applications onto a virtualized cloud platform. PaaS includes middleware, databases, development tools, and some runtime support such as Web 2.0 and Java. The platform includes both hardware and software integrated with specific programming interfaces. The provider supplies the API and software tools (e.g., Java, Python, Web 2.0, .NET). The user is freed from managing the cloud infrastructure.
- Software as a Service (SaaS) This refers to browser-initiated application software over thousands of paid cloud customers. The SaaS model applies to business processes, industry applications, *consumer relationship management (CRM), enterprise resources planning (ERP), human resources (HR)*, and collaborative applications. On the customer side, there is no upfront investment in servers or software licensing. On the provider side, costs are rather low, compared with conventional hosting of user applications.

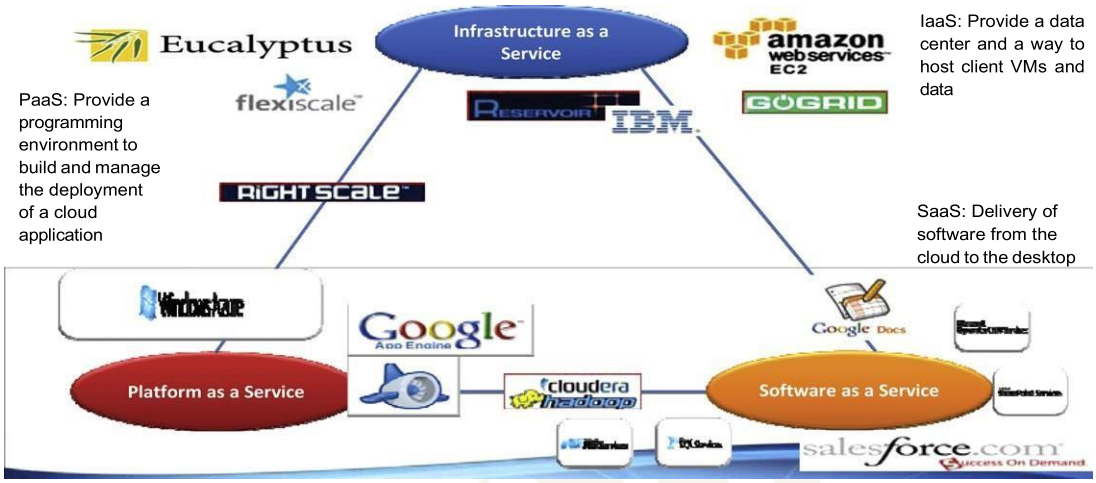

FIGURE 1.19

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

(*Courtesy of Dennis Gannon, keynote address at Cloudcom2010* [19])

Internet clouds offer four deployment modes: *private*, *public*, *managed*, and *hybrid* [11]. 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.

In Chapter 4, we will describe major cloud platforms that have been built and various cloud services offerings. 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

1.4 SOFTWARE ENVIRONMENTS FOR DISTRIBUTED SYSTEMS AND CLOUDS

This section introduces popular software environments for using distributed and cloud computing systems. Chapters 5 and 6 discuss this subject in more depth.

1.4.1 Service-Oriented Architecture (SOA)

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

1.4.1.1 Layered Architecture for Web Services and Grids

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

In the case of fault tolerance, the features in the *Web Services Reliable Messaging (WSRM)* framework mimic the OSI layer capability (as in TCP fault tolerance) modified to match the differ- ent abstractions (such as messages versus packets, virtualized addressing) at the entity levels. Secur- ity is a critical capability that either uses or reimplements the capabilities seen in concepts such as *Internet Protocol Security* (IPsec) and secure sockets in the OSI layers. Entity communication is supported by higher level services for registries, metadata, and management of the entities discussed in Section 5.4.

