CHAPTER

Distributed System Models and Enabling Technologies

1

CHAPTER OUTLINE

SUMMARY

This chapter presents the evolutionary changes that have occurred in parallel, distributed, and cloud computing over the past 30 years, driven by applications with variable workloads and large data sets. We study both high-performance and high-throughput computing systems in parallel computers appearing as computer clusters, service-oriented architecture, computational grids, peer-to-peer networks, Internet clouds, and the Internet of Things. These systems are distinguished by their hardware architectures, OS platforms, processing algorithms, communication protocols, and service models applied. We also introduce essential issues on the scalability, performance, availability, security, and energy efficiency in distributed systems.

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 distributed 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 minicomputers such as the DEC PDP 11 and VAX Series became popular among small businesses and on college campuses.

From 1970 to 1990, we saw widespread use of personal computers built with VLSI microprocessors. From 1980 to 2000, massive numbers of portable computers and pervasive devices appeared in both wired and wireless applications. Since 1990, the use of both HPC and HTC systems hidden in

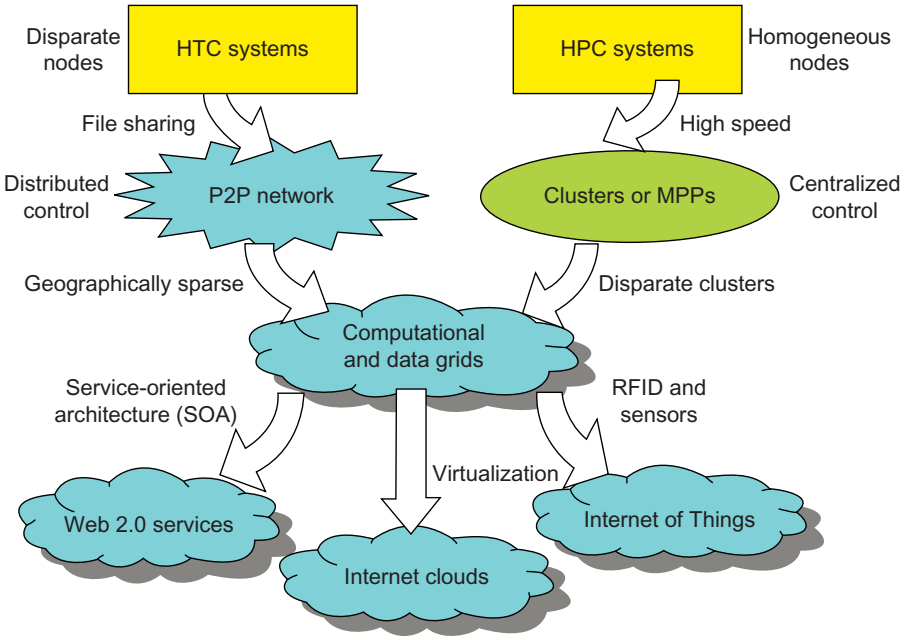

FIGURE 1.1

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

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 generated 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 with 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 computing 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 clusters. 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, superscalar computing, *VLIW (very long instruction word)* architectures, and multithreading. ILP requires branch prediction, dynamic scheduling, speculation, and compiler support to work efficiently.

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

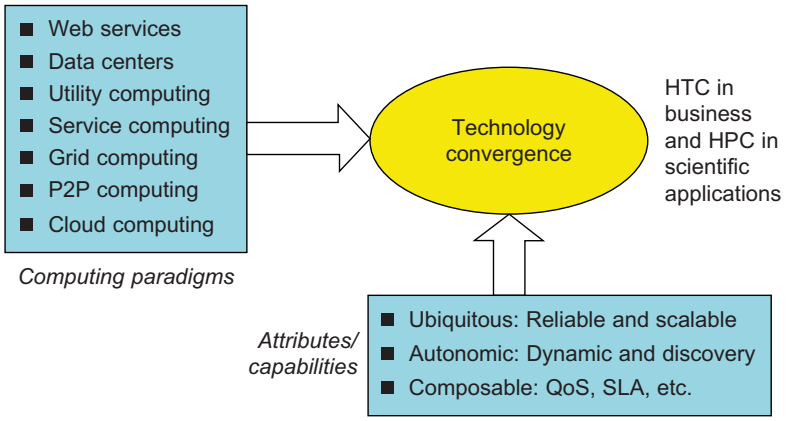

FIGURE 1.2

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 adoption, and *mesh network sensors* were expected to take more than 10 years to move from the inflated expectation stage to a plateau of mainstream adoption.

Also as shown in Figure 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 promising technologies are the clouds, biometric authentication, interactive TV, speech recognition, predictive 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 everyday 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 sensors that interconnect all things in our daily life. These things can be large or small and they vary

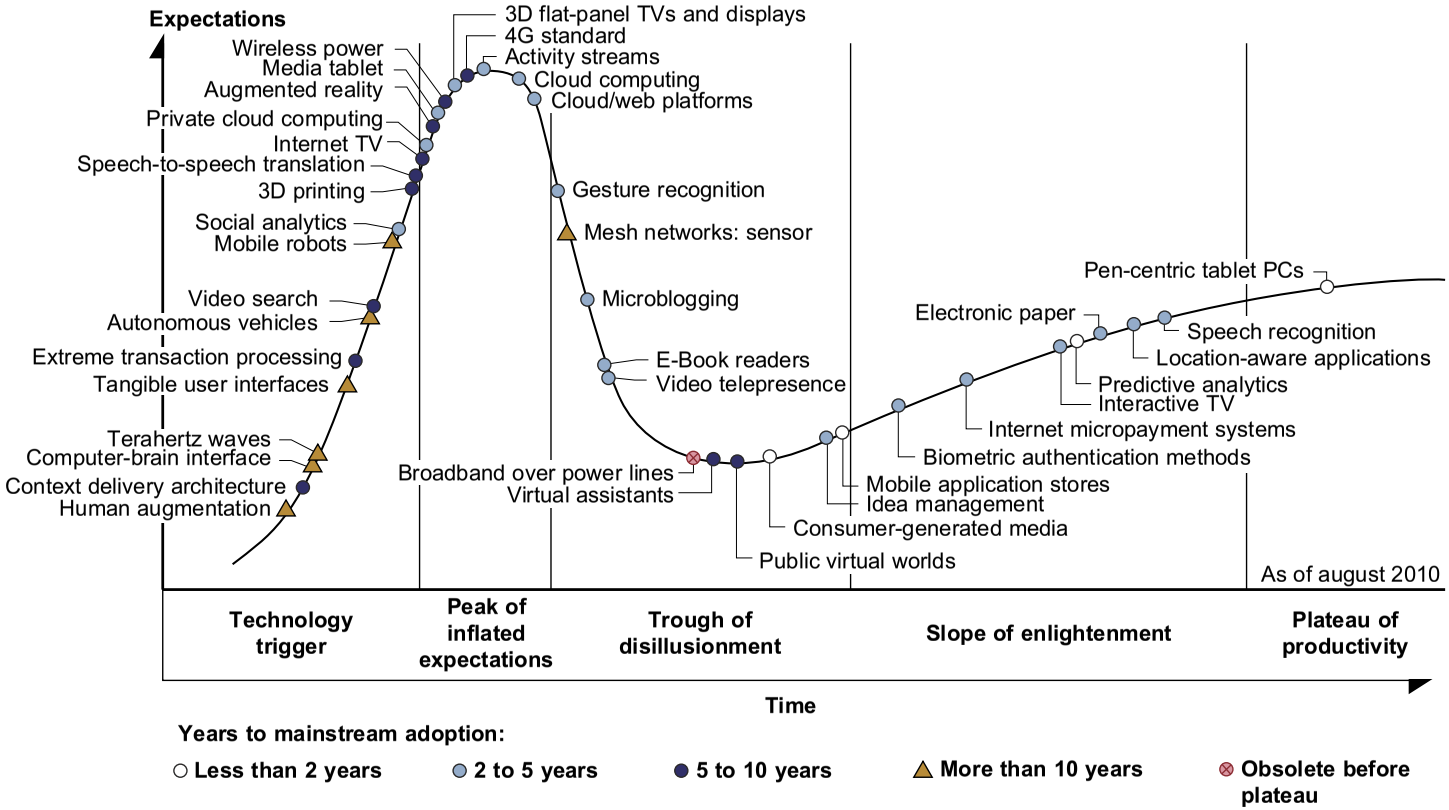

FIGURE 1.3

Hype cycle for Emerging Technologies, 2010.

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 "Gartner'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 addressability of all of the objects or things. To reduce the complexity of identification, search, and storage, one can set the threshold to filter out fine-grain objects. The IoT obviously extends the Internet and is more heavily developed in Asia and European countries.

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

The dynamic connections will grow exponentially into a new dynamic network of networks, called the *Internet of Things* (IoT). The IoT is still in its infancy stage of development. Many prototype 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" (concurrent and information-dense) objects. A CPS merges the "3C" technologies of *computation, communication*, and *control* into an intelligent closed feedback system between the physical world and the information world, a concept which is actively explored in the United States. The IoT emphasizes various networking connections among physical objects, while the CPS emphasizes exploration of *virtual reality* (VR) applications in the physical world. We may transform how we interact with the physical world just like the Internet transformed how we interact with the virtual world. 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 parallelism 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.

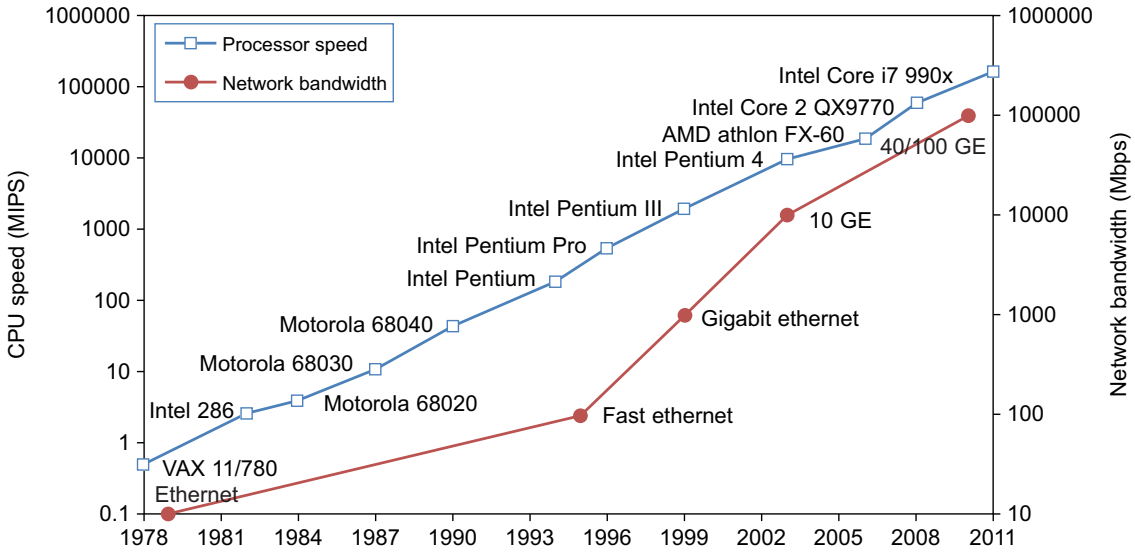

FIGURE 1.4

Improvement in processor and network technologies over 33 years.

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

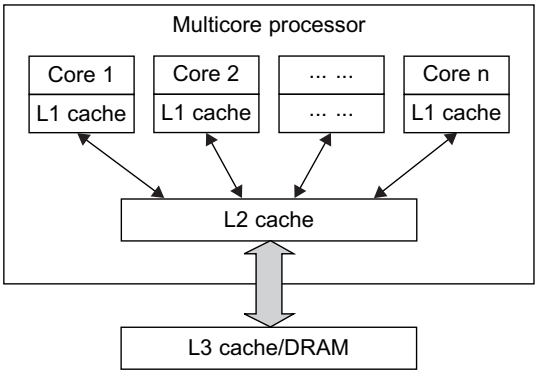

FIGURE 1.5

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 multithreaded CPUs are equipped with many high-end processors, including the Intel i7, Xeon, AMD Opteron, Sun Niagara, IBM Power 6, and X cell processors. Each core could be also multithreaded. For example, the Niagara II is built with eight cores with eight threads handled by each core. This implies that the maximum ILP and TLP that can be exploited in Niagara is 64 ($8 \times 8 = 64$). In 2011, the Intel Core i7 990x has reported 159,000 MIPS execution rate as shown in the uppermost 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

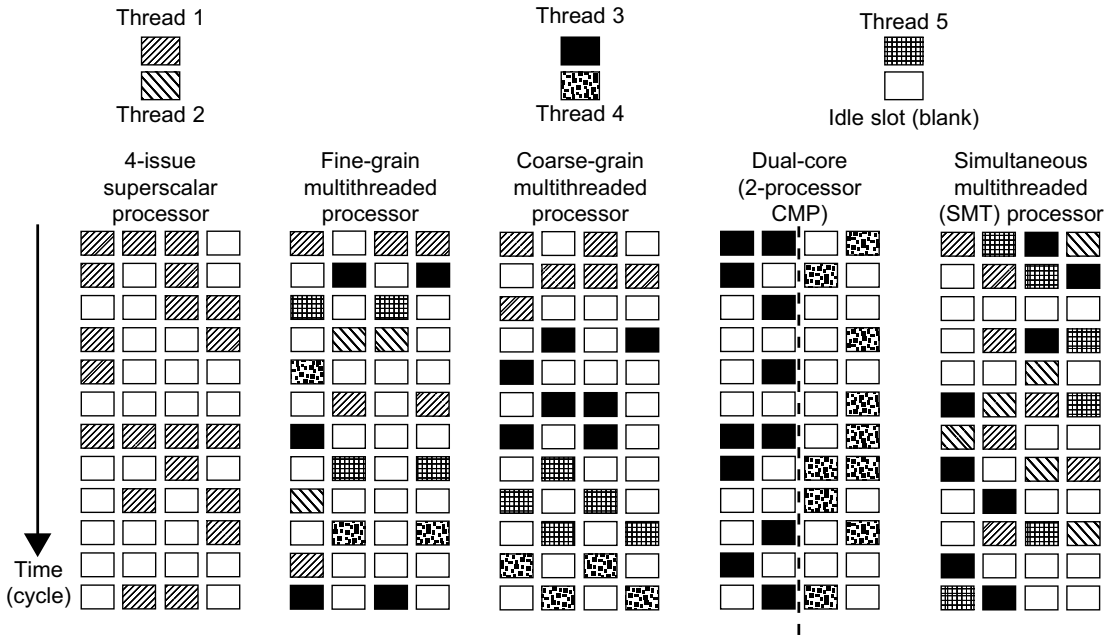

FIGURE 1.6

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 multithreaded processor, a two-core CMP, *and a simultaneous multithreaded* (SMT) *processor*. The superscalar processor is single-threaded with four functional units. Each of the three multithreaded processors is four-way multithreaded over four functional data paths. In the dual-core processor, assume two processing cores, each a single-threaded two-way superscalar processor.

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

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 conventional microprocessor very quickly. Lately, parallel GPUs or GPU clusters have been garnering a lot of attention against the use of CPUs with limited parallelism. *General-purpose computing on GPUs*, known as 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 systems, PCs, and servers. The NVIDIA CUDA Tesla or Fermi is used in GPU clusters or in HPC systems for parallel processing of massive floating-pointing data.

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.

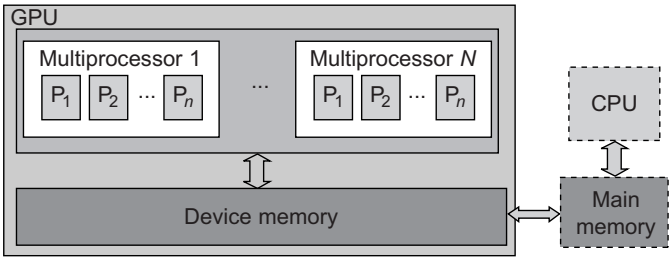

FIGURE 1.7

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

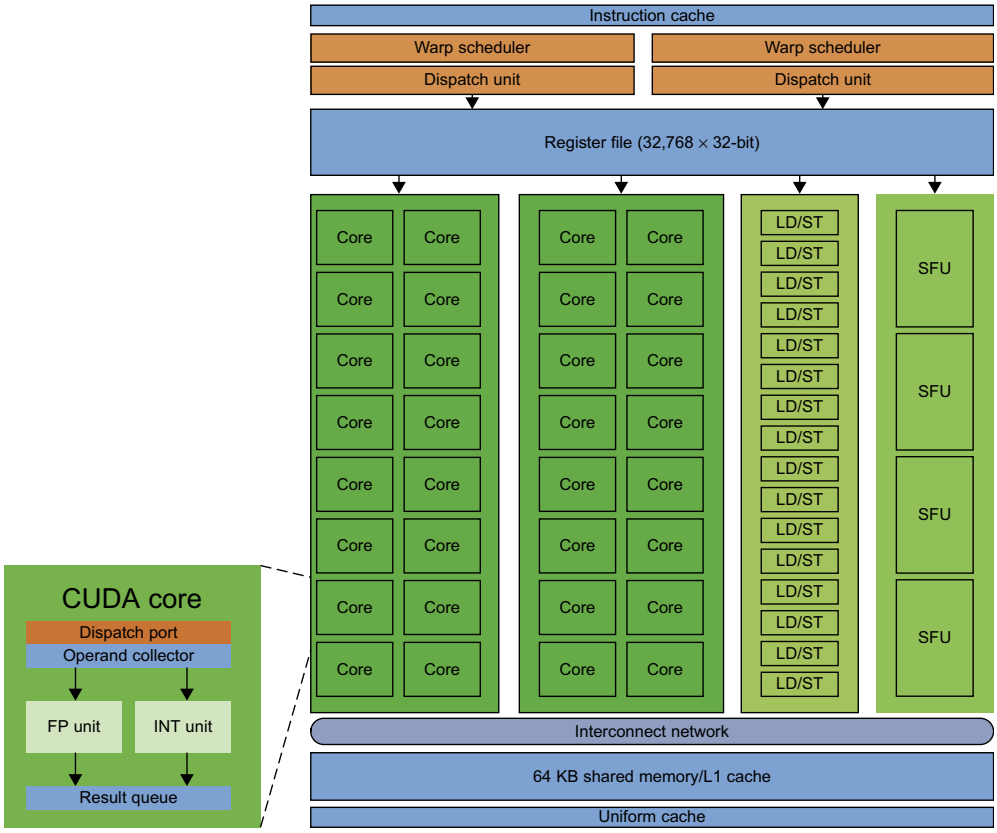

FIGURE 1.8

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

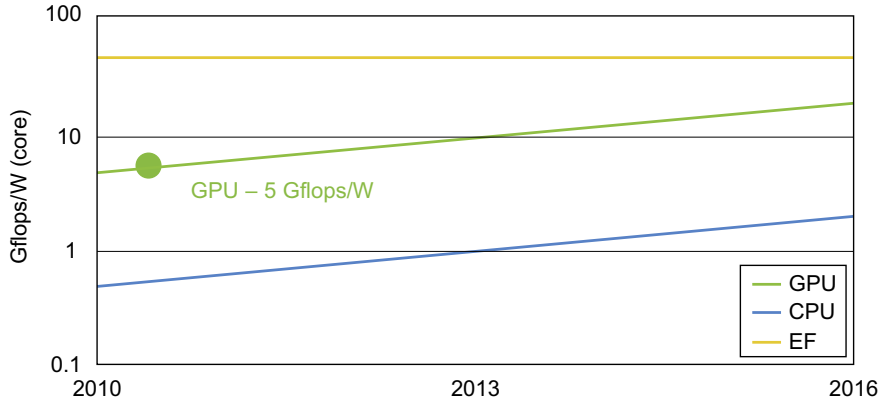

FIGURE 1.9

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 mortality rate of SSD is not bad at all. A typical SSD can handle 300,000 to 1 million write cycles per