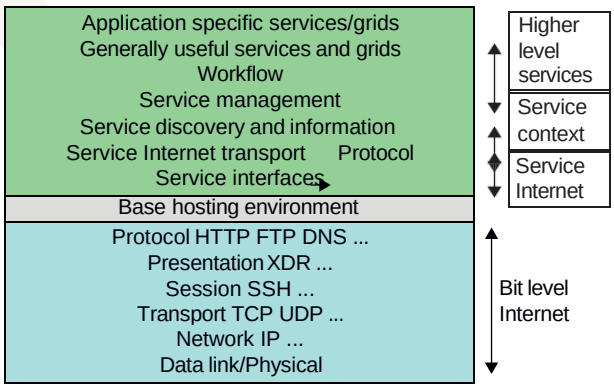

FIGURE 1.20

Layered achitecture for web services and the grids.

Here, one might get several models with, for example, JNDI (*Jini and Java Naming and Directory Interface*) illustrating different approaches within the Java distributed object model. The CORBA Trading Service, UDDI (*Universal Description, Discovery, and Integration*), LDAP (*Lightweight Directory Access Protocol*), and ebXML (*Electronic Business using eXtensible Markup Language*) are other examples of discovery and information services described in Section 5.4. Management services include service state and lifetime support; examples include the CORBA Life Cycle and Persistent states, the different Enterprise JavaBeans models, Jini's lifetime model, and a suite of web services specifications in Chapter 5. The above language or interface terms form a collection of entity-level capabilities.

The latter can have performance advantages and offers a "shared memory" model allowing more convenient exchange of information. However, the distributed model has two critical advantages: namely, higher performance (from multiple CPUs when communication is unimportant) and a cleaner separation of software functions with clear software reuse and maintenance advantages. The distributed model is expected to gain popularity as the default approach to software systems. In the earlier years, CORBA and Java approaches were used in distributed systems rather than today's SOAP, XML, or REST (*Representational State Transfer*).

1.4.1.2 Web Services and Tools

Loose coupling and support of heterogeneous implementations make services more attractive than distributed objects. Figure 1.20 corresponds to two choices of service architecture: web services or REST systems (these are further discussed in Chapter 5). Both web services and REST systems have very distinct approaches to building reliable interoperable systems. In web services, one aims to fully specify all aspects of the service and its environment. This specification is carried with communicated messages using Simple Object Access Protocol (SOAP). The hosting environment then becomes a universal distributed operating system with fully distributed capability carried by SOAP messages. This approach has mixed success as it has been hard to agree on key parts of the protocol and even harder to efficiently implement the protocol by software such as Apache Axis.

In the REST approach, one adopts simplicity as the universal principle and delegates most of the difficult problems to application (implementation-specific) software. 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 appro- priate 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 applica- tion). 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. Above the communication and management layers, we have the ability to compose new entities or distributed programs by integrating several entities together.

In CORBA and Java, the distributed entities are linked with RPCs, and the simplest way to build composite applications is to view the entities as objects and use the traditional ways of linking them together. For Java, this could be as simple as writing a Java program with method calls replaced by Remote Method Invocation (RMI), while CORBA supports a similar model with a syntax reflecting the C++ style of its entity (object) interfaces. Allowing the term "grid" to refer to a single service or to represent a collection of services, here sensors represent entities that output data (as messages), and grids and clouds represent collections of services that have multiple message-based inputs and outputs.

1.4.1.3 The Evolution of SOA

As shown in Figure 1.21, *service-oriented architecture (SOA)* has evolved over the years. SOA applies to building grids, clouds, grids of clouds, clouds of grids, clouds of clouds (also known as interclouds), and systems of systems in general. A large number of sensors provide data-collection services, denoted in the figure as *SS (sensor service)*. A sensor can be a ZigBee device, a Bluetooth device, a WiFi access point, a personal computer, a GPA, or a wireless phone, among other things. Raw data is collected by sensor services. All the SS devices interact with large or small computers, many forms of grids, databases, the compute cloud, the storage cloud, the filter cloud, the discovery cloud, and so on. *Filter services* (*fs* in the figure) are used to eliminate unwanted raw data, in order to respond to specific requests from the web, the grid, or web services.