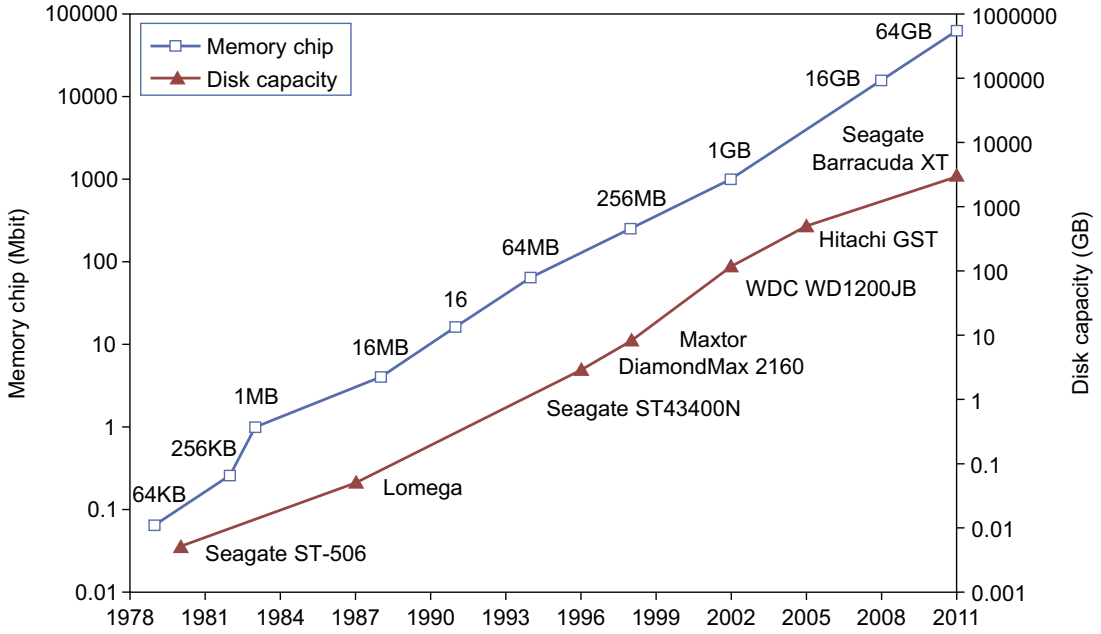

FIGURE 1.10

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 commercially available.

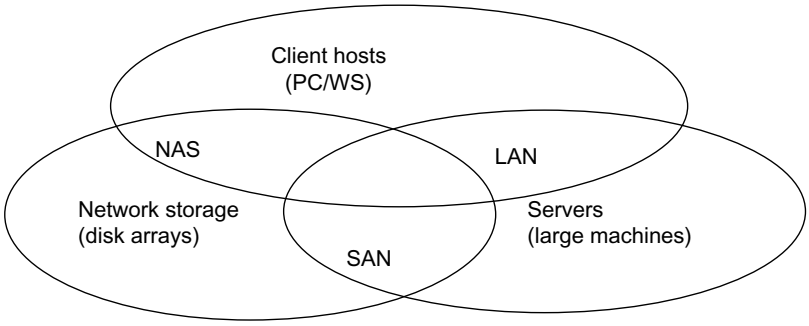

FIGURE 1.11

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.

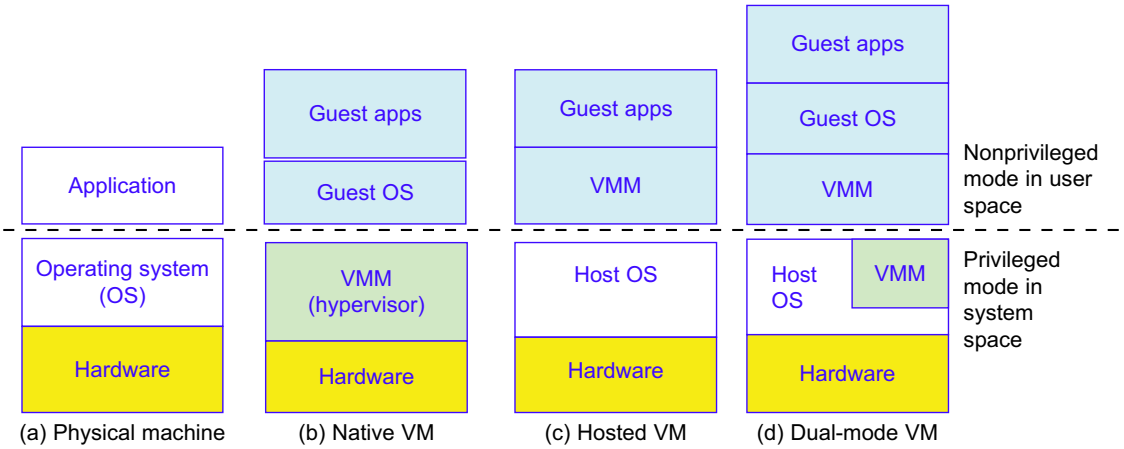

FIGURE 1.12

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 privileged mode. For example, the hardware has x-86 architecture running the Windows system.

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

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.

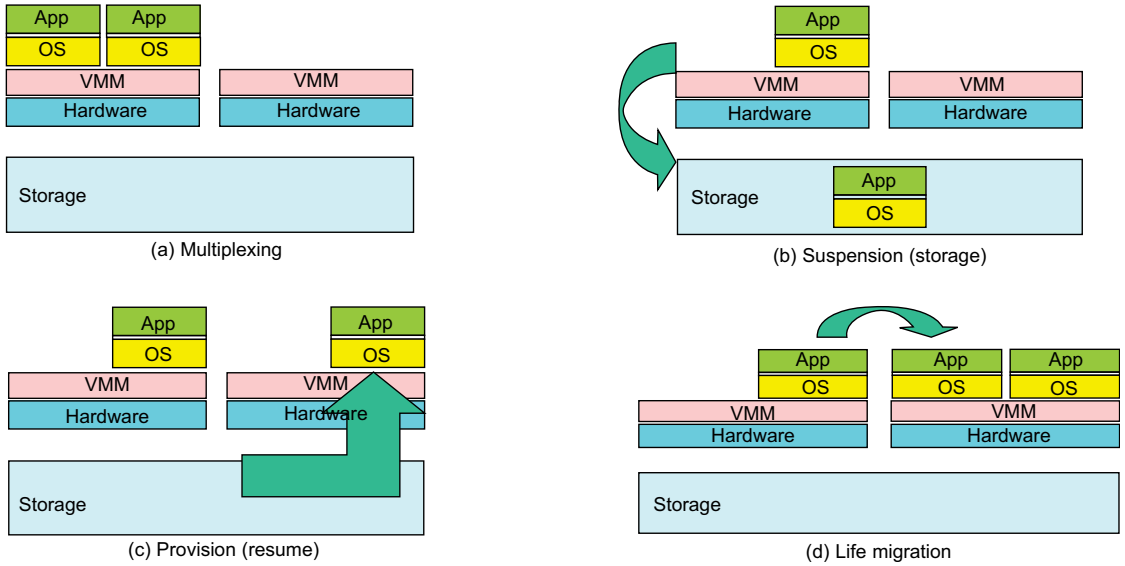

FIGURE 1.13

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.

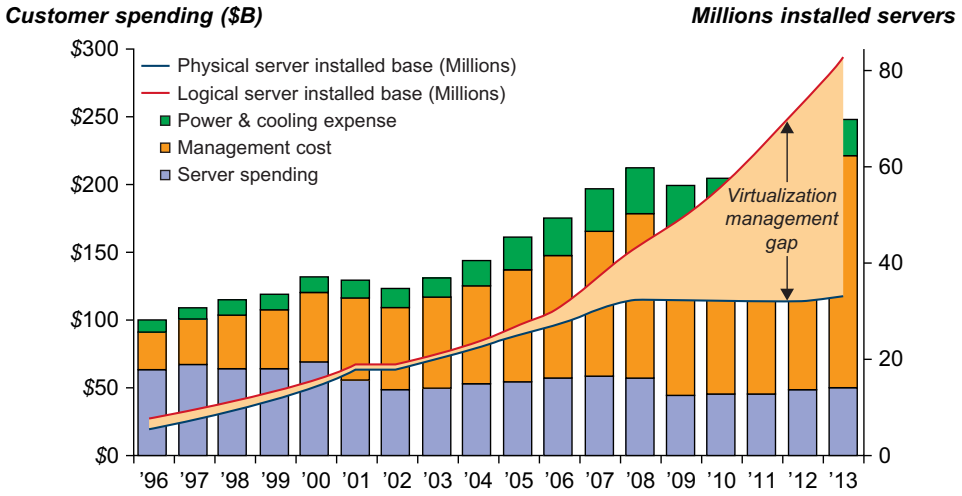

FIGURE 1.14

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 maintenance. The server purchase cost did not increase much with time. The cost of electricity and cooling did increase from 5 percent to 14 percent in 15 years.

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) hardware 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 multicore 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)* recommended 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 transparently 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 commodity 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 computing 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 manner. With today's networking technology, a few LAN switches can easily connect hundreds of machines as a working cluster. A WAN can connect many local clusters to form a very large cluster of clusters. In this sense, one can build a massive system with millions of computers connected to edge networks.

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

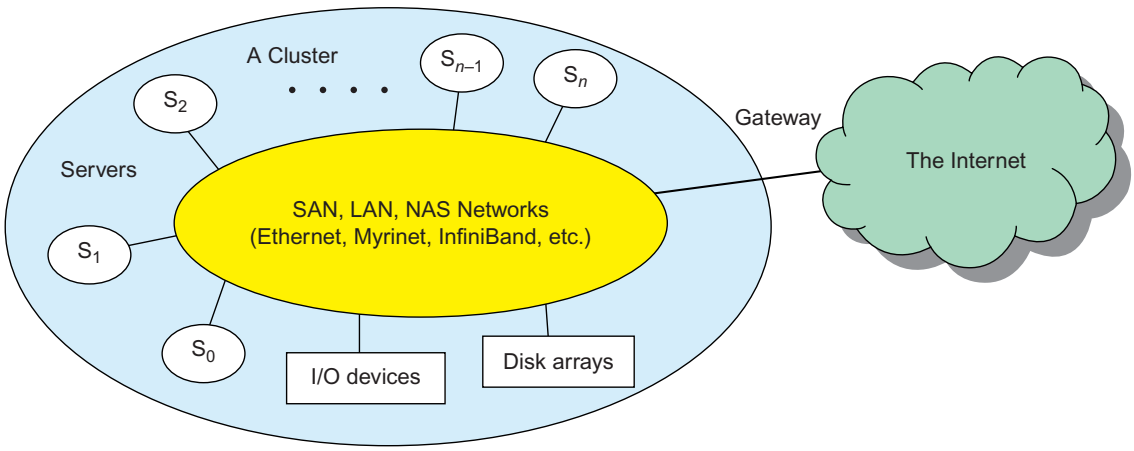

FIGURE 1.15

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 middleware to support SSI at various levels, including the sharing of CPUs, memory, and I/O across all cluster nodes. An SSI is an illusion created by software or hardware that presents a collection of resources as one integrated, powerful resource. SSI makes the cluster appear like a single machine to the user. A cluster with multiple system images is nothing but a collection of independent computers.

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 *distributed shared memory (DSM)*. Many SSI features are expensive or difficult to achieve at various cluster operational levels. Instead of achieving SSI, many clusters are loosely coupled machines. Using virtualization, one can build many virtual clusters dynamically, upon user demand. 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 connect to a remote computer. A web service such as HTTP enables remote access of remote web pages. Grid computing is envisioned to allow close interaction among applications running on distant computers simultaneously. *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 primarily 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 astrophysics@Swineburne for pulsars. At the server end, the grid is a network. At the client end, we see wired or wireless terminal devices. The grid integrates the computing, communication, contents, and transactions as rented services. Enterprises and consumers form the user base, which then defines the usage trends and service characteristics. 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 platform development by IBM, Microsoft, Sun, HP, Dell, Cisco, EMC, Platform Computing, and others. New *grid service providers* (GSPs) and new grid applications have emerged rapidly, similar to the growth of Internet and web services in the past two decades. 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.

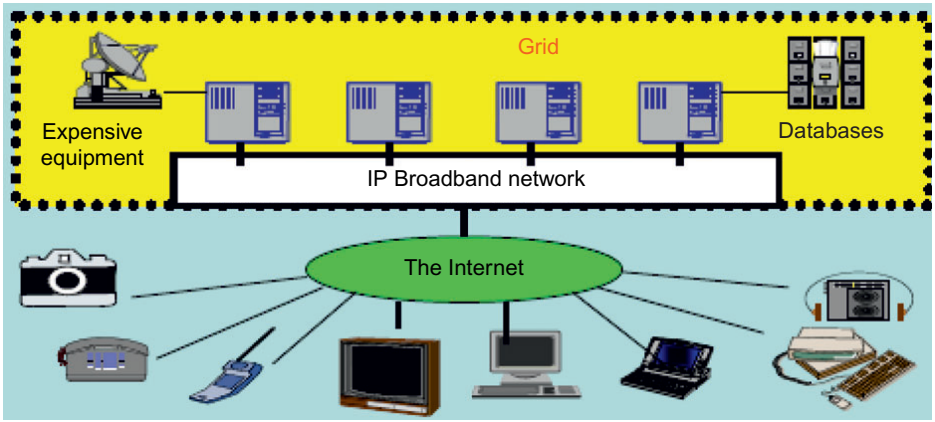

FIGURE 1.16

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

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 distributed 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 participating peers form the *physical network* at any time. Unlike the cluster or grid, a P2P network does not use a dedicated interconnection network. The physical network is simply an ad hoc network formed at various Internet domains randomly using the TCP/IP and NAI protocols. Thus, the physical network varies in size and topology dynamically due to the free membership in the P2P network.

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

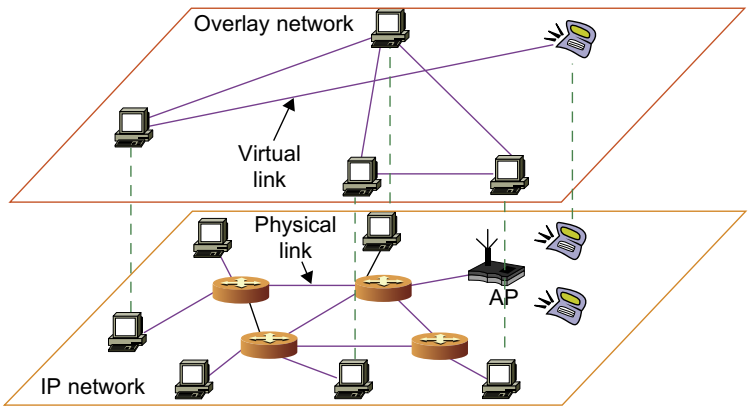

FIGURE 1.17

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; incompatibility 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 networks 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

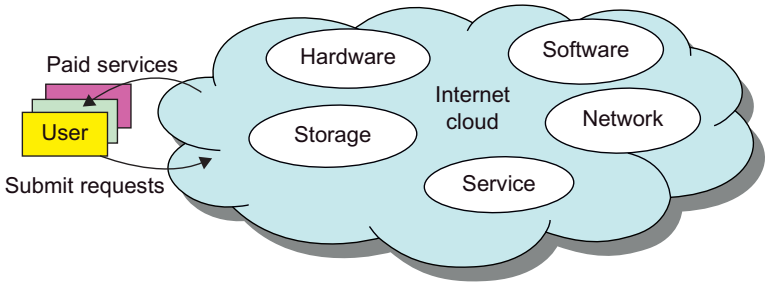

FIGURE 1.18

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.

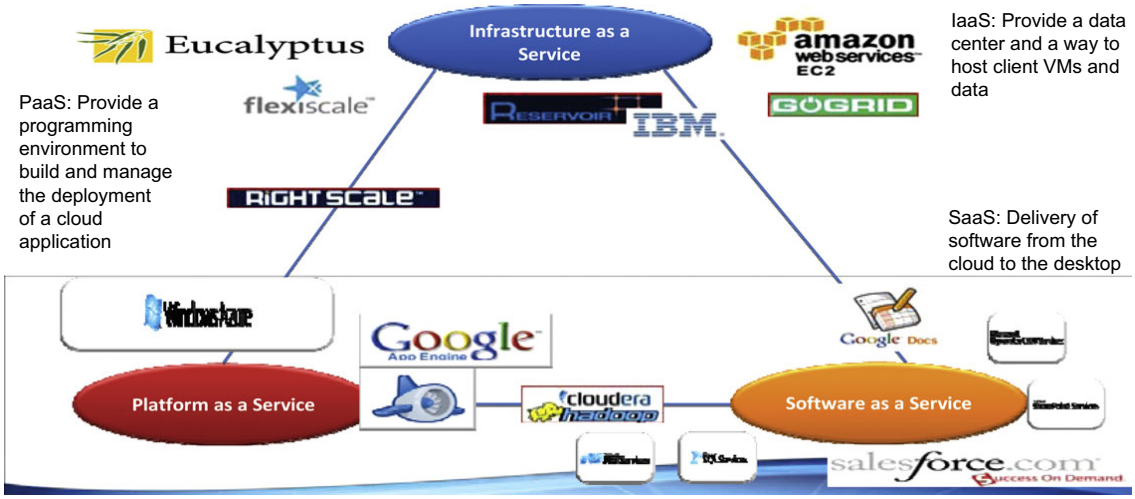

FIGURE 1.19

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 different abstractions (such as messages versus packets, virtualized addressing) at the entity levels. Security is a critical capability that either uses or reimplements the capabilities seen in concepts such as *Internet Protocol Security* (IPsec) and secure sockets in the OSI layers. Entity communication is supported by higher level services for registries, metadata, and management of the entities discussed in Section 5.4.

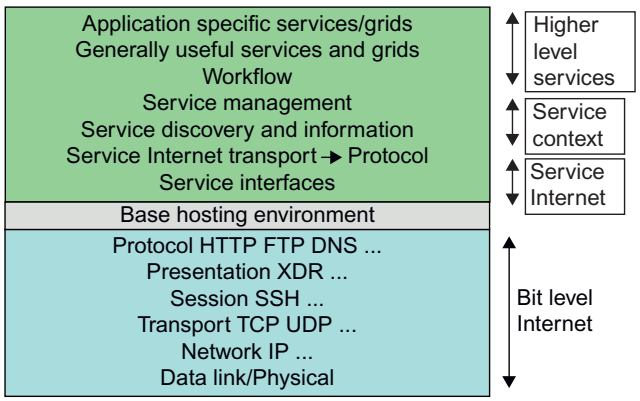

FIGURE 1.20

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 appropriate for rapid technology environments. However, the ideas in web services are important and probably will be required in mature systems at a different level in the stack (as part of the application). Note that REST can use XML schemas but not those that are part of SOAP; "XML over HTTP" is a popular design choice in this regard. 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

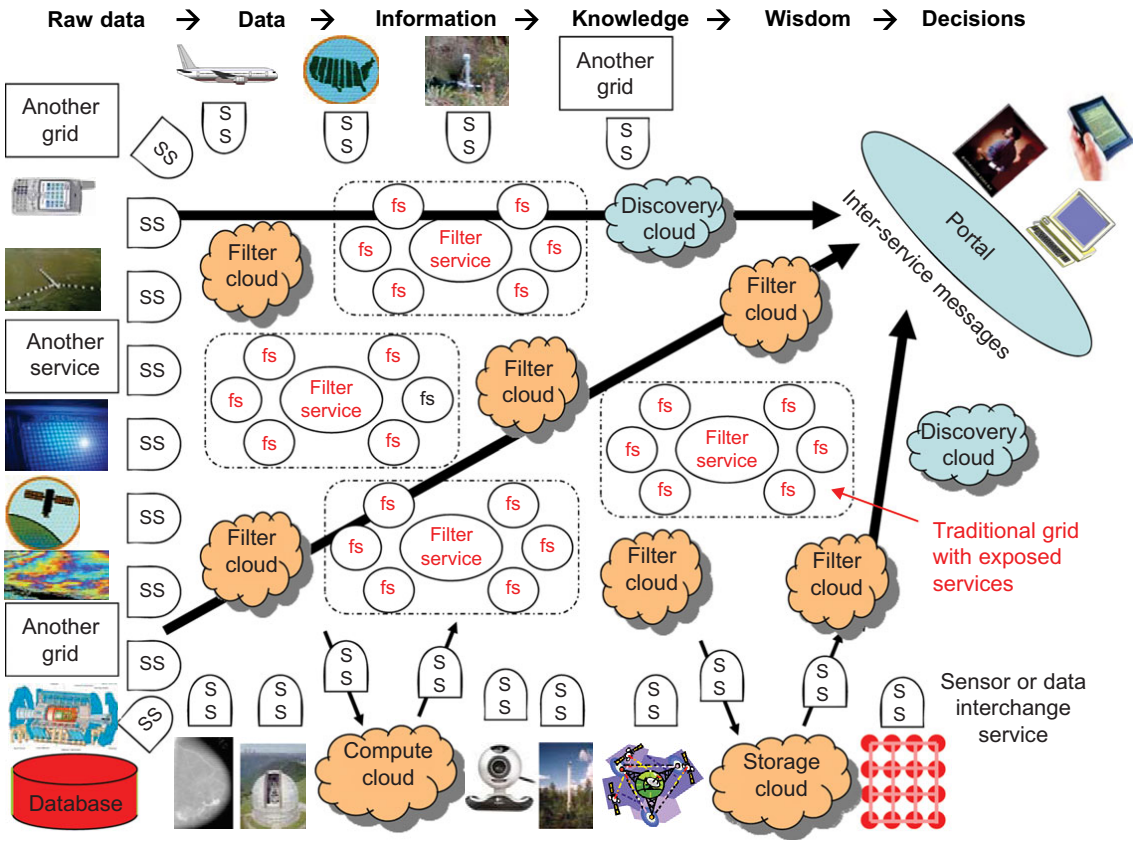

FIGURE 1.21

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

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 commercial 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 infrastructure, 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 separate 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

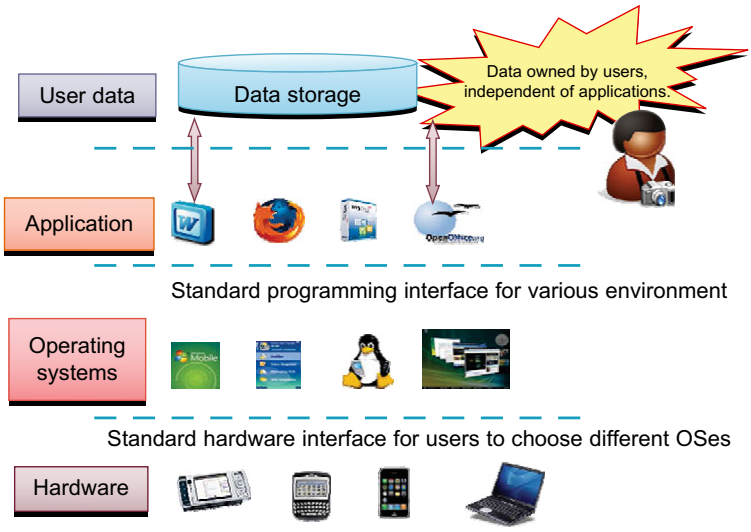

FIGURE 1.22

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 discuss 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 applications. Besides MPI, distributed programming can be also supported with low-level primitives such as the *Parallel Virtual Machine* (PVM). Both MPI and PVM are described in Hwang and Xu [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 MapReduce 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 package enables users to write and run applications over vast amounts of distributed data. Users can easily scale Hadoop to store and process petabytes of data in the web space. Also, Hadoop is economical in that it comes with an open source version of MapReduce that minimizes overhead

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 automatically keeps multiple data copies to facilitate redeployment of computing tasks upon unexpected 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 version of Globus, GT 4, has been in use since 2008. In addition, IBM has extended Globus for business 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, programming 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 technologies, such as the component and networking technologies discussed in Section 3.1. When scaling a system design with new technology one must consider three aspects: *time*, *space*, and *heterogeneity*. (1) Time refers to generation scalability. When changing to new-generation processors, one must consider the impact to the motherboard, power supply, packaging and cooling, and so forth. Based on past experience, most systems upgrade their commodity processors every three to five years. (2) Space is related to packaging and energy concerns. Technology scalability demands harmony and portability among suppliers. (3) Heterogeneity refers to the use of hardware components or software packages from different vendors. Heterogeneity may limit the scalability.

1.5.1.3 Scalability versus OS Image Count

In Figure 1.23, *scalable performance* is estimated against the *multiplicity of OS images* in distributed 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 independent 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 distributed, 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 processors 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

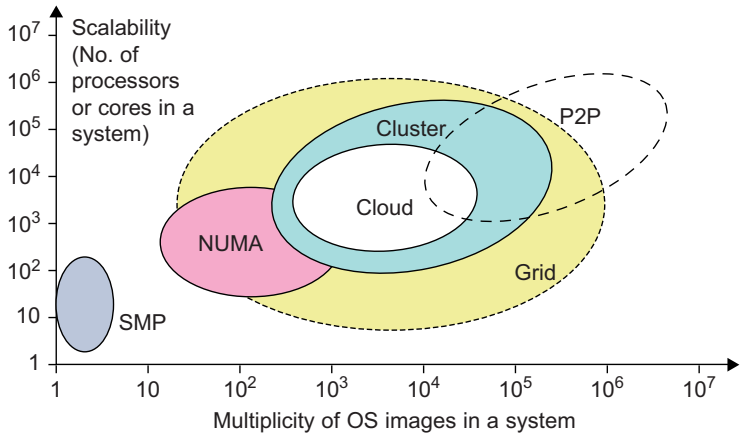

FIGURE 1.23

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 executed sequentially, called the *sequential bottleneck*. Therefore, $(1 - \alpha)$ of the code can be compiled for parallel execution by n processors. The total execution time of the program is calculated by $\alpha T + (1 - \alpha)T/n$, where the first term is the sequential execution time on a single processor and the second term is the parallel execution time on n processing nodes.

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

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

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 independent of the cluster size n. The sequential bottleneck is the portion of the code that cannot be parallelized. For example, the maximum speedup achieved is 4, if $\alpha = 0.25$ or $1 - \alpha = 0.75$, even if one uses hundreds of processors. Amdahl's law teaches us that we should make the sequential bottleneck 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:

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

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

$$E' = S'/n = \alpha/n + (1-\alpha) \tag{1.4}$$

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:

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

System availability is attributed to many factors. All hardware, software, and network components may fail. Any failure that will pull down the operation of the entire system is called a *single point of failure*. The rule of thumb is to design a dependable computing system with no single point of failure. Adding hardware redundancy, increasing component reliability, and designing for testability will help to enhance system availability and dependability. In 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, private clouds are created out of virtualized data centers; hence, a cloud has an estimated availability similar to that of the hosting cluster. A grid is visualized as a hierarchical cluster of clusters. Grids have higher availability due to the isolation of faults. Therefore, clusters, clouds, and grids have decreasing availability as the system increases in size. A P2P file-sharing network has the highest aggregation of client machines. However, it operates independently with low availability, and even many peer nodes depart or fail simultaneously.

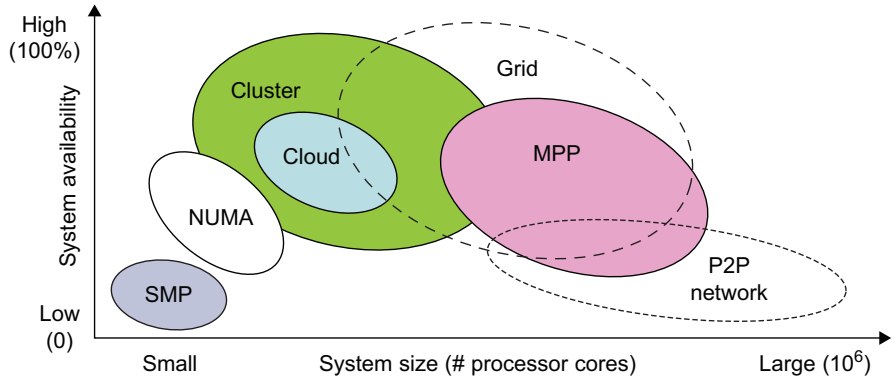

FIGURE 1.24

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

1.5.3 Network Threats and Data Integrity

Clusters, grids, P2P networks, and clouds demand security and copyright protection if they are to be accepted in today's digital society. This section introduces system vulnerability, network threats, defense countermeasures, and copyright protection in distributed or cloud computing systems.

1.5.3.1 Threats to Systems and Networks

Network viruses have threatened many users in widespread attacks. These incidents have created a worm epidemic by pulling down many routers and servers, and are responsible for the loss of billions of dollars in business, government, and services. Figure 1.25 summarizes various attack types and their potential damage to users. As the figure shows, information leaks lead to a loss of confidentiality. Loss of data integrity may be caused by user alteration, Trojan horses, and service spoofing attacks. A *denial of service (DoS)* results in a loss of system operation and Internet connections.

Lack of authentication or authorization leads to attackers' illegitimate use of computing resources. Open resources such as data centers, P2P networks, and grid and cloud infrastructures could become the next targets. Users need to protect clusters, grids, clouds, and P2P systems. Otherwise, users should not use or trust them for outsourced work. Malicious intrusions to these systems may destroy valuable hosts, as well as network and storage resources. Internet anomalies found in routers, gateways, and distributed hosts may hinder the acceptance of these public-resource computing services.

1.5.3.2 Security Responsibilities

Three security requirements are often considered: *confidentiality*, *integrity*, and *availability* for most Internet service providers and cloud users. In the order of SaaS, PaaS, and IaaS, the providers gradually release the responsibility of security control to the cloud users. In summary, the SaaS model relies on the cloud provider to perform all security functions. At the other extreme, the IaaS model wants the users to assume almost all security functions, but to leave availability in the hands of the providers. The PaaS model relies on the provider to maintain data integrity and availability, but burdens the user with confidentiality and privacy control.

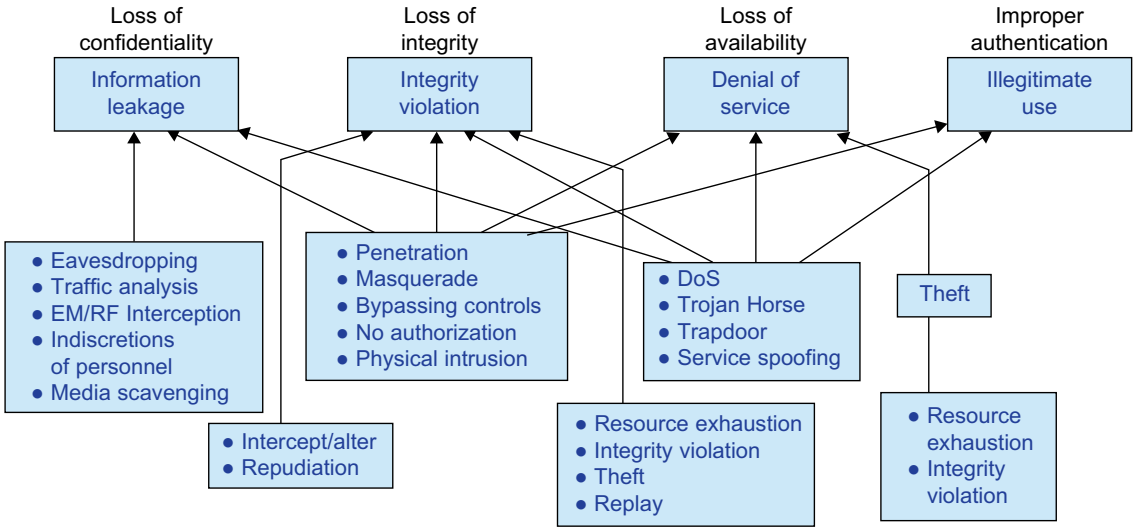

FIGURE 1.25

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

1.5.3.3 Copyright Protection

Collusive piracy is the main source of intellectual property violations within the boundary of a P2P network. Paid clients (colluders) may illegally share copyrighted content files with unpaid clients (pirates). Online piracy has hindered the use of open P2P networks for commercial content delivery. One can develop a proactive content poisoning scheme to stop colluders and pirates from alleged copyright infringements in P2P file sharing. Pirates are detected in a timely manner with identity-based signatures and timestamped tokens. This scheme stops collusive piracy from occurring without hurting legitimate P2P clients. Chapters 4 and 7 cover grid and cloud security, P2P reputation systems, and copyright protection.

1.5.3.4 System Defense Technologies

Three generations of network defense technologies have appeared in the past. In the first generation, tools were designed to prevent or avoid intrusions. These tools usually manifested themselves as access control policies or tokens, cryptographic systems, and so forth. However, an intruder could always penetrate a secure system because there is always a weak link in the security provisioning process. The second generation detected intrusions in a timely manner to exercise remedial actions. These techniques included firewalls, intrusion detection systems (IDSes), PKI services, reputation systems, and so on. The third generation provides more intelligent responses to intrusions.

1.5.3.5 Data Protection Infrastructure

Security infrastructure is required to safeguard web and cloud services. At the user level, one needs to perform trust negotiation and reputation aggregation over all users. At the application end, we need to establish security precautions in worm containment and intrusion detection

against virus, worm, and distributed DoS (DDoS) attacks. We also need to deploy mechanisms to prevent online piracy and copyright violations of digital content. In Chapter 4, we will study reputation systems for protecting cloud systems and data centers. Security responsibilities are divided between cloud providers and users differently for the three cloud service models. The providers are totally responsible for platform availability. The IaaS users are more responsible for the confidentiality issue. The IaaS providers are more responsible for data integrity. In PaaS and SaaS services, providers and users are equally responsible for preserving data integrity and confidentiality.

1.5.4 Energy Efficiency in Distributed Computing

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

Protection of data centers demands integrated solutions. Energy consumption in parallel and distributed computing systems raises various monetary, environmental, and system performance issues. For example, Earth Simulator and Petaflop are two systems with 12 and 100 megawatts of peak power, respectively. With an approximate price of $100 per megawatt, their energy costs during peak operation times are $1,200 and $10,000 per hour; this is beyond the acceptable budget of many (potential) system operators. In addition to power cost, cooling is another issue that must be addressed due to negative effects of high temperature on electronic components. The rising temperature of a circuit not only derails the circuit from its normal range, but also decreases the lifetime of its components.

1.5.4.1 Energy Consumption of Unused Servers

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

The potential savings in turning off these servers are large—$3.8 billion globally in energy costs alone, and $24.7 billion in the total cost of running nonproductive servers, according to a study by 1E Company in partnership with the Alliance to Save Energy (ASE). This amount of wasted energy is equal to 11.8 million tons of carbon dioxide per year, which is equivalent to the CO pollution of 2.1 million cars. In the United States, this equals 3.17 million tons of carbon dioxide, or 580,678 cars. Therefore, the first step in IT departments is to analyze their servers to find unused and/or underutilized servers.