A collection of filter services forms a filter cloud. We will cover various clouds for compute, storage, filter, and discovery in Chapters 4, 5, and 6, and various grids, P2P networks, and the IoT in Chapters 7, 8, and 9. SOA aims to search for, or sort out, the useful data from the massive

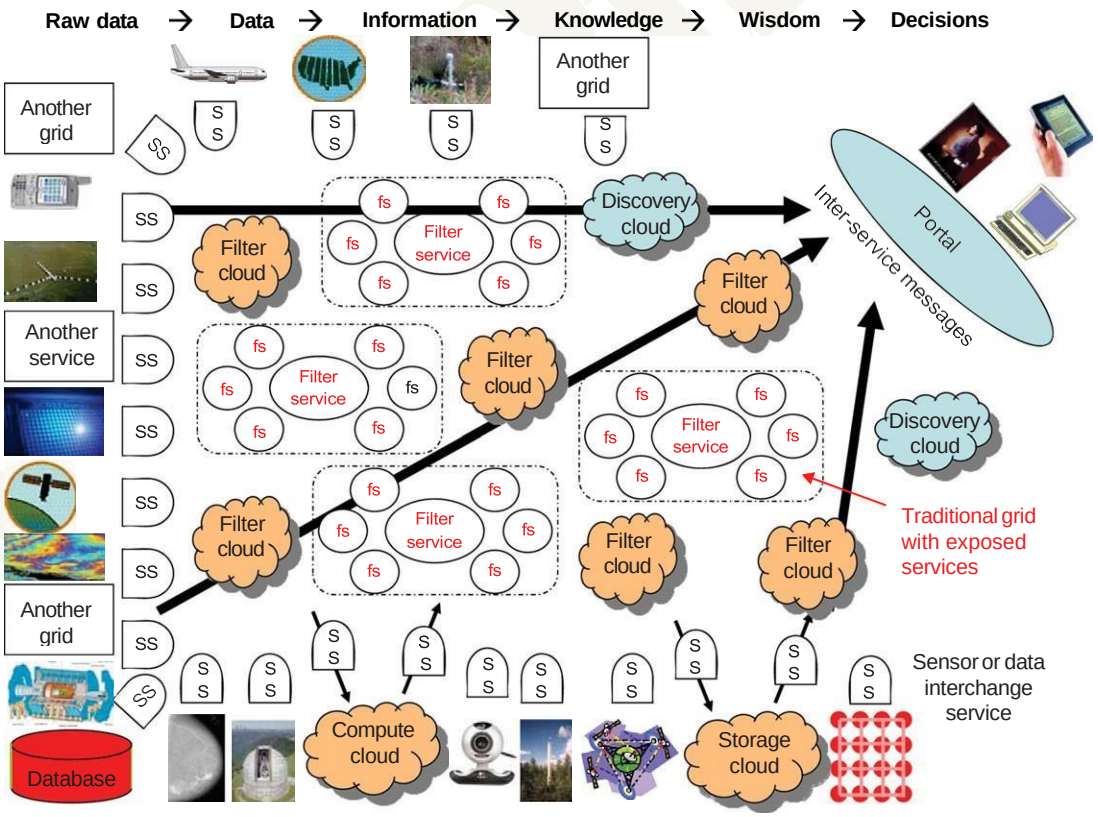

FIGURE 1.21

The evolution of SOA: grids of clouds and grids, where "SS" refers to a sensor service and "fs" to a filter or transforming service.

amounts of raw data items. Processing this data will generate useful information, and subsequently, the knowledge for our daily use. In fact, wisdom or intelligence is sorted out of large knowledge bases. Finally, we make intelligent decisions based on both biological and machine wisdom. Read- ers will see these structures more clearly in subsequent chapters.

Most distributed systems require a web interface or portal. For raw data collected by a large number of sensors to be transformed into useful information or knowledge, the data stream may go through a sequence of compute, storage, filter, and discovery clouds. Finally, the inter-service mes- sages converge at the portal, which is accessed by all users. Two example portals, OGFCE and HUBzero, are described in Section 5.3 using both web service (portlet) and Web 2.0 (gadget) tech- nologies. Many distributed programming models are also built on top of these basic constructs.

1.4.1.4 Grids versus Clouds

The boundary between grids and clouds are getting blurred in recent years. For web services, work- flow technologies are used to coordinate or orchestrate services with certain specifications used to define critical business process models such as two-phase transactions. Section 5.2 discusses the gen- eral approach used in workflow, the BPEL Web Service standard, and several important workflow approaches including Pegasus, Taverna, Kepler, Trident, and Swift. In all approaches, one is building a collection of services which together tackle all or part of a distributed computing problem.

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.

1.4.2 Trends toward Distributed Operating Systems

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

1.4.2.1 Distributed Operating Systems

Tanenbaum [26] identifies three approaches for distributing resource management functions in a distributed computer system. The first approach is to build a *network OS* over a large number of heterogeneous OS platforms. Such an OS offers the lowest transparency to users, and is essentially a distributed file system, with independent computers relying on file sharing as a means of communication. The second approach is to develop middleware to offer a limited degree of resource sharing, similar to the MOSIX/OS developed for clustered systems (see Section 2.4.4). The third approach is to develop a truly *distributed OS* to achieve higher use or system transparency. Table 1.6 compares the functionalities of these three distributed operating systems.

Table 1.6 Feature Comparison of Three Distributed Operating Systems

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

1.4.2.2 Amoeba versus DCE

DCE is a middleware-based system for distributed computing environments. The Amoeba was academically developed at Free University in the Netherlands. The Open Software Foundation (OSF) has pushed the use of DCE for distributed computing. However, the Amoeba, DCE, and MOSIX2 are still research prototypes that are primarily used in academia. No successful commer- cial OS products followed these research systems.

We need new web-based operating systems to support virtualization of resources in distributed environments. This is still a wide-open area of research. To balance the resource management workload, the functionalities of such a distributed OS should be distributed to any available server. In this sense, the conventional OS runs only on a centralized platform. With the distribution of OS services, the distributed OS design should take a lightweight microkernel approach like the Amoeba [46], or should extend an existing OS like the DCE [7] by extending UNIX. The trend is to free users from most resource management duties.

1.4.2.3 MOSIX2 for Linux Clusters

MOSIX2 is a distributed OS [3], which runs with a virtualization layer in the Linux environment. This layer provides a partial *single-system image* to user applications. MOSIX2 supports both sequential and parallel applications, and discovers resources and migrates software processes among Linux nodes. MOSIX2 can manage a Linux cluster or a grid of multiple clusters. Flexible management

of a grid allows owners of clusters to share their computational resources among multiple cluster owners. A MOSIX-enabled grid can extend indefinitely as long as trust exists among the cluster owners. The MOSIX2 is being explored for managing resources in all sorts of clusters, including Linux clusters, GPU clusters, grids, and even clouds if VMs are used. We will study MOSIX and its applications in Section 2.4.4.

1.4.2.4 Transparency in Programming Environments

Figure 1.22 shows the concept of a transparent computing infrastructure for future computing platforms. The user data, applications, OS, and hardware are separated into four levels. Data is owned by users, independent of the applications. The OS provides clear interfaces, standard programming interfaces, or system calls to application programmers. In future cloud infrastruc- ture, the hardware will be separated by standard interfaces from the OS. Thus, users will be able to choose from different OSes on top of the hardware devices they prefer to use. To sepa- rate user data from specific application programs, users can enable cloud applications as SaaS. Thus, users can switch among different services. The data will not be bound to specific applications.