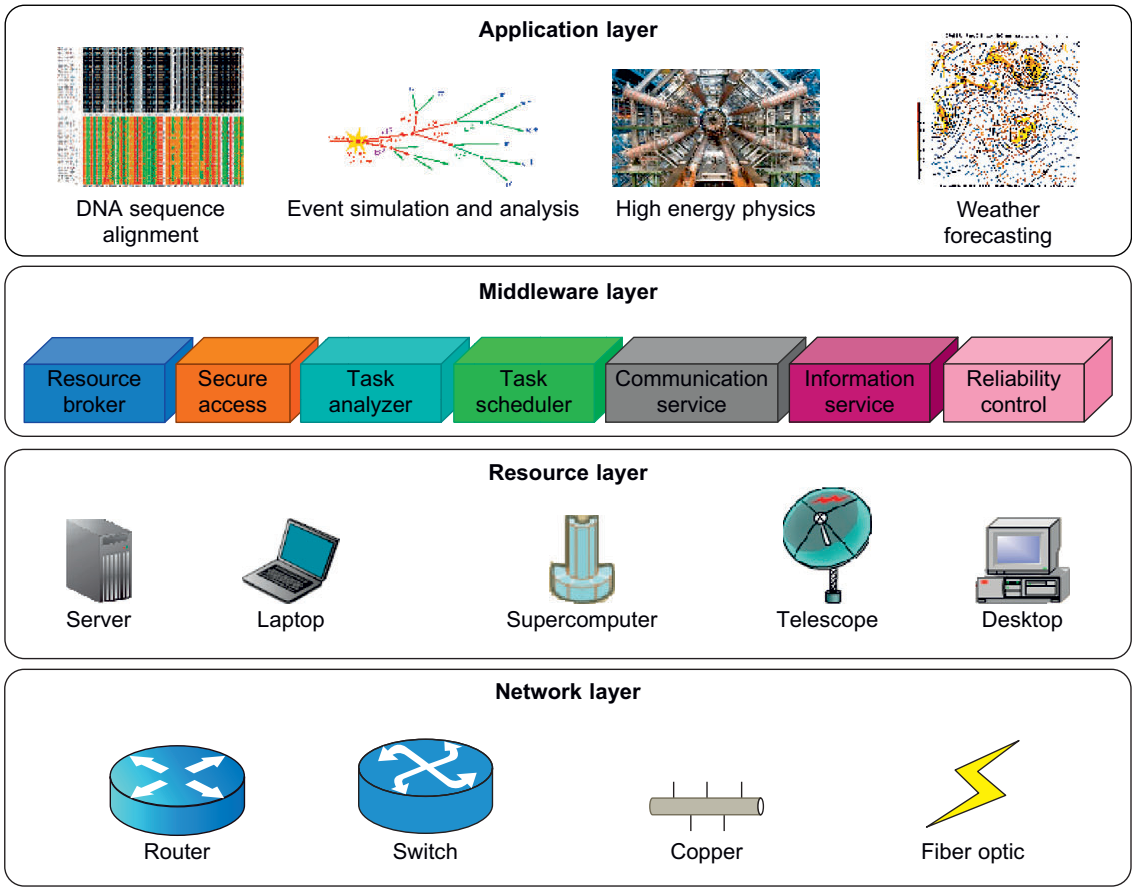

FIGURE 1.26

Four operational layers of distributed computing systems.

(*Courtesy of Zomaya, Rivandi and Lee of the University of Sydney [33]*)

1.5.4.2 Reducing Energy in Active Servers

In addition to identifying unused/underutilized servers for energy savings, it is also necessary to apply appropriate techniques to decrease energy consumption in active distributed systems with negligible influence on their performance. Power management issues in distributed computing platforms can be categorized into four layers (see Figure 1.26): the application layer, middleware layer, resource layer, and network layer.

1.5.4.3 Application Layer

Until now, most user applications in science, business, engineering, and financial areas tend to increase a system's speed or quality. By introducing energy-aware applications, the challenge is to design sophisticated multilevel and multi-domain energy management applications without hurting

performance. The first step toward this end is to explore a relationship between performance and energy consumption. Indeed, an application's energy consumption depends strongly on the number of instructions needed to execute the application and the number of transactions with the storage unit (or memory). These two factors (compute and storage) are correlated and they affect completion time.

1.5.4.4 Middleware Layer

The middleware layer acts as a bridge between the application layer and the resource layer. This layer provides resource broker, communication service, task analyzer, task scheduler, security access, reliability control, and information service capabilities. It is also responsible for applying energy-efficient techniques, particularly in task scheduling. Until recently, scheduling was aimed at minimizing *makespan*, that is, the execution time of a set of tasks. Distributed computing systems necessitate a new cost function covering both makespan and energy consumption.

1.5.4.5 Resource Layer

The resource layer consists of a wide range of resources including computing nodes and storage units. This layer generally interacts with hardware devices and the operating system; therefore, it is responsible for controlling all distributed resources in distributed computing systems. In the recent past, several mechanisms have been developed for more efficient power management of hardware and operating systems. The majority of them are hardware approaches particularly for processors.

Dynamic power management (DPM) and *dynamic voltage-frequency scaling (DVFS)* are two popular methods incorporated into recent computer hardware systems [21]. In DPM, hardware devices, such as the CPU, have the capability to switch from idle mode to one or more lower-power modes. In DVFS, energy savings are achieved based on the fact that the power consumption in CMOS circuits has a direct relationship with frequency and the square of the voltage supply. Execution time and power consumption are controllable by switching among different frequencies and voltages [31].

1.5.4.6 Network Layer

Routing and transferring packets and enabling network services to the resource layer are the main responsibility of the network layer in distributed computing systems. The major challenge to build energy-efficient networks is, again, determining how to measure, predict, and create a balance between energy consumption and performance. Two major challenges to designing energy-efficient networks are:

- The models should represent the networks comprehensively as they should give a full understanding of interactions among time, space, and energy.
- New, energy-efficient routing algorithms need to be developed. New, energy-efficient protocols should be developed against network attacks.

As information resources drive economic and social development, data centers become increasingly important in terms of where the information items are stored and processed, and where services are provided. Data centers become another core infrastructure, just like the power grid and

transportation systems. Traditional data centers suffer from high construction and operational costs, complex resource management, poor usability, low security and reliability, and huge energy consumption. It is necessary to adopt new technologies in next-generation data-center designs, a topic we will discuss in more detail in Chapter 4.

1.5.4.7 DVFS Method for Energy Efficiency

The DVFS method enables the exploitation of the slack time (idle time) typically incurred by inter-task relationship. Specifically, the slack time associated with a task is utilized to execute the task in a lower voltage frequency. The relationship between energy and voltage frequency in CMOS circuits is related by:

$$\begin{cases} E = C_{eff} f v^2 t \\ f = K \dfrac{(v - v_t)^2}{v} \end{cases} \tag{1.6}$$

where v, C_{eff}, K, and v_t are the voltage, circuit switching capacity, a technology dependent factor, and threshold voltage, respectively, and the parameter t is the execution time of the task under clock frequency f. By reducing voltage and frequency, the device's energy consumption can also be reduced.

Example 1.2 Energy Efficiency in Distributed Power Management

Figure 1.27 illustrates the DVFS method. This technique as shown on the right saves the energy compared to traditional practices shown on the left. The idea is to reduce the frequency and/or voltage during workload slack time. The transition latencies between lower-power modes are very small. Thus energy is saved by switching between operational modes. Switching between low-power modes affects performance. Storage units must interact with the computing nodes to balance power consumption. According to Ge, Feng, and Cameron [21], the storage devices are responsible for about 27 percent of the total energy consumption in a data center. This figure increases rapidly due to a 60 percent increase in storage needs annually, making the situation even worse.

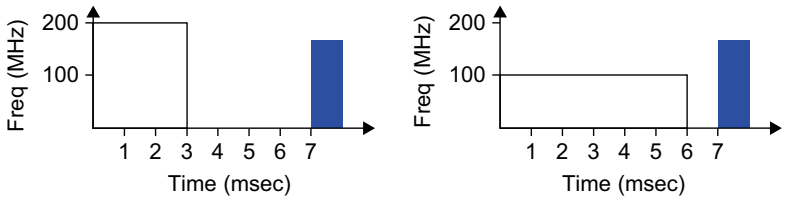

FIGURE 1.27

The DVFS technique (right) saves energy, compared to traditional practices (left) by reducing the frequency or voltage during slack time.

1.6 BIBLIOGRAPHIC NOTES AND HOMEWORK PROBLEMS

Over the past four decades, parallel processing and distributed computing have been hot topics for research and development. Earlier work in parallel computing can be found in several classic books [14,17,27,28]. Recent coverage of distributed systems can be found in [8,13,20,22]. Cluster computing was covered in [2,10,28,38] and grid computing in [6,18,42,47,51]. P2P networks were introduced in [6,34,46]. Multicore CPUs and many-core GPU processors were discussed in [15,32,36]. Information on the Top 500 supercomputers can be found in [50].

Data centers are introduced in [4,19,26], and recent computer architecture in [24,26]. Cloud computing is studied in [1,9,11,18,29,30,39,44]. The edited volume [11] on cloud computing by Buyya, Broberg, and Goscinski serves as a good resource on cloud computing research. Chou's book [12] emphasizes business models for cloud services. Virtualization techniques were introduced in [40–44]. The articles by Bell, Gray, and Szalay [5]; Foster, et al. [18]; and Hey [25] address critical issues concerning data-intensive grid and cloud computing. Massive data parallelism and programming are covered in [14,32].

Distributed algorithms and MPI programming are studied in [3,12,15,22,28]. Distributed operating systems and software tools are covered in [3,7,13,46]. Energy efficiency and power management are studied in [21,31,52]. The Internet of Things is studied in [45,48]. The work of Hwang and Li [30] suggested ways to cope with cloud security and data protection problems. In subsequent chapters, additional references will be provided. The following list highlights some international conferences, magazines, and journals that often report on the latest developments in parallelism, clusters, grids, P2P systems, and cloud and distributed systems:

- **IEEE and Related Conference Publications** Internet Computing, TPDS (Transactions on Parallel and Distributed Systems), TC (Transactions on Computers), TON (Transactions on Networking), ICDCS (International Conference on Distributed Computing Systems), IPDPS (International Parallel and Distributed Processing Symposium), INFOCOM, GLOBECOM, CCGrid (Clusters, Clouds and The Grid), P2P Computing, HPDC (High-Performance Distributed Computing), CloudCom (International Conference on Cloud Computing Technology and Science), ISCA (International Symposium on Computer Architecture), HPCA (High-Performance Computer Architecture), Computer Magazine, TDSC (Transactions on Dependable and Secure Computing), TKDE (Transactions on Knowledge and Data Engineering), HPCC (High Performance Computing and Communications), ICPADS (International Conference on Parallel and Distributed Applications and Systems), NAS (Networks, Architectures, and Storage), and GPC (Grid and Pervasive Computing)
- **ACM, Internet Society, IFIP, and Other Relevant Publications** Supercomputing Conference, ACM Transactions on Computing Systems, USENIX Technical Conference, JPDC (Journal of Parallel and Distributed Computing), Journal of Cloud Computing, Journal of Distributed Computing, Journal of Cluster Computing, Future Generation Computer Systems, Journal of Grid Computing, Journal of Parallel Computing, International Conference on Parallel Processing (ICPP), European Parallel Computing Conference (EuroPAR), Concurrency: Practice and Experiences (Wiley), NPC (IFIP Network and Parallel Computing), and PDCS (ISCA Parallel and Distributed Computer Systems)

Acknowledgments

This chapter was authored by Kai Hwang primarily. Geoffrey Fox and Albert Zomaya have contributed to Sections 1.4.1 and 1.5.4, respectively. Xiaosong Lou and Lizhong Chen at the University of Southern California have assisted in plotting Figures 1.4 and 1.10. Nikzad Babaii Rizvandi and Young-Choon Lee of the University of Sydney have contributed partially to Section 1.5.4. Jack Dongarra has edited the entire Chapter 1.

References

[1] Amazon EC2 and S3, Elastic Compute Cloud (EC2) and Simple Scalable Storage (S3). http://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud and http://spatten_presentations.s3.amazonaws.com/s3-on-rails.pdf.
[2] M. Baker, R. Buyya, Cluster computing at a glance, in: R. Buyya (Ed.), High-Performance Cluster Computing, Architecture and Systems, vol. 1, Prentice-Hall, Upper Saddle River, NJ, 1999, pp. 3–47, Chapter 1.
[3] A. Barak, A. Shiloh, The MOSIX Management System for Linux Clusters, Multi-Clusters, CPU Clusters, and Clouds, White paper. www.MOSIX.org//txt_pub.html, 2010.
[4] L. Barroso, U. Holzle, The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Morgan & Claypool Publishers, 2009.
[5] G. Bell, J. Gray, A. Szalay, Petascale computational systems: balanced cyberstructure in a data-centric World, IEEE Comput. Mag. (2006).
[6] F. Berman, G. Fox, T. Hey (Eds.), Grid Computing, Wiley, 2003.
[7] M. Bever, et al., Distributed systems, OSF DCE, and beyond, in: A. Schill (Ed.), DCE-The OSF Distributed Computing Environment, Springer-Verlag, 1993, pp. 1–20.
[8] K. Birman, Reliable Distributed Systems: Technologies, Web Services, and Applications, Springer-Verlag, 2005.
[9] G. Boss, et al., Cloud Computing–The BlueCloud Project. www.ibm.com/developerworks/websphere/zones/hipods/, October 2007.
[10] R. Buyya (Ed.), High-Performance Cluster Computing, Vol. 1 and 2, Prentice-Hall, Englewood Cliffs, NJ, 1999.
[11] R. Buyya, J. Broberg, A. Goscinski (Eds.), Cloud Computing: Principles and Paradigms, Wiley, 2011.
[12] T. Chou, Introduction to Cloud Computing: Business and Technology. Lecture Notes at Stanford University and Tsinghua University, Active Book Press, 2010.
[13] G. Coulouris, J. Dollimore, T. Kindberg, Distributed Systems: Concepts and Design, Wesley, 2005.
[14] D. Culler, J. Singh, A. Gupta, Parallel Computer Architecture, Kaufmann Publishers, 1999.
[15] B. Dally, GPU Computing to Exascale and Beyond, Keynote Address at ACM Supercomputing Conference, November 2010.
[16] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, in: Proceedings of OSDI 2004. Also, Communication of ACM, Vol. 51, 2008, pp. 107–113.
[17] J. Dongarra, et al. (Eds.), Source Book of Parallel Computing, Morgan Kaufmann, San Francisco, 2003.
[18] I. Foster, Y. Zhao, J. Raicu, S. Lu, Cloud Computing and Grid Computing 360-Degree Compared, Grid Computing Environments Workshop, 12–16 November 2008.
[19] D. Gannon, The Client+Cloud: Changing the Paradigm for Scientific Research, Keynote Address, IEEE CloudCom2010, Indianapolis, 2 November 2010.
[20] V.K. Garg, Elements of Distributed Computing. Wiley-IEEE Press, 2002.
[21] R. Ge, X. Feng, K. Cameron, Performance Constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters, in: Proceedings Supercomputing Conf., Washington, DC, 2005.

[22] S. Ghosh, Distributed Systems–An Algorithmic Approach, Chapman & Hall/CRC, 2007.
[23] B. He, W. Fang, Q. Luo, N. Govindaraju, T. Wang, Mars: A MapReduce Framework on Graphics Processor, ACM PACT'08, Toronto, Canada, 25–29 October 2008.
[24] J. Hennessy, D. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 2007.
[25] T. Hey, et al., The Fourth Paradigm: Data-Intensive Scientific Discovery, Microsoft Research, 2009.
[26] M.D. Hill, et al., The Data Center as a Computer, Morgan & Claypool Publishers, 2009.
[27] K. Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programming, McGraw-Hill, 1993.
[28] K. Hwang, Z. Xu, Scalable Parallel Computing, McGraw-Hill, 1998.
[29] K. Hwang, S. Kulkarni, Y. Hu, Cloud Security with Virtualized Defense and Reputation-based Trust Management, in: IEEE Conference on Dependable, Autonomous, and Secure Computing (DAC-2009), Chengdu, China, 14 December 2009.
[30] K. Hwang, D. Li, Trusted Cloud Computing with Secure Resources and Data Coloring, in: IEEE Internet Computing, Special Issue on Trust and Reputation Management, September 2010, pp. 14–22.
[31] Kelton Research, Server Energy & Efficiency Report. www.1e.com/EnergyCampaign/downloads/Server_Energy_and_Efficiency_Report_2009.pdf, September 2009.
[32] D. Kirk, W. Hwu, Programming Massively Processors: A Hands-on Approach, Morgan Kaufmann, 2010.
[33] Y.C. Lee, A.Y. Zomaya, A Novel State Transition Method for Metaheuristic-Based Scheduling in Heterogeneous Computing Systems, in: IEEE Transactions on Parallel and Distributed Systems, September 2008.
[34] Z.Y. Li, G. Xie, K. Hwang, Z.C. Li, Churn-Resilient Protocol for Massive Data Dissemination in P2P Networks, in: IEEE Trans. Parallel and Distributed Systems, May 2011.
[35] X. Lou, K. Hwang, Collusive Piracy Prevention in P2P Content Delivery Networks, in: IEEE Trans. on Computers, July, 2009, pp. 970–983.
[36] NVIDIA Corp. Fermi: NVIDIA's Next-Generation CUDA Compute Architecture, White paper, 2009.
[37] D. Peleg, Distributed Computing: A Locality-Sensitive Approach, SIAM, 2000.
[38] G.F. Pfister, In Search of Clusters, Second ed., Prentice-Hall, 2001.
[39] J. Qiu, T. Gunarathne, J. Ekanayake, J. Choi, S. Bae, H. Li, et al., Hybrid Cloud and Cluster Computing Paradigms for Life Science Applications, in: 11th Annual Bioinformatics Open Source Conference BOSC 2010, 9–10 July 2010.
[40] M. Rosenblum, T. Garfinkel, Virtual machine monitors: current technology and future trends, IEEE Computer (May) (2005) 39–47.
[41] M. Rosenblum, Recent Advances in Virtual Machines and Operating Systems, Keynote Address, ACM ASPLOS 2006.
[42] J. Smith, R. Nair, Virtual Machines, Morgan Kaufmann, 2005.
[43] B. Sotomayor, R. Montero, I. Foster, Virtual Infrastructure Management in Private and Hybrid Clouds, IEEE Internet Computing, September 2009.
[44] SRI. The Internet of Things, in: Disruptive Technologies: Global Trends 2025, www.dni.gov/nic/PDF_GIF_Confreports/disruptivetech/appendix_F.pdf, 2010.
[45] A. Tanenbaum, Distributed Operating Systems, Prentice-Hall, 1995.
[46] I. Taylor, From P2P to Web Services and Grids, Springer-Verlag, London, 2005.
[47] Twister, Open Source Software for Iterative MapReduce, http://www.iterativemapreduce.org/.
[48] Wikipedia. Internet of Things, http://en.wikipedia.org/wiki/Internet_of_Things, June 2010.
[49] Wikipedia. CUDA, http://en.wikipedia.org/wiki/CUDA, March 2011.
[50] Wikipedia. TOP500, http://en.wikipedia.org/wiki/TOP500, February 2011.
[51] Y. Wu, K. Hwang, Y. Yuan, W. Zheng, Adaptive Workload Prediction of Grid Performance in Confidence Windows, in: IEEE Trans. on Parallel and Distributed Systems, July 2010.
[52] Z. Zong, Energy-Efficient Resource Management for High-Performance Computing Platforms, Ph.D. dissertation, Auburn University, 9 August 2008.