1.4.3 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. As we will discuss, MPI is the most popular programming model for message-passing systems. Google's MapReduce and BigTable are for

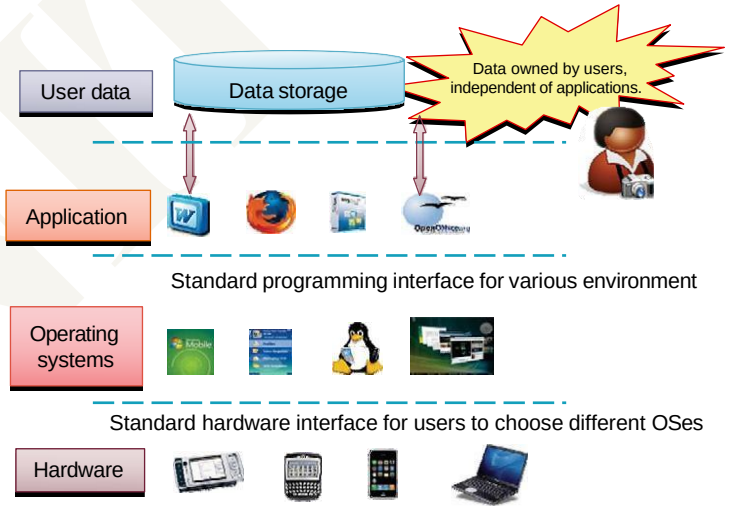

FIGURE 1.22

A transparent computing environment that separates the user data, application, OS, and hardware in time and space – an ideal model for cloud computing.

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

effective use of resources from Internet clouds and data centers. Service clouds demand extending Hadoop, EC2, and S3 to facilitate distributed computing over distributed storage systems. Many other models have also been proposed or developed in the past. In Chapters 5 and 6, we will dis- cuss parallel and distributed programming in more details.

1.4.3.1 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 applica- tions. 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 [28].

1.4.3.2 MapReduce

This is a web programming model for scalable data processing on large clusters over large data sets [16]. 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 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 MapRe- duce jobs are executed on Google's clusters every day.

1.4.3.3 Hadoop Library

Hadoop offers a software platform that was originally developed by a Yahoo! group. The pack- age 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

Table 1.8 Grid Standards and Toolkits for Scientific and Engineering Applications [6]

Standards	Service Functionalities	Key Features and Security Infrastructure
OGSA Standard	Open Grid Services Architecture; offers common grid service standards for general public use	Supports a heterogeneous distributed environment, bridging CAs, multiple trusted intermediaries, dynamic policies, multiple security mechanisms, etc.
Globus Toolkits	Resource allocation, Globus security infrastructure (GSI), and generic security service API	Sign-in multisite authentication with PKI, Kerberos, SSL, Proxy, delegation, and GSS API for message integrity and confidentiality
IBM Grid Toolbox	AIX and Linux grids built on top of Globus Toolkit, autonomic computing, replica services	Uses simple CA, grants access, grid service (ReGS), supports grid application for Java (GAF4J), GridMap in IntraGrid for security update

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 auto- matically keeps multiple data copies to facilitate redeployment of computing tasks upon unex- pected system failures.

1.4.3.4 Open Grid Services Architecture (OGSA)

The development of grid infrastructure is driven by large-scale distributed computing applications. These applications must count on a high degree of resource and data sharing. Table 1.8 introduces OGSA as a common standard for general public use of grid services. Genesis II is a realization of OGSA. Key features include a distributed execution environment, *Public Key Infrastructure (PKI)* services using a local *certificate authority (CA)*, trust management, and security policies in grid computing.

1.4.3.5 Globus Toolkits and Extensions

Globus is a middleware library jointly developed by the U.S. Argonne National Laboratory and USC Information Science Institute over the past decade. This library implements some of the OGSA standards for resource discovery, allocation, and security enforcement in a grid environment. The Globus packages support multisite mutual authentication with PKI certificates. The current ver- sion of Globus, GT 4, has been in use since 2008. In addition, IBM has extended Globus for busi- ness applications. We will cover Globus and other grid computing middleware in more detail in Chapter 7.

1.5 PERFORMANCE, SECURITY, AND ENERGY EFFICIENCY

In this section, we will discuss the fundamental design principles along with rules of thumb for building massively distributed computing systems. Coverage includes scalability, availability, pro- gramming models, and security issues in clusters, grids, P2P networks, and Internet clouds.