HOMEWORK PROBLEMS

Problem 1.1

Briefly define the following basic techniques and technologies that represent recent related advances in computer architecture, parallel processing, distributed computing, Internet technology, and information services:

- **a.** High-performance computing (HPC) system
- **b.** High-throughput computing (HTC) system
- **c.** Peer-to-peer (P2P) network
- **d.** Computer cluster versus computational grid
- **e.** Service-oriented architecture (SOA)
- **f.** Pervasive computing versus Internet computing
- **g.** Virtual machine versus virtual infrastructure
- **h.** Public cloud versus private cloud
- **i.** Radio-frequency identifier (RFID)
- **j.** Global positioning system (GPS)
- **k.** Sensor network
- **l.** Internet of Things (IoT)
- **m.** Cyber-physical system (CPS)

Problem 1.2

Circle only one correct answer in each of the following two questions:

1. In the 2009 Top 500 list of the fastest computer systems, which architecture dominates?
   - **a.** Symmetric shared-memory multiprocessor systems.
   - **b.** Centralized massively parallel processor (MPP) systems.
   - **c.** Clusters of cooperative computers.
2. In a cloud formed by a cluster of servers, all servers must be selected as follows:
   - **a.** All cloud machines must be built on physical servers.
   - **b.** All cloud machines must be built with virtual servers.
   - **c.** The cloud machines can be either physical or virtual servers.

Problem 1.3

An increasing number of organizations in industry and business sectors adopt cloud systems. Answer the following questions regarding cloud computing:

- **a.** List and describe the main characteristics of cloud computing systems.
- **b.** Discuss key enabling technologies in cloud computing systems.
- **c.** Discuss different ways for cloud service providers to maximize their revenues.

Problem 1.4

Match 10 abbreviated terms and system models on the left with their descriptions on the right. Enter the description label (**a, b, c, …, j**) in the underlined blanks in front of the terms.

_____**Globus**	(**a**) A scalable software platform promoted by Apache for web users to write and run applications over vast amounts of distributed data
_____**BitTorrent**	(**b**) A P2P network for MP3 music delivery with a centralized directory server
_____**Gnutella**	(**c**) The programming model and associated implementation by Google for distributed mapping and reduction of very large data sets
_____**EC2**	(**d**) A middleware library jointly developed by USC/ISI and Argonne National Lab for grid resource management and job scheduling
_____**TeraGrid**	(**e**) A distributed storage program by Google for managing structured data that can scale to very large sizes
_____**EGEE**	(**f**) A P2P file-sharing network using multiple file index trackers
_____**Hadoop**	(**g**) A critical design goal of clusters of computers to tolerate nodal faults or recovery from host failures
_____**SETI@home**	(**h**) The service architecture specification as an open grid standard
_____**Napster**	(**i**) An elastic and flexible computing environment that allows web application developers to acquire cloud resources effectively
_____**BigTable**	(**j**) A P2P grid over 3 million desktops for distributed signal processing in search of extraterrestrial intelligence

Problem 1.5

Consider a multicore processor with four heterogeneous cores labeled A, B, C, and D. Assume cores A and D have the same speed. Core B runs twice as fast as core A, and core C runs three times faster than core A. Assume that all four cores start executing the following application at the same time and no cache misses are encountered in all core operations. Suppose an application needs to compute the square of each element of an array of 256 elements. Assume 1 unit time for core A or D to compute the square of an element. Thus, core B takes $\frac{1}{2}$ unit time and core C takes $\frac{1}{3}$ unit time to compute the square of an element. Given the following division of labor in four cores:

Core A	32 elements
Core B	128 elements
Core C	64 elements
Core D	32 elements

a. Compute the *total execution time* (in time units) for using the four-core processor to compute the squares of 256 elements in parallel. The four cores have different speeds. Some faster cores finish the job and may become idle, while others are still busy computing until all squares are computed.

b. Calculate the *processor utilization rate*, which is the total amount of time the cores are busy (not idle) divided by the total execution time they are using all cores in the processor to execute the above application.

Problem 1.6

Consider parallel execution of an MPI-coded C program in SPMD (single program and multiple data streams) mode on a server cluster consisting of n identical Linux servers. SPMD mode means

the same MPI program is running simultaneously on all servers but over different data sets of identical workloads. Assume that 25 percent of the program execution is attributed to the execution of MPI commands. For simplicity, assume that all MPI commands take the same amount of execution time. Answer the following questions using Amdahl's law:

a. Given that the total execution time of the MPI program on a four-server cluster is T minutes, what is the speedup factor of executing the same MPI program on a 256-server cluster, compared with using the four-server cluster? Assume that the program execution is deadlock-free and ignore all other runtime execution overheads in the calculation.

b. Suppose that all MPI commands are now enhanced by a factor of 2 by using active messages executed by message handlers at the user space. The enhancement can reduce the execution time of all MPI commands by half. What is the speedup of the 256-server cluster installed with this MPI enhancement, computed with the old 256-server cluster without MPI enhancement?

Problem 1.7

Consider a program for multiplying two large-scale $N \times N$ matrices, where N is the matrix size. The sequential multiply time on a single server is $T_1 = cN^3$ minutes, where c is a constant determined by the server used. An MPI-code parallel program requires $T_n = cN^3/n + dN^2/n^{0.5}$ minutes to complete execution on an n-server cluster system, where d is a constant determined by the MPI version used. Assume the program has a zero sequential bottleneck ($\alpha = 0$). The second term in T_n accounts for the total message-passing overhead experienced by n servers.

Answer the following questions for a given cluster configuration with $n = 64$ servers, $c = 0.8$, and $d = 0.1$. Parts (a, b) have a fixed workload corresponding to the matrix size $N = 15{,}000$. Parts (c, d) have a scaled workload associated with an enlarged matrix size $N' = n^{1/3} N = 64^{1/3} \times 15{,}000 = 4 \times 15{,}000 = 60{,}000$. Assume the same cluster configuration to process both workloads. Thus, the system parameters n, c, and d stay unchanged. Running the scaled workload, the overhead also increases with the enlarged matrix size N'.

a. Using Amdahl's law, calculate the speedup of the n-server cluster over a single server.

b. What is the efficiency of the cluster system used in Part (a)?

c. Calculate the speedup in executing the scaled workload for an enlarged $N' \times N'$ matrix on the same cluster configuration using Gustafson's law.

d. Calculate the efficiency of running the scaled workload in Part (c) on the 64-processor cluster.

e. Compare the above speedup and efficiency results and comment on their implications.

Problem 1.8

Compare the similarities and differences between traditional computing clusters/grids and the computing clouds launched in recent years. Consider all technical and economic aspects as listed below. Answer the following questions against real example systems or platforms built in recent years. Also discuss the possible convergence of the two computing paradigms in the future.

a. Hardware, software, and networking support

b. Resource allocation and provisioning methods

c. Infrastructure management and protection
d. Support of utility computing services
e. Operational and cost models applied

Problem 1.9

Answer the following questions regarding PC and HPC systems:

a. Explain why PCs and HPCs were evolutionary rather than revolutionary in the past 30 years.
b. Discuss the drawbacks in disruptive changes in processor architecture. Why is the memory wall a major problem in achieving scalable changes in performance?
c. Explain why x-86 processors are still dominating the PC and HPC markets.

Problem 1.10

Multicore and many-core processors have appeared in widespread use in both desktop computers and HPC systems. Answer the following questions regarding advanced processors, memory devices, and system interconnects:

a. What are the differences between multicore CPUs and GPUs in terms of architecture and usage?
b. Explain why parallel programming cannot match the progress of processor technology.
c. Suggest ideas and defend your argument with some plausible solutions to this mismatch problem between core scaling and effective programming and use of multicores.
d. Explain why flash memory SSD can deliver better speedups in some HPC or HTC applications.
e. Justify the prediction that InfiniBand and Ethernet will continue to dominate the HPC market.

Problem 1.11

In Figure 1.7, you studied five categories of modern processors. Characterize in Table 1.9 five micro-architectures for designing these processors. Comment on their advantages/shortcomings and identify the names of two example commercial processors that are built in each processor category. Assume a single core in the superscalar processor and the three multithreaded processors. The last processor category is a multicore CMP and each core is assumed to handle one thread at a time.

Table 1.9 Comparison of Five Micro-architectures for Modern Processors

Processor Micro-architectures	Architecture Charcteristics	Advantages/ Shortcomings	Representative Processors
Single-threaded Superscalar			
Fine-grain Multithreading			
Coarse-grain Multithreading			
Simultaneous Multithreading (SMT)			
Multicore Chip Multiprocessor (CMP)			

Problem 1.12

Discuss the major advantages and disadvantages in the following areas:

a. Why are virtual machines and virtual clusters suggested in cloud computing systems?
b. What breakthroughs are required to build virtualized cloud systems cost-effectively?
c. What are the impacts of cloud platforms on the future of the HPC and HTC industry?

Problem 1.13

Characterize the following three cloud computing models:

a. What is an IaaS (Infrastructure-as-a-Service) cloud? Give one example system.
b. What is a PaaS (Platform-as-a-Service) cloud? Give one example system.
c. What is a SaaS (Software-as-a-Service) cloud? Give one example system.

Problem 1.14

Briefly explain each of the following cloud computing services. Identify two cloud providers by company name in each service category.

a. Application cloud services
b. Platform cloud services
c. Compute and storage services
d. Collocation cloud services
e. Network cloud services

Problem 1.15

Briefly explain the following terms associated with network threats or security defense in a distributed computing system:

a. Denial of service (DoS)
b. Trojan horse
c. Network worm
d. Service spoofing
e. Authorization
f. Authentication
g. Data integrity
h. Confidentiality

Problem 1.16

Briefly answer the following questions regarding green information technology and energy efficiency in distributed systems:

a. Why is power consumption critical to data-center operations?
b. What constitutes the dynamic voltage frequency scaling (DVFS) technique?
c. Conduct in-depth research on recent progress in green IT research, and write a report on its applications to data-center design and cloud service applications.

Problem 1.17

Compare GPU and CPU chips in terms of their strengths and weaknesses. In particular, discuss the trade-offs between power efficiency, programmability, and performance. Also compare various MPP architectures in processor selection, performance target, efficiency, and packaging constraints.

Problem 1.18

Compare three distributed operating systems: Amoeba, DCE, and MOSIX. Research their recent developments and their impact on applications in clusters, grids, and clouds. Discuss the suitability of each system in its commercial or experimental distributed applications. Also discuss each system's limitations and explain why they were not successful as commercial systems.

This page intentionally left blank

CHAPTER

Computer Clusters for Scalable Parallel Computing

2

CHAPTER OUTLINE

SUMMARY

Clustering of computers enables scalable parallel and distributed computing in both science and business applications. This chapter is devoted to building cluster-structured massively parallel processors. We focus on the design principles and assessment of the hardware, software, middleware, and operating system support to achieve scalability, availability, programmability, single-system images, and fault tolerance in clusters. We will examine the cluster architectures of Tianhe-1A, Cray XT5 Jaguar, and IBM Roadrunner. The study also covers the LSF middleware and MOSIX/OS for job and resource management in Linux clusters, GPU clusters and cluster extensions to building grids and clouds. Only physical clusters are studied in this chapter. Virtual clusters will be studied in Chapters 3 and 4.

2.1 CLUSTERING FOR MASSIVE PARALLELISM

A *computer cluster* is a collection of interconnected stand-alone computers which can work together collectively and cooperatively as a single integrated computing resource pool. Clustering explores massive parallelism at the job level and achieves *high availability* (HA) through stand-alone operations. The benefits of computer clusters and *massively parallel processors* (MPPs) include scalable performance, HA, fault tolerance, modular growth, and use of commodity components. These features can sustain the generation changes experienced in hardware, software, and network components. Cluster computing became popular in the mid-1990s as traditional mainframes and vector supercomputers were proven to be less cost-effective in many *high-performance computing* (HPC) applications.

Of the Top 500 supercomputers reported in 2010, 85 percent were computer clusters or MPPs built with homogeneous nodes. Computer clusters have laid the foundation for today's supercomputers, computational grids, and Internet clouds built over data centers. We have come a long way toward becoming addicted to computers. According to a recent IDC prediction, the HPC market will increase from $8.5 billion in 2010 to $10.5 billion by 2013. A majority of the Top 500 supercomputers are used for HPC applications in science and engineering. Meanwhile, the use of *high-throughput computing* (HTC) clusters of servers is growing rapidly in business and web services applications.

2.1.1 Cluster Development Trends

Support for clustering of computers has moved from interconnecting high-end mainframe computers to building clusters with massive numbers of x86 engines. Computer clustering started with the linking of large mainframe computers such as the IBM Sysplex and the SGI Origin 3000. Originally, this was motivated by a demand for cooperative group computing and to provide higher availability in critical enterprise applications. Subsequently, the clustering trend moved toward the networking of many minicomputers, such as DEC's VMS cluster, in which multiple VAXes were interconnected to share the same set of disk/tape controllers. Tandem's Himalaya was designed as a business cluster for fault-tolerant *online transaction processing* (OLTP) applications.

In the early 1990s, the next move was to build UNIX-based workstation clusters represented by the Berkeley NOW (Network of Workstations) and IBM SP2 AIX-based server cluster. Beyond

2000, we see the trend moving to the clustering of RISC or x86 PC engines. Clustered products now appear as integrated systems, software tools, availability infrastructure, and operating system extensions. This clustering trend matches the downsizing trend in the computer industry. Supporting clusters of smaller nodes will increase sales by allowing modular incremental growth in cluster configurations. From IBM, DEC, Sun, and SGI to Compaq and Dell, the computer industry has leveraged clustering of low-cost servers or x86 desktops for their cost-effectiveness, scalability, and HA features.

2.1.1.1 Milestone Cluster Systems

Clustering has been a hot research challenge in computer architecture. Fast communication, job scheduling, SSI, and HA are active areas in cluster research. Table 2.1 lists some milestone cluster research projects and commercial cluster products. Details of these old clusters can be found in [14]. These milestone projects have pioneered clustering hardware and middleware development over the past two decades. Each cluster project listed has developed some unique features. Modern clusters are headed toward HPC clusters as studied in Section 2.5.

The NOW project addresses a whole spectrum of cluster computing issues, including architecture, software support for web servers, single system image, I/O and file system, efficient communication, and enhanced availability. The Rice University TreadMarks is a good example of software-implemented shared-memory cluster of workstations. The memory sharing is implemented with a user-space runtime library. This was a research cluster built over Sun Solaris workstations. Some cluster OS functions were developed, but were never marketed successfully.

Table 2.1 Milestone Research or Commercial Cluster Computer Systems [14]

Project	Special Features That Support Clustering
DEC VAXcluster (1991)	A UNIX cluster of symmetric multiprocessing (SMP) servers running the VMS OS with extensions, mainly used in HA applications
U.C. Berkeley NOW Project (1995)	A serverless network of workstations featuring active messaging, cooperative filing, and GLUnix development
Rice University TreadMarks (1996)	Software-implemented distributed shared memory for use in clusters of UNIX workstations based on page migration
Sun Solaris MC Cluster (1995)	A research cluster built over Sun Solaris workstations; some cluster OS functions were developed but were never marketed successfully
Tandem Himalaya Cluster (1994)	A scalable and fault-tolerant cluster for OLTP and database processing, built with nonstop operating system support
IBM SP2 Server Cluster (1996)	An AIX server cluster built with Power2 nodes and the Omega network, and supported by IBM LoadLeveler and MPI extensions
Google Search Engine Cluster (2003)	A 4,000-node server cluster built for Internet search and web service applications, supported by a distributed file system and fault tolerance
MOSIX (2010) www.mosix.org	A distributed operating system for use in Linux clusters, multiclusters, grids, and clouds; used by the research community

A Unix cluster of SMP servers running VMS/OS with extensions, mainly used in high-availability applications. An AIX server cluster built with Power2 nodes and Omega network and supported by IBM Loadleveler and MPI extensions. A scalable and fault-tolerant cluster for OLTP and database processing built with non-stop operating system support. The Google search engine was built at Google using commodity components. MOSIX is a distributed operating systems for use in Linux clusters, multi-clusters, grids, and the clouds, originally developed by Hebrew University in 1999.

2.1.2 Design Objectives of Computer Clusters

Clusters have been classified in various ways in the literature. We classify clusters using six orthogonal attributes: *scalability*, *packaging*, *control*, *homogeneity*, *programmability*, and *security*.

2.1.2.1 Scalability

Clustering of computers is based on the concept of modular growth. To scale a cluster from hundreds of uniprocessor nodes to a supercluster with 10,000 multicore nodes is a nontrivial task. The scalability could be limited by a number of factors, such as the multicore chip technology, cluster topology, packaging method, power consumption, and cooling scheme applied. The purpose is to achieve scalable performance constrained by the aforementioned factors. We have to also consider other limiting factors such as the memory wall, disk I/O bottlenecks, and latency tolerance, among others.