1.5.1 Performance Metrics and Scalability Analysis

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

1.5.1.1 Performance Metrics

We discussed *CPU speed* in MIPS and *network bandwidth* in Mbps in Section 1.3.1 to estimate processor and network performance. In a distributed system, performance is attributed to a large number of factors. *System throughput* is often measured in MIPS, *Tflops (tera floating-point operations per second)*, or *TPS (transactions per second)*. Other measures include *job response time* and *network latency*. An interconnection network that has low latency and high bandwidth is preferred. System overhead is often attributed to OS boot time, compile time, I/O data rate, and the runtime support system used. Other performance-related metrics include the QoS for Internet and web services; *system availability* and *dependability*; and *security resilience* for system defense against network attacks.

1.5.1.2 Dimensions of Scalability

Users want to have a distributed system that can achieve scalable performance. Any resource upgrade in a system should be backward compatible with existing hardware and software resources. Overdesign may not be cost-effective. System scaling can increase or decrease resources depending on many practical factors. The following dimensions of scalability are characterized in parallel and distributed systems:

- Size scalability This refers to achieving higher performance or more functionality by increasing the *machine size*. The word "size" refers to adding processors, cache, memory, storage, or I/O channels. The most obvious way to determine size scalability is to simply count the number of processors installed. Not all parallel computer or distributed architectures are equally size- scalable. For example, the IBM S2 was scaled up to 512 processors in 1997. But in 2008, the IBM BlueGene/L system scaled up to 65,000 processors.
- Software scalability This refers to upgrades in the OS or compilers, adding mathematical and engineering libraries, porting new application software, and installing more user-friendly programming environments. Some software upgrades may not work with large system configurations. Testing and fine-tuning of new software on larger systems is a nontrivial job.
- Application scalability This refers to matching *problem size* scalability with *machine size* scalability. Problem size affects the size of the data set or the workload increase. Instead of increasing machine size, users can enlarge the problem size to enhance system efficiency or cost-effectiveness.
- Technology scalability This refers to a system that can adapt to changes in building tech- nologies, such as the component and networking technologies discussed in Section 3.1. When scaling a system design with new technology one must consider three aspects: *time*, *space*, and *heterogeneity*. (1) Time refers to generation scalability. When changing to new-generation pro- cessors, one must consider the impact to the motherboard, power supply, packaging and cooling, and so forth. Based on past experience, most systems upgrade their commodity processors every three to five years. (2) Space is related to packaging and energy concerns. Technology scalabil- ity demands harmony and portability among suppliers. (3) Heterogeneity refers to the use of hardware components or software packages from different vendors. Heterogeneity may limit the scalability.

1.5.1.3 Scalability versus OS Image Count

In Figure 1.23, *scalable performance* is estimated against the *multiplicity of OS images* in distribu- ted systems deployed up to 2010. Scalable performance implies that the system can achieve higher speed by adding more processors or servers, enlarging the physical node's memory size, extending the disk capacity, or adding more I/O channels. The OS image is counted by the number of inde- pendent OS images observed in a cluster, grid, P2P network, or the cloud. SMP and NUMA are included in the comparison. An *SMP (symmetric multiprocessor)* server has a single system image, which could be a single node in a large cluster. By 2010 standards, the largest shared-memory SMP node was limited to a few hundred processors. The scalability of SMP systems is constrained primarily by packaging and the system interconnect used.

NUMA (nonuniform memory access) machines are often made out of SMP nodes with distribu- ted, shared memory. A NUMA machine can run with multiple operating systems, and can scale to a few thousand processors communicating with the MPI library. For example, a NUMA machine may have 2,048 processors running 32 SMP operating systems, resulting in 32 OS images in the 2,048-processor NUMA system. The cluster nodes can be either SMP servers or high-end machines that are loosely coupled together. Therefore, clusters have much higher scalability than NUMA machines. The number of OS images in a cluster is based on the cluster nodes concurrently in use. The cloud could be a virtualized cluster. As of 2010, the largest cloud was able to scale up to a few thousand VMs.

Keeping in mind that many cluster nodes are SMP or multicore servers, the total number of pro- cessors or cores in a cluster system is one or two orders of magnitude greater than the number of OS images running in the cluster. The grid node could be a server cluster, or a mainframe, or a supercomputer, or an MPP. Therefore, the number of OS images in a large grid structure could be hundreds or thousands fewer than the total number of processors in the grid. A P2P network can easily scale to millions of independent peer nodes, essentially desktop machines. P2P performance

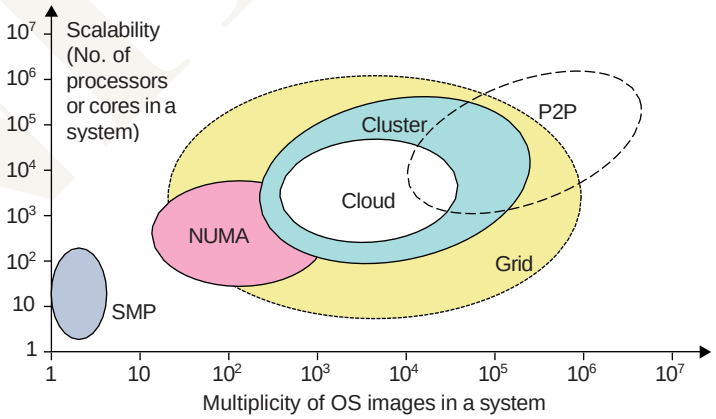

FIGURE 1.23

System scalability versus multiplicity of OS images based on 2010 technology.

depends on the QoS in a public network. Low-speed P2P networks, Internet clouds, and computer clusters should be evaluated at the same networking level.

1.5.1.4 Amdahl's Law

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

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

$$\text{Speedup} = S = T/[\alpha T + (1-\alpha)T/n] = 1/[\alpha + (1-\alpha)/n] \tag{1.1}$$

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

1.5.1.5 Problem with Fixed Workload

In Amdahl's law, we have assumed the same amount of workload for both sequential and parallel execution of the program with a fixed problem size or data set. This was called *fixed-workload speedup* by Hwang and Xu [14]. To execute a fixed workload on n processors, parallel processing may lead to a s*ystem efficiency* defined as follows:

$$E = S/n = 1/[\alpha n + 1 - \alpha] \tag{1.2}$$

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

1.5.1.6 Gustafson's Law

To achieve higher efficiency when using a large cluster, we must consider scaling the problem size to match the cluster capability. This leads to the following speedup law proposed by John Gustafson (1988), referred as *scaled-workload speedup* in [14]. Let W be the workload in a given program. When using an n-processor system, the user scales the workload to $W' = \alpha W + (1 - \alpha)nW$. Note that only the parallelizable portion of the workload is scaled n times in the second term. This scaled

workload W' is essentially the sequential execution time on a single processor. The parallel execution time of a scaled workload W' on n processors is defined by a *scaled-workload speedup* as follows:

$$S' = W'/W = [\alpha W + (1-\alpha)nW]/W = \alpha + (1-\alpha)n \quad (1.3)$$

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

$$E' = S'/n = \alpha/n + (1-\alpha) \quad (1.4)$$

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

1.5.2 Fault Tolerance and System Availability

In addition to performance, system availability and application flexibility are two other important design goals in a distributed computing system.

1.5.2.1 System Availability

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

$$\textit{System Availability} = MTTF/(MTTF + MTTR) \quad (1.5)$$

System availability is attributed to many factors. All hardware, software, and network compo- nents may fail. Any failure that will pull down the operation of the entire system is called a *single point of failure*. The rule of thumb is to design a dependable computing system with no single point of failure. Adding hardware redundancy, increasing component reliability, and designing for testability will help to enhance system availability and dependability. In Figure 1.24, the effects on system availability are estimated by scaling the system size in terms of the number of processor cores in the system.

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