2.1.2.2 Packaging

Cluster nodes can be packaged in a compact or a slack fashion. In a *compact* cluster, the nodes are closely packaged in one or more racks sitting in a room, and the nodes are not attached to peripherals (monitors, keyboards, mice, etc.). In a *slack* cluster, the nodes are attached to their usual peripherals (i.e., they are complete SMPs, workstations, and PCs), and they may be located in different rooms, different buildings, or even remote regions. Packaging directly affects communication wire length, and thus the selection of interconnection technology used. While a compact cluster can utilize a high-bandwidth, low-latency communication network that is often proprietary, nodes of a slack cluster are normally connected through standard LANs or WANs.

2.1.2.3 Control

A cluster can be either controlled or managed in a *centralized* or *decentralized* fashion. A compact cluster normally has centralized control, while a slack cluster can be controlled either way. In a centralized cluster, all the nodes are owned, controlled, managed, and administered by a central operator. In a decentralized cluster, the nodes have individual owners. For instance, consider a cluster comprising an interconnected set of desktop workstations in a department, where each workstation is individually owned by an employee. The owner can reconfigure, upgrade, or even shut down the workstation at any time. This lack of a single point of control makes system administration of such a cluster very difficult. It also calls for special techniques for process scheduling, workload migration, checkpointing, accounting, and other similar tasks.

2.1.2.4 Homogeneity

A *homogeneous* cluster uses nodes from the same platform, that is, the same processor architecture and the same operating system; often, the nodes are from the same vendors. A *heterogeneous*

cluster uses nodes of different platforms. Interoperability is an important issue in heterogeneous clusters. For instance, process migration is often needed for load balancing or availability. In a homogeneous cluster, a binary process image can migrate to another node and continue execution. This is not feasible in a heterogeneous cluster, as the binary code will not be executable when the process migrates to a node of a different platform.

2.1.2.5 Security

Intracluster communication can be either *exposed* or *enclosed*. In an exposed cluster, the communication paths among the nodes are exposed to the outside world. An outside machine can access the communication paths, and thus individual nodes, using standard protocols (e.g., TCP/IP). Such exposed clusters are easy to implement, but have several disadvantages:

- Being exposed, intracluster communication is not secure, unless the communication subsystem performs additional work to ensure privacy and security.
- Outside communications may disrupt intracluster communications in an unpredictable fashion. For instance, heavy BBS traffic may disrupt production jobs.
- Standard communication protocols tend to have high overhead.

In an enclosed cluster, intracluster communication is shielded from the outside world, which alleviates the aforementioned problems. A disadvantage is that there is currently no standard for efficient, enclosed intracluster communication. Consequently, most commercial or academic clusters realize fast communications through one-of-a-kind protocols.

2.1.2.6 Dedicated versus Enterprise Clusters

A *dedicated cluster* is typically installed in a deskside rack in a central computer room. It is homogeneously configured with the same type of computer nodes and managed by a single administrator group like a frontend host. Dedicated clusters are used as substitutes for traditional mainframes or supercomputers. A dedicated cluster is installed, used, and administered as a single machine. Many users can log in to the cluster to execute both interactive and batch jobs. The cluster offers much enhanced throughput, as well as reduced response time.

An *enterprise cluster* is mainly used to utilize idle resources in the nodes. Each node is usually a full-fledged SMP, workstation, or PC, with all the necessary peripherals attached. The nodes are typically geographically distributed, and are not necessarily in the same room or even in the same building. The nodes are individually owned by multiple owners. The cluster administrator has only limited control over the nodes, as a node can be turned off at any time by its owner. The owner's "local" jobs have higher priority than enterprise jobs. The cluster is often configured with heterogeneous computer nodes. The nodes are often connected through a low-cost Ethernet network. Most data centers are structured with clusters of low-cost servers. Virtual clusters play a crucial role in upgrading data centers. We will discuss virtual clusters in Chapter 6 and clouds in Chapters 7, 8, and 9.

2.1.3 Fundamental Cluster Design Issues

In this section, we will classify various cluster and MPP families. Then we will identify the major design issues of clustered and MPP systems. Both physical and virtual clusters are covered. These systems are often found in computational grids, national laboratories, business data centers,

supercomputer sites, and virtualized cloud platforms. A good understanding of how clusters and MPPs work collectively will pave the way toward understanding the ins and outs of large-scale grids and Internet clouds in subsequent chapters. Several issues must be considered in developing and using a cluster. Although much work has been done in this regard, this is still an active research and development area.

2.1.3.1 Scalable Performance

This refers to the fact that scaling of resources (cluster nodes, memory capacity, I/O bandwidth, etc.) leads to a proportional increase in performance. Of course, both scale-up and scale-down capabilities are needed, depending on application demand or cost-effectiveness considerations. Clustering is driven by scalability. One should not ignore this factor in all applications of cluster or MPP computing systems.

2.1.3.2 Single-System Image (SSI)

A set of workstations connected by an Ethernet network is not necessarily a cluster. A cluster is a single system. For example, suppose a workstation has a 300 Mflops/second processor, 512 MB of memory, and a 4 GB disk and can support 50 active users and 1,000 processes. By clustering 100 such workstations, can we get a single system that is equivalent to one huge workstation, or a *megastation*, that has a 30 Gflops/second processor, 50 GB of memory, and a 400 GB disk and can support 5,000 active users and 100,000 processes? This is an appealing goal, but it is very difficult to achieve. SSI techniques are aimed at achieving this goal.

2.1.3.3 Availability Support

Clusters can provide cost-effective HA capability with lots of redundancy in processors, memory, disks, I/O devices, networks, and operating system images. However, to realize this potential, availability techniques are required. We will illustrate these techniques later in the book, when we discuss how DEC clusters (Section 10.4) and the IBM SP2 (Section 10.3) attempt to achieve HA.

2.1.3.4 Cluster Job Management

Clusters try to achieve high system utilization from traditional workstations or PC nodes that are normally not highly utilized. Job management software is required to provide batching, load balancing, parallel processing, and other functionality. We will study cluster job management systems in Section 3.4. Special software tools are needed to manage multiple jobs simultaneously.

2.1.3.5 Internode Communication

Because of their higher node complexity, cluster nodes cannot be packaged as compactly as MPP nodes. The internode physical wire lengths are longer in a cluster than in an MPP. This is true even for centralized clusters. A long wire implies greater interconnect network latency. But more importantly, longer wires have more problems in terms of reliability, clock skew, and cross talking. These problems call for reliable and secure communication protocols, which increase overhead. Clusters often use commodity networks (e.g., Ethernet) with standard protocols such as TCP/IP.

2.1.3.6 Fault Tolerance and Recovery

Clusters of machines can be designed to eliminate all single points of failure. Through redundancy, a cluster can tolerate faulty conditions up to a certain extent. Heartbeat mechanisms can be installed

to monitor the running condition of all nodes. In case of a node failure, critical jobs running on the failing nodes can be saved by failing over to the surviving node machines. Rollback recovery schemes restore the computing results through periodic checkpointing.

2.1.3.7 Cluster Family Classification

Based on application demand, computer clusters are divided into three classes:

- **Compute clusters** These are clusters designed mainly for collective computation over a single large job. A good example is a cluster dedicated to numerical simulation of weather conditions. The compute clusters do not handle many I/O operations, such as database services. When a single compute job requires frequent communication among the cluster nodes, the cluster must share a dedicated network, and thus the nodes are mostly homogeneous and tightly coupled. This type of clusters is also known as a ***Beowulf cluster***.
  When the nodes require internode communication over a small number of heavy-duty nodes, they are essentially known as a *computational grid*. Tightly coupled compute clusters are designed for supercomputing applications. Compute clusters apply middleware such as a message-passing interface (**MPI**) or Parallel Virtual Machine (**PVM**) to port programs to a wide variety of clusters.
- **High-Availability clusters** HA (high-availability) clusters are designed to be fault-tolerant and achieve HA of services. HA clusters operate with many redundant nodes to sustain faults or failures. The simplest HA cluster has only two nodes that can fail over to each other. Of course, high redundancy provides higher availability. HA clusters should be designed to avoid all single points of failure. Many commercial HA clusters are available for various operating systems.
- **Load-balancing clusters** These clusters shoot for higher resource utilization through load balancing among all participating nodes in the cluster. All nodes share the workload or function as a single *virtual machine* (VM). Requests initiated from the user are distributed to all node computers to form a cluster. This results in a balanced workload among different machines, and thus higher resource utilization or higher performance. Middleware is needed to achieve dynamic load balancing by job or process migration among all the cluster nodes.

2.1.4 Analysis of the Top 500 Supercomputers

Every six months, the world's Top 500 supercomputers are evaluated by running the Linpack Benchmark program over very large data sets. The ranking varies from year to year, similar to a competition. In this section, we will analyze the historical share in architecture, speed, operating systems, countries, and applications over time. In addition, we will compare the top five fastest systems in 2010.

2.1.4.1 Architectural Evolution

It is interesting to observe in Figure 2.1 the architectural evolution of the Top 500 supercomputers over the years. In 1993, 250 systems assumed the SMP architecture and these SMP systems all disappeared in June of 2002. Most SMPs are built with shared memory and shared I/O devices. There were 120 MPP systems built in 1993, MPPs reached the peak mof 350 systems in mod-200, and dropped to less than 100 systems in 2010. The single instruction, multiple data (SIMD) machines disappeared in 1997. The cluster architecture appeared in a few systems in 1999. The cluster systems are now populated in the Top-500 list with more than 400 systems as the dominating architecture class.

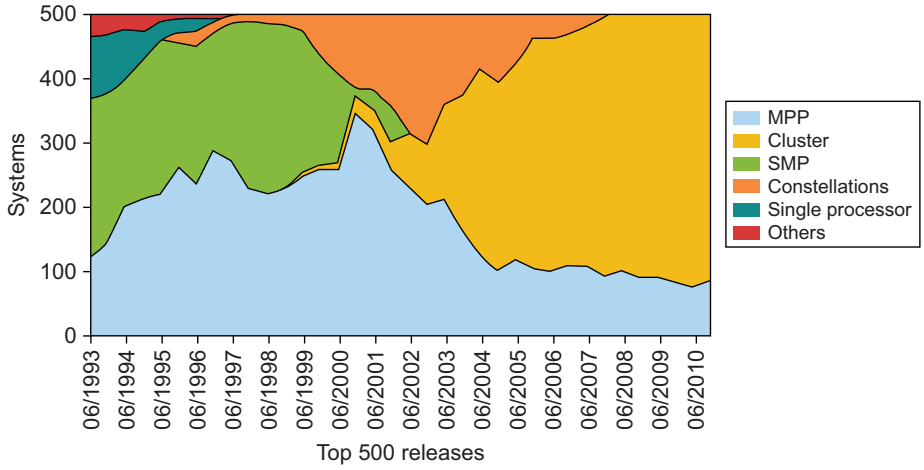

FIGURE 2.1

Architectural share of the Top 500 systems.

(*Courtesy of www.top500.org [25]*)

In 2010, the Top 500 architecture is dominated by clusters (420 systems) and MPPs (80 systems). The basic distinction between these two classes lies in the components they use to build the systems. Clusters are often built with commodity hardware, software, and network components that are commercially available. MPPs are built with custom-designed compute nodes, boards, modules, and cabinets that are interconnected by special packaging. MPPs demand high bandwidth, low latency, better power efficiency, and high reliability. Cost-wise, clusters are affordable by allowing modular growth with scaling capability. The fact that MPPs appear in a much smaller quantity is due to their high cost. Typically, only a few MPP-based supercomputers are installed in each country.

2.1.4.2 Speed Improvement over Time

Figure 2.2 plots the measured performance of the Top 500 fastest computers from 1993 to 2010. The y-axis is scaled by the sustained speed performance in terms of Gflops, Tflops, and Pflops. The middle curve plots the performance of the fastest computers recorded over 17 years; peak performance increases from 58.7 Gflops to 2.566 Pflops. The bottom curve corresponds to the speed of the 500th computer, which increased from 0.42 Gflops in 1993 to 31.1 Tflops in 2010. The top curve plots the speed sum of all 500 computers over the same period. In 2010, the total speed sum of 43.7 Pflops was achieved by all 500 computers, collectively. It is interesting to observe that the total speed sum increases almost linearly with time.

2.1.4.3 Operating System Trends in the Top 500

The five most popular operating systems have more than a 10 percent share among the Top 500 computers, according to data released by TOP500.org (www.top500.org/stats/list/36/os) in November 2010. According to the data, 410 supercomputers are using Linux with a total processor count exceeding 4.5 million. This constitutes 82 percent of the systems adopting Linux. The IBM AIX/OS is in second place with 17 systems (a 3.4 percent share) and more than 94,288 processors.

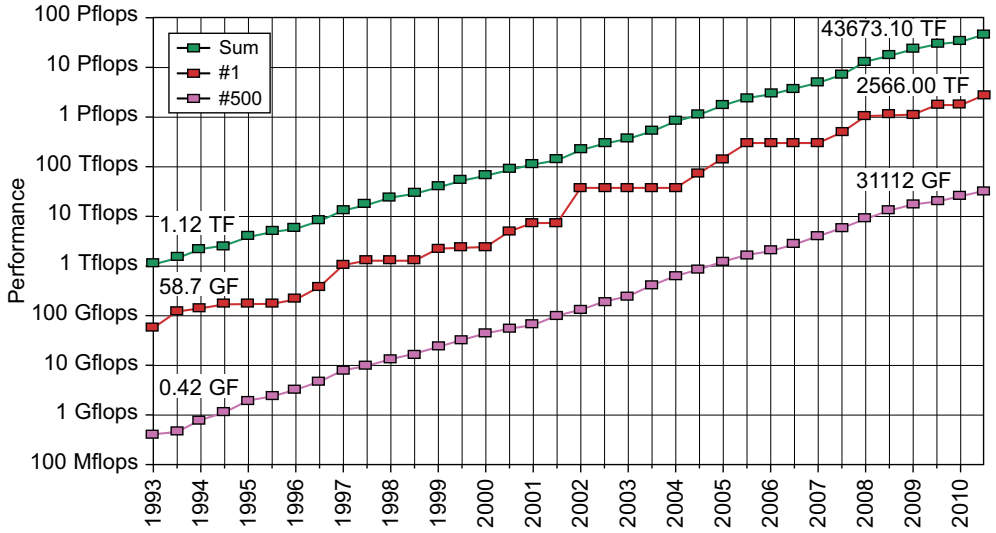

FIGURE 2.2

Performance plot of the Top 500 supercomputers from 1993 to 2010.

(*Courtesy of www.top500.org [25]*)

Third place is represented by the combined use of the SLEs10 with the SGI ProPack5, with 15 systems (3 percent) over 135,200 processors. Fourth place goes to the CNK/SLES9 used by 14 systems (2.8 percent) over 1.13 million processors. Finally, the CNL/OS was used in 10 systems (2 percent) over 178,577 processors. The remaining 34 systems applied 13 other operating systems with a total share of only 6.8 percent. In conclusion, the Linux OS dominates the systems in the Top 500 list.

2.1.4.4 The Top Five Systems in 2010

In Table 2.2, we summarize the key architecture features and sustained Linpack Benchmark performance of the top five supercomputers reported in November 2010. The Tianhe-1A was ranked as the fastest MPP in late 2010. This system was built with 86,386 Intel Xeon CPUs and NVIDIA GPUs by the National University of Defense Technology in China. We will present in Section 2.5 some of the top winners: namely the Tianhe-1A, Cray Jaguar, Nebulae, and IBM Roadrunner that were ranked among the top systems from 2008 to 2010. All the top five machines in Table 2.3 have achieved a speed higher than 1 Pflops. The *sustained speed*, $\boldsymbol{R_{max}}$, in Pflops is measured from the execution of the Linpack Benchmark program corresponding to a maximum matrix size.

The *system efficiency* reflects the ratio of the *sustained speed* to the *peak speed*, $\boldsymbol{R_{peak}}$, when all computing elements are fully utilized in the system. Among the top five systems, the two U.S.-built systems, the Jaguar and the Hopper, have the highest efficiency, more than 75 percent. The two systems built in China, the Tianhe-1A and the Nebulae, and Japan's TSUBAME 2.0, are all low in efficiency. In other words, these systems should still have room for improvement in the future. The average power consumption in these 5 systems is 3.22 MW. This implies that excessive power consumption may post the limit to build even faster supercomputers in the future. These top systems all emphasize massive parallelism using up to 250,000 processor cores per system.

Table 2.2 The Top Five Supercomputers Evaluated in November 2010

System Name, Site, and URL	System Name, Processors, OS, Topology, and Developer	Linpack Speed (R_{max}), Power	Efficiency (R_{max}/R_{peak})
1. Tianhe-1A, National Supercomputing Center, Tianjin, China, http://www.nscc-tj.gov.cn/en/	NUDT TH1A with 14,336 Xeon X5670 CPUs (six cores each) plus 7168 NVIDIA Tesla M2050 GPUs (448 CUDA cores each), running Linux, built by National Univ. of Defense Technology, China	2.57 Pflops, 4.02 MW	54.6% (over a peak of 4.7 Pflops)
2. Jaguar, DOE/SC/Oak Ridge National Lab., United States, http://computing.ornl.gov	Cray XT5-HE: MPP with 224,162 x 6 AMD Opteron, 3D torus network, Linux (CLE), manufactured by Cray, Inc.	1.76 Pflops, 6.95 MW	75.6% (over a peak of 4.7 Pflops)
3. Nebulae at China's National Supercomputer Center, ShenZhen, China http://www.ict.cas.cas.cn	TC3600 Blade, 120,640 cores in 55,680 Xeon X5650 plus 64,960 NVIDIA Tesla C2050 GPUs, Linux, InfiniBand, built by Dawning, Inc.	1.27 Pflops, 2.55 MW	42.6% (over a peak of 2.98 Pflops)
4. TSUBAME 2.0, GSIC Center, Tokyo Institute of Technology, Tokyo, Japan, http://www.gsic.titech.ac.jp/	HP cluster, 3000SL, 73,278 x 6 Xeon X5670 processors, NVIDIA GPU, Linux/SLES 11, built by NEC/HP	1.19 Pflops, 1.8 MW	52.19% (over a peak of 2.3 Pflops)
5. Hopper, DOE/SC/LBNL/ NERSC, Berkeley, CA. USA, http//www/nersc.gov/	Cray XE6 150,408 x 12 AMD Opteron, Linux (CLE), built by Cray, Inc.	1.05 Pflops, 2.8 MW	78.47% (over a peak of 1.35 Pflops)

Table 2.3 Sample Compute Node Architectures for Large Cluster Construction

Node Architecture	Major Characteristics	Representative Systems
Homogeneous node using the same multicore processors	Multicore processors mounted on the same node with a crossbar connected to shared memory or local disks	The Cray XT5 uses two six-core AMD Opteron processors in each compute node
Hybrid nodes using CPU plus GPU or FLP accelerators	General-purpose CPU for integer operations, with GPUs acting as coprocessors to speed up FLP operations	China's Tianhe-1A uses two Intel Xeon processors plus one NVIDIA GPU per compute node

2.1.4.5 Country Share and Application Share

In the 2010 Top-500 list, there were 274 supercomputing systems installed in the US, 41 systems in China, and 103 systems in Japan, France, UK, and Germany. The remaining countries have 82 systems. This country share roughly reflects the countries' economic growth over the years. Countries compete in the Top 500 race every six months. Major increases of supercomputer applications are in the areas of database, research, finance, and information services.

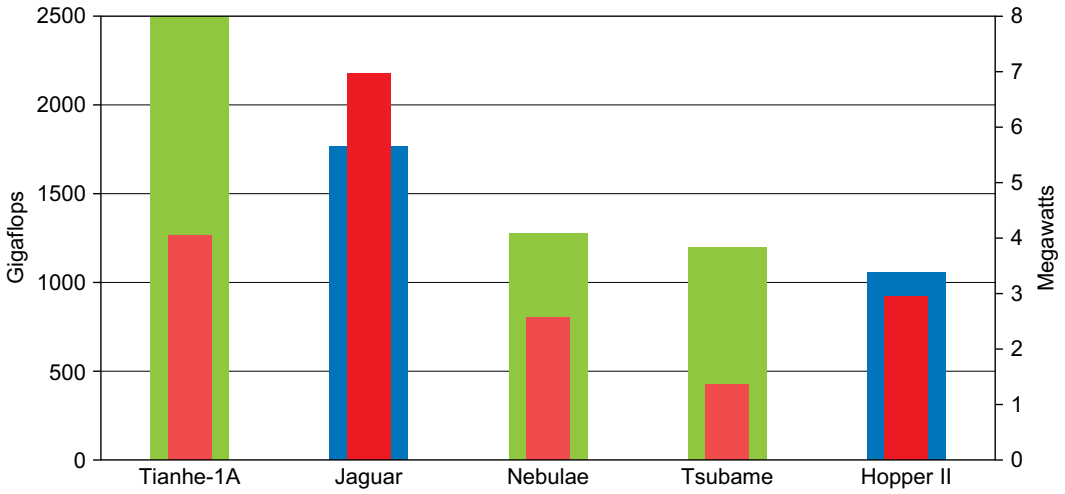

FIGURE 2.3

Power and performance of the top 5 supercomputers in November 2010.

(*Courtesy of www.top500.org [25] and B. Dally [10]*)

2.1.4.6 Power versus Performance in the Top Five in 2010

In Figure 2.3, the top five supercomputers are ranked by their speed (Gflops) on the left side and by their power consumption (MW per system) on the right side. The Tiahhe-1A scored the highest with a 2.57 Pflops speed and 4.01 MW power consumption. In second place, the Jaguar consumes the highest power of 6.9 MW. In fourth place, the TSUBAME system consumes the least power, 1.5 MW, and has a speed performance that almost matches that of the Nebulae system. One can define a performance/power ratio to see the trade-off between these two metrics. There is also a Top 500 Green contest that ranks the supercomputers by their power efficiency. This chart shows that all systems using the hybrid CPU/GPU architecture consume much less power.

2.2 COMPUTER CLUSTERS AND MPP ARCHITECTURES

Most clusters emphasize higher availability and scalable performance. Clustered systems evolved from the Microsoft Wolfpack and Berkeley NOW to the SGI Altix Series, IBM SP Series, and IBM Roadrunner. NOW was a UC Berkeley research project designed to explore new mechanisms for clustering of UNIX workstations. Most clusters use commodity networks such as Gigabit Ethernet, Myrinet switches, or InfiniBand networks to interconnect the compute and storage nodes. The clustering trend moves from supporting large rack-size, high-end computer systems to high-volume, desktop or deskside computer systems, matching the downsizing trend in the computer industry.

2.2.1 Cluster Organization and Resource Sharing

In this section, we will start by discussing basic, small-scale PC or server clusters. We will discuss how to construct large-scale clusters and MPPs in subsequent sections.

2.2.1.1 A Basic Cluster Architecture

Figure 2.4 shows the basic architecture of a computer cluster over PCs or workstations. The figure shows a simple cluster of computers built with commodity components and fully supported with desired SSI features and HA capability. The processing nodes are commodity workstations, PCs, or servers. These commodity nodes are easy to replace or upgrade with new generations of hardware. The node operating systems should be designed for multiuser, multitasking, and multithreaded applications. The nodes are interconnected by one or more fast commodity networks. These networks use standard communication protocols and operate at a speed that should be two orders of magnitude faster than that of the current TCP/IP speed over Ethernet.

The network interface card is connected to the node's standard I/O bus (e.g., PCI). When the processor or the operating system is changed, only the driver software needs to change. We desire to have a platform-independent *cluster operating system*, sitting on top of the node platforms. But such a cluster OS is not commercially available. Instead, we can deploy some cluster middleware to glue together all node platforms at the user space. An availability middleware offers HA services. An SSI layer provides a single entry point, a single file hierarchy, a single point of control, and a

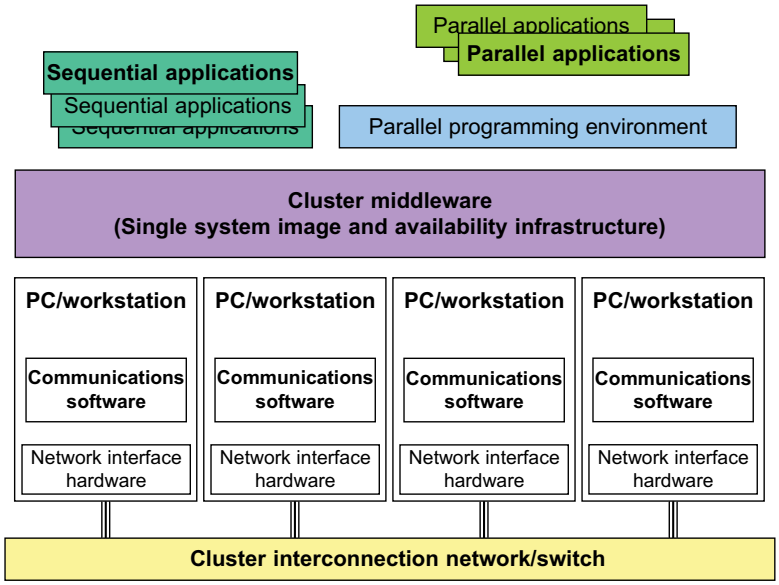

FIGURE 2.4

The architecture of a computer cluster built with commodity hardware, software, middleware, and network components supporting HA and SSI.

(*Courtesy of M. Baker and R. Buyya, reprinted with Permission [3]*)

single job management system. Single memory may be realized with the help of the compiler or a runtime library. A single process space is not necessarily supported.

In general, an idealized cluster is supported by three subsystems. First, conventional databases and OLTP monitors offer users a desktop environment in which to use the cluster. In addition to running sequential user programs, the cluster supports parallel programming based on standard languages and communication libraries using PVM, MPI, or OpenMP. The programming environment also includes tools for debugging, profiling, monitoring, and so forth. A user interface subsystem is needed to combine the advantages of the web interface and the Windows GUI. It should also provide user-friendly links to various programming environments, job management tools, hypertext, and search support so that users can easily get help in programming the computer cluster.

2.2.1.2 Resource Sharing in Clusters

Supporting clusters of smaller nodes will increase computer sales. Clustering improves both availability and performance. These two clustering goals are not necessarily in conflict. Some HA clusters use hardware redundancy for scalable performance. The nodes of a cluster can be connected in one of three ways, as shown in Figure 2.5. The shared-nothing architecture is used in most clusters, where the nodes are connected through the I/O bus. The shared-disk architecture is in favor of small-scale *availability clusters* in business applications. When one node fails, the other node takes over.

The shared-nothing configuration in Part (a) simply connects two or more autonomous computers via a LAN such as Ethernet. A shared-disk cluster is shown in Part (b). This is what most business clusters desire so that they can enable recovery support in case of node failure. The shared disk can hold checkpoint files or critical system images to enhance cluster availability. Without shared disks, checkpointing, rollback recovery, failover, and failback are not possible in a cluster. The shared-memory cluster in Part (c) is much more difficult to realize. The nodes could be connected by a *scalable coherence interface (SCI)* ring, which is connected to the memory bus of each node through an NIC module. In the other two architectures, the interconnect is attached to the I/O bus. The memory bus operates at a higher frequency than the I/O bus.

There is no widely accepted standard for the memory bus. But there are such standards for the I/O buses. One recent, popular standard is the PCI I/O bus standard. So, if you implement an NIC card to attach a faster Ethernet network to the PCI bus you can be assured that this card can be used in other systems that use PCI as the I/O bus. The I/O bus evolves at a much slower rate than the memory bus. Consider a cluster that uses connections through the PCI bus. When the processors are upgraded, the interconnect and the NIC do not have to change, as long as the new system still uses PCI. In a shared-memory cluster, changing the processor implies a redesign of the node board and the NIC card.

2.2.2 Node Architectures and MPP Packaging

In building large-scale clusters or MPP systems, cluster nodes are classified into two categories: *compute nodes* and *service nodes*. Compute nodes appear in larger quantities mainly used for large-scale searching or parallel floating-point computations. Service nodes could be built with different processors mainly used to handle I/O, file access, and system monitoring. For MPP clusters, the

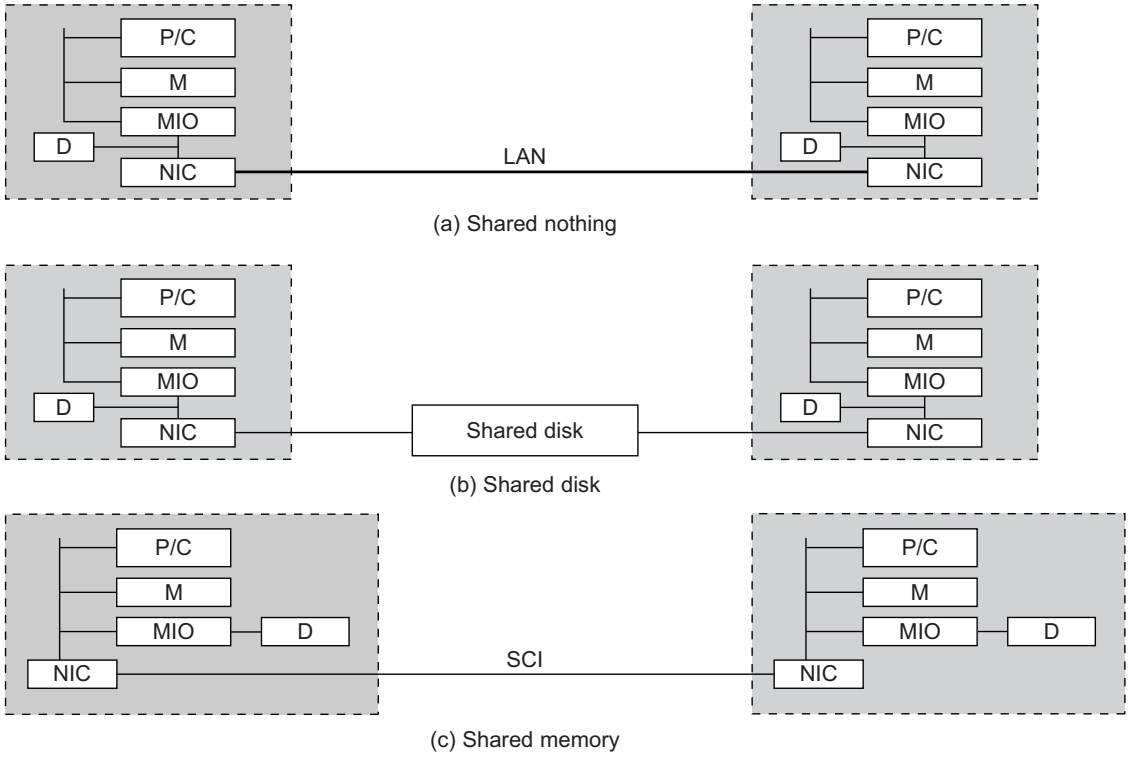

FIGURE 2.5

Three ways to connect cluster nodes (P/C: Processor and Cache; M: Memory; D: Disk; NIC: Network Interface Circuitry; MIO: Memory-I/O Bridge.)

(*Courtesy of Hwang and Xu [14]*)

compute nodes dominate in system cost, because we may have 1,000 times more compute nodes than service nodes in a single large clustered system. Table 2.3 introduces two example compute node architectures: *homogeneous design* and *hybrid node design.*

In the past, most MPPs are built with a homogeneous architecture by interconnecting a large number of the same compute nodes. In 2010, the Cray XT5 Jaguar system was built with 224,162 AMD Opteron processors with six cores each. The Tiahe-1A adopted a hybrid node design using two Xeon CPUs plus two AMD GPUs per each compute node. The GPU could be replaced by special floating-point accelerators. A homogeneous node design makes it easier to program and maintain the system.

Example 2.1 Modular Packaging of the IBM Blue Gene/L System

The Blue Gene/L is a supercomputer jointly developed by IBM and Lawrence Livermore National Laboratory. The system became operational in 2005 with a 136 Tflops performance at the No. 1 position in the

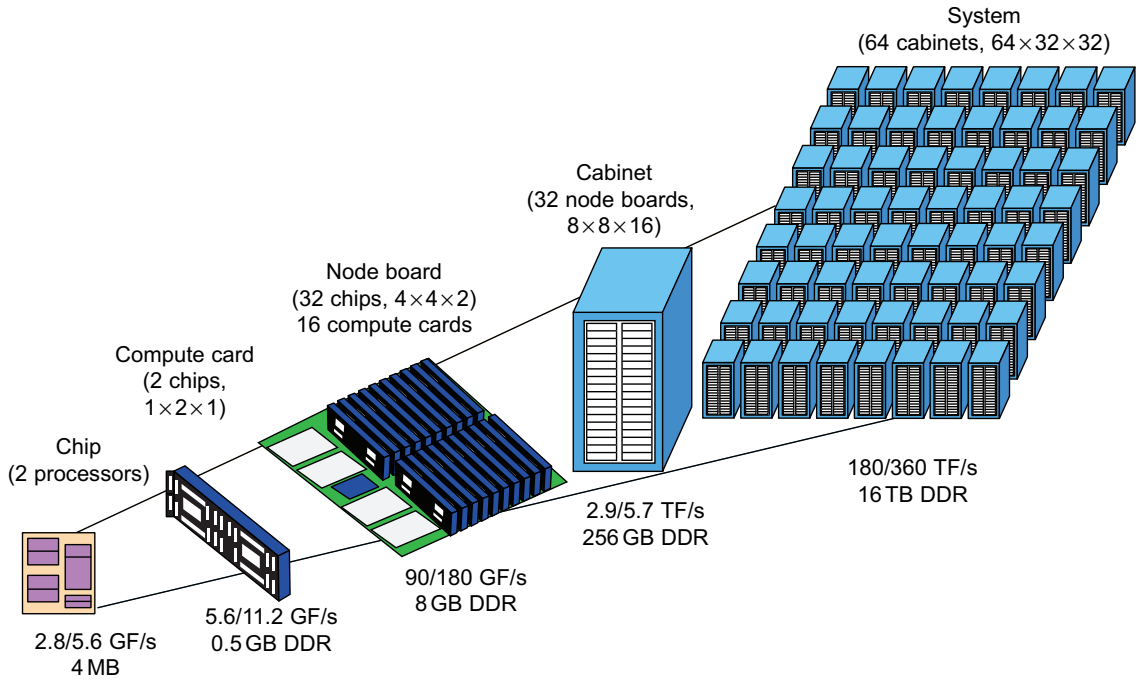

FIGURE 2.6

The IBM Blue Gene/L architecture built with modular components packaged hierarchically in five levels.

(*Courtesy of N. Adiga, et al., IBM Corp., 2005 [1]*)

Top-500 list—toped the Japanese Earth Simulator. The system was upgraded to score a 478 Tflops speed in 2007. By examining the architecture of the Blue Gene series, we reveal the modular construction of a scalable MPP system as shown in Figure 2.6. With modular packaging, the Blue Gene/L system is constructed hierarchically from processor chips to 64 physical racks. This system was built with a total of 65,536 nodes with two PowerPC 449 FP2 processors per node. The 64 racks are interconnected by a huge 3D 64 x 32 x 32 torus network.

In the lower-left corner, we see a dual-processor chip. Two chips are mounted on a computer card. Sixteen computer cards (32 chips or 64 processors) are mounted on a node board. A cabinet houses 32 node boards with an 8 x 8 x 16 torus interconnect. Finally, 64 cabinets (racks) form the total system at the upper-right corner. This packaging diagram corresponds to the 2005 configuration. Customers can order any size to meet their computational needs. The Blue Gene cluster was designed to achieve scalable performance, reliability through built-in testability, resilience by preserving locality of failures and checking mechanisms, and serviceability through partitioning and isolation of fault locations.

2.2.3 Cluster System Interconnects

2.2.3.1 High-Bandwidth Interconnects

Table 2.4 compares four families of high-bandwidth system interconnects. In 2007, Ethernet used a 1 Gbps link, while the fastest InfiniBand links ran at 30 Gbps. The Myrinet and Quadrics perform in between. The MPI latency represents the state of the art in long-distance message passing. All four technologies can implement any network topology, including crossbar switches, fat trees, and torus networks. The InfiniBand is the most expensive choice with the fastest link speed. The Ethernet is still the most cost-effective choice. We consider two example cluster interconnects over 1,024 nodes in Figure 2.7 and Figure 2.9. The popularity of five cluster interconnects is compared in Figure 2.8.

Table 2.4 Comparison of Four Cluster Interconnect Technologies Reported in 2007

Feature	Myrinet	Quadrics	InfiniBand	Ethernet
Available link speeds	1.28 Gbps (*M-XP*) 10 Gbps (*M-10G*)	2.8 Gbps (*QsNet*) 7.2 Gbps (*QsNetII*)	2.5 Gbps (*1X*) 10 Gbps (*4X*) 30 Gbps (*12X*)	1 Gbps
MPI latency	~3 us	~3 us	~4.5 us	~40 us
Network processor	Yes	Yes	Yes	No
Topologies	Any	Any	Any	Any
Network topology	Clos	Fat tree	Fat tree	Any
Routing	Source-based, cut-through	Source-based, cut-through	Destination-based	Destination-based
Flow control	Stop and go	Worm-hole	Credit-based	802.3x

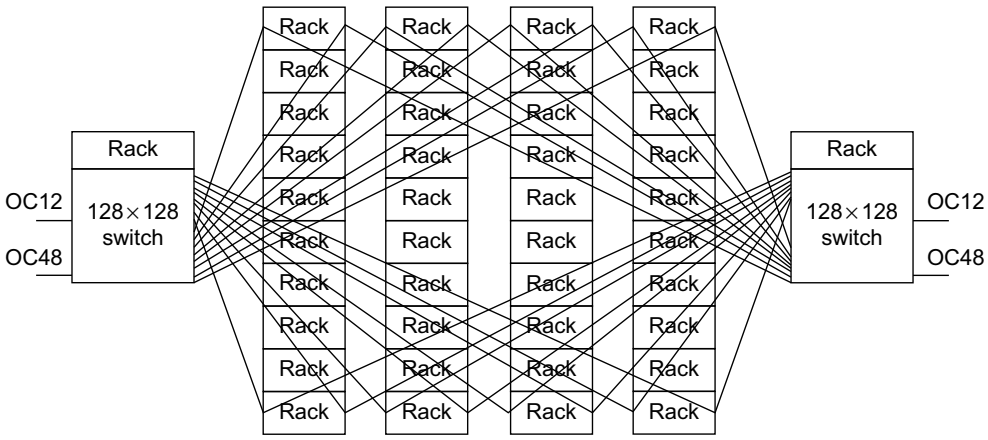

FIGURE 2.7

Google search engine cluster architecture.

(*Courtesy of Google, Inc. [6]*)

Example 2.2 Crossbar Switch in Google Search Engine Cluster

Google has many data centers using clusters of low-cost PC engines. These clusters are mainly used to support Google's web search business. Figure 2.7 shows a Google cluster interconnect of 40 racks of PC engines via two racks of 128 x 128 Ethernet switches. Each Ethernet switch can handle 128 one Gbps Ethernet links. A rack contains 80 PCs. This is an earlier cluster of 3,200 PCs. Google's search engine clusters are built with a lot more nodes. Today's server clusters from Google are installed in data centers with container trucks.

Two switches are used to enhance cluster availability. The cluster works fine even when one switch fails to provide the links among the PCs. The front ends of the switches are connected to the Internet via 2.4 Gbps OC 12 links. The 622 Mbps OC 12 links are connected to nearby data-center networks. In case of failure of the OC 48 links, the cluster is still connected to the outside world via the OC 12 links. Thus, the Google cluster eliminates all single points of failure.

2.2.3.2 Share of System Interconnects over Time

Figure 2.8 shows the distribution of large-scale system interconnects in the Top 500 systems from 2003 to 2008. Gigabit Ethernet is the most popular interconnect due to its low cost and market readiness. The InfiniBand network has been chosen in about 150 systems for its high-bandwidth performance. The Cray interconnect is designed for use in Cray systems only. The use of Myrinet and Quadrics networks had declined rapidly in the Top 500 list by 2008.

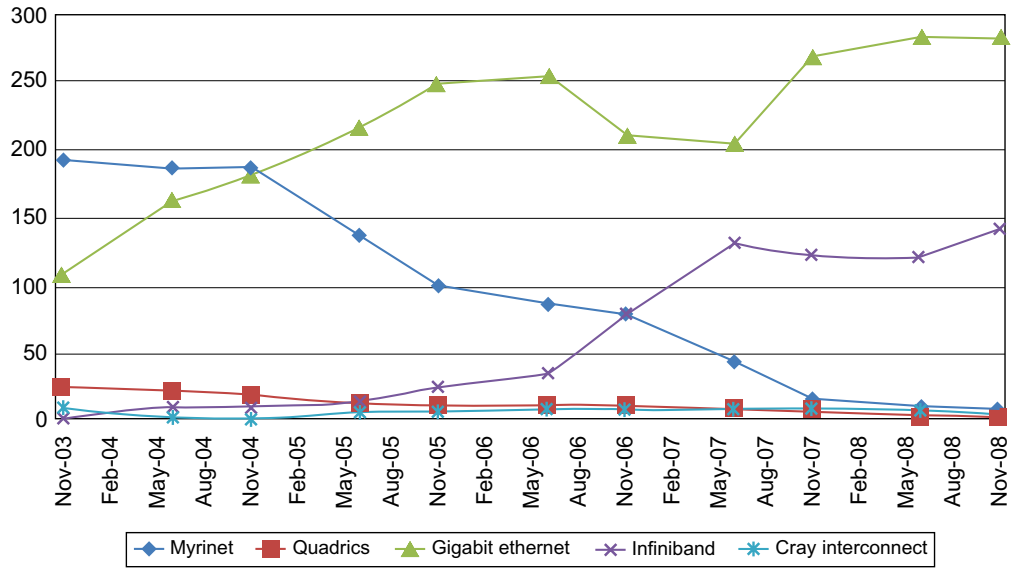

FIGURE 2.8

Distribution of high-bandwidth interconnects in the Top 500 systems from 2003 to 2008.

(*Courtesy of www.top500.org [25]*)

Example 2.3 Understanding the InfiniBand Architecture [8]

The InfiniBand has a switch-based point-to-point interconnect architecture. A large InfiniBand has a layered architecture. The interconnect supports the virtual interface architecture (VIA) for distributed messaging. The InfiniBand switches and links can make up any topology. Popular ones include crossbars, fat trees, and torus networks. Figure 2.9 shows the layered construction of an InfiniBand network. According to Table 2.5, the InfiniBand provides the highest speed links and the highest bandwidth in reported large-scale systems. However, InfiniBand networks cost the most among the four interconnect technologies.

Each end point can be a storage controller, a network interface card (NIC), or an interface to a host system. A host channel adapter (HCA) connected to the host processor through a standard peripheral component interconnect (PCI), PCI extended (PCI-X), or PCI express bus provides the host interface. Each HCA has more than one InfiniBand port. A target channel adapter (TCA) enables I/O devices to be loaded within the network. The TCA includes an I/O controller that is specific to its particular device's protocol such as SCSI, Fibre Channel, or Ethernet. This architecture can be easily implemented to build very large scale cluster interconnects that connect thousands or more hosts together. Supporting the InfiniBand in cluster applications can be found in [8].

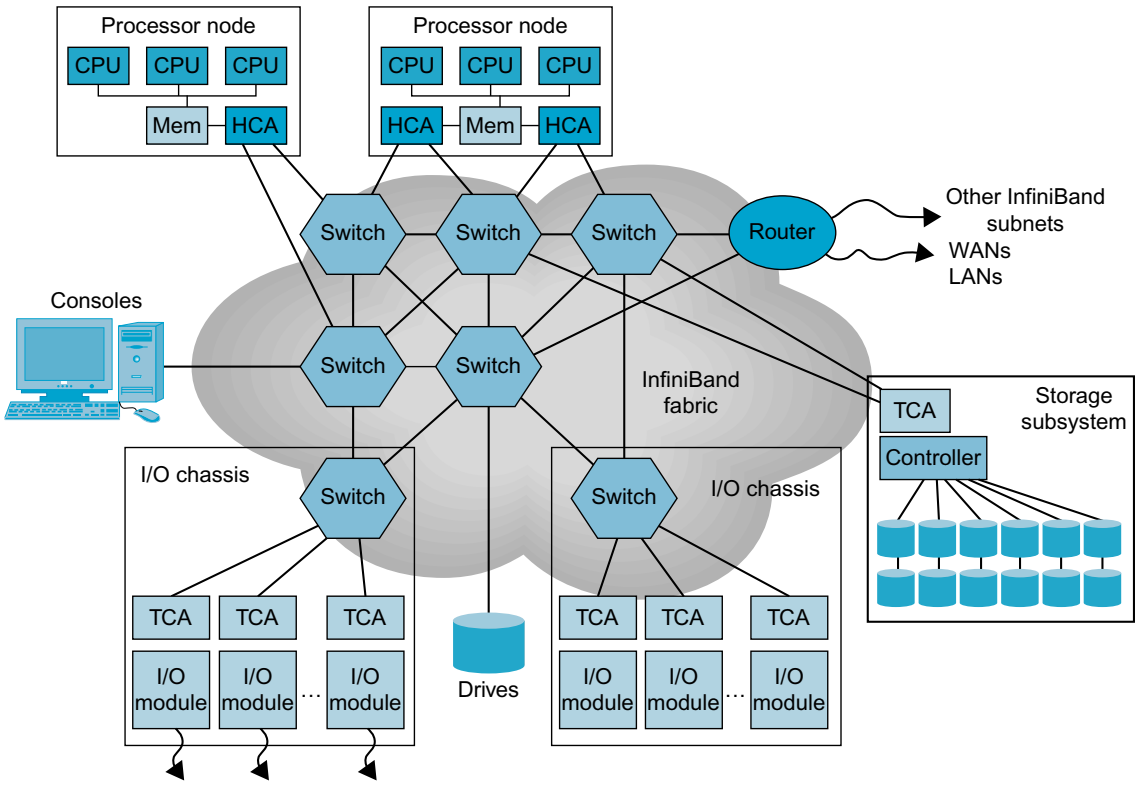

FIGURE 2.9

The InfiniBand system fabric built in a typical high-performance computer cluster.

(*Source: O. Celebioglu, et al, "Exploring InfiniBand as an HPC Cluster Interconnect", Dell Power Solutions, Oct.2004 © 2011 Dell Inc. All Rights Reserved*)

2.2.4 Hardware, Software, and Middleware Support

Realistically, SSI and HA features in a cluster are not obtained free of charge. They must be supported by hardware, software, middleware, or OS extensions. Any change in hardware design and OS extensions must be done by the manufacturer. The hardware and OS support could be cost-prohibitive to ordinary users. However, programming level is a big burden to cluster users. Therefore, the middleware support at the application level costs the least to implement. As an example, we show in Figure 2.10 the middleware, OS extensions, and hardware support needed to achieve HA in a typical Linux cluster system.

Close to the user application end, middleware packages are needed at the cluster management level: one for fault management to support *failover* and *failback*, to be discussed in Section 2.3.3. Another desired feature is to achieve HA using failure detection and recovery and packet switching. In the middle of Figure 2.10, we need to modify the Linux OS to support HA, and we need special drivers to support HA, I/O, and hardware devices. Toward the bottom, we need special hardware to support hot-swapped devices and provide router interfaces. We will discuss various supporting mechanisms in subsequent sections.

2.2.5 GPU Clusters for Massive Parallelism

Commodity GPUs are becoming high-performance accelerators for data-parallel computing. Modern GPU chips contain hundreds of processor cores per chip. Based on a 2010 report [19], each GPU

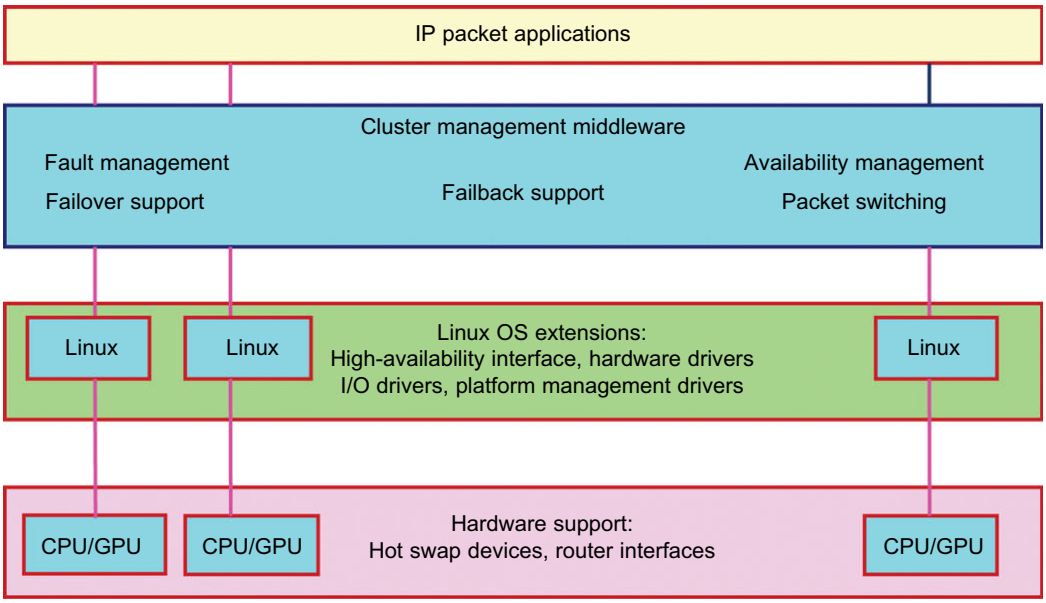

FIGURE 2.10

Middleware, Linux extensions, and hardware support for achieving massive parallelism and HA in a Linux cluster system built with CPUs and GPUs.

chip is capable of achieving up to 1 Tflops for single-precision (SP) arithmetic, and more than 80 Gflops for double-precision (DP) calculations. Recent HPC-optimized GPUs contain up to 4 GB of on-board memory, and are capable of sustaining memory bandwidths exceeding 100 GB/second. GPU clusters are built with a large number of GPU chips. GPU clusters have already demonstrated their capability to achieve Pflops performance in some of the Top 500 systems. Most GPU clusters are structured with homogeneous GPUs of the same hardware class, make, and model. The software used in a GPU cluster includes the OS, GPU drivers, and clustering API such as an MPI.

The high performance of a GPU cluster is attributed mainly to its massively parallel multicore architecture, high throughput in multithreaded floating-point arithmetic, and significantly reduced time in massive data movement using large on-chip cache memory. In other words, GPU clusters already are more cost-effective than traditional CPU clusters. GPU clusters result in not only a quantum jump in speed performance, but also significantly reduced space, power, and cooling demands. A GPU cluster can operate with a reduced number of operating system images, compared with CPU-based clusters. These reductions in power, environment, and management complexity make GPU clusters very attractive for use in future HPC applications.

2.2.5.1 The Echelon GPU Chip Design

Figure 2.11 shows the architecture of a future GPU accelerator that was suggested for use in building a NVIDIA Echelon GPU cluster for Exascale computing. This Echelong project led by Bill Dally at NVIDIA is partially funded by DARPA under the Ubiquitous High-Performance Computing (UHPC) program. This GPU design incorporates 1024 stream cores and 8 latency-optimized CPU-like cores (called latency processor) on a single chip. Eight stream cores form a *stream multiprocessor* (SM) and there are 128 SMs in the Echelon GPU chip.

Each SM is designed with 8 processor cores to yield a 160 Gflops peak speed. With 128 SMs, the chip has a peak speed of 20.48 Tflops. These nodes are interconnected by a NoC (network on chip) to 1,024 SRAM banks (L2 caches). Each cache band has a 256 KB capacity. The MCs (*memory controllers*) are used to connect to off-chip DRAMs and the NI (*network interface*) is to scale the size of the GPU cluster hierarchically, as shown in Figure 2.14. At the time of this writing, the Echelon is only a research project. With permission from Bill Dally, we present the design for

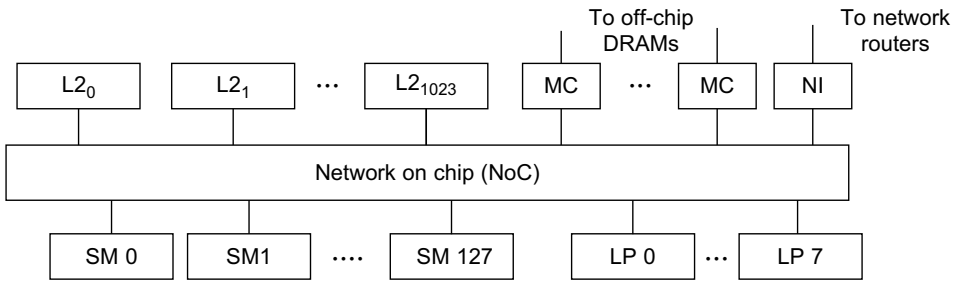

FIGURE 2.11

The proposed GPU chip design for 20 Tflops performance and 1.6 TB/s memory bandwidth in the Echelon system.
(*Courtesy of Bill Dally, Reprinted with Permission [10]*)

academic interest to illustrate how one can explore the many-core GPU technology to achieve Exascale computing in the future PU technology to achieve Exascale computing in the future.

2.2.5.2 GPU Cluster Components

A GPU cluster is often built as a heterogeneous system consisting of three major components: the CPU host nodes, the GPU nodes and the cluster interconnect between them. The GPU nodes are formed with general-purpose GPUs, known as GPGPUs, to carry out numerical calculations. The host node controls program execution. The cluster interconnect handles inter-node communications. To guarantee the performance, multiple GPUs must be fully supplied with data streams over high-bandwidth network and memory. Host memory should be optimized to match with the on-chip cache bandwidths on the GPUs. Figure 2.12 shows the proposed Echelon GPU clusters using the GPU chips shown in Figure 2.13 as building blocks interconnected by a hierarchically constructed network.

2.2.5.3 Echelon GPU Cluster Architecture

The Echelon system architecture is shown in Figure 2.11, hierarchically. The entire Echelon system is built with *N* cabinets, labeled C0, C1, ... , C*N*. Each cabinet is built with 16 compute module labeled as M0, M1, ... , M15. Each compute module is built with 8 GPU nodes labeled as N0, N1, ... , N7. Each GPU node is the innermost block labeled as PC in Figure 2.12 (also detailed in Figure 2.11).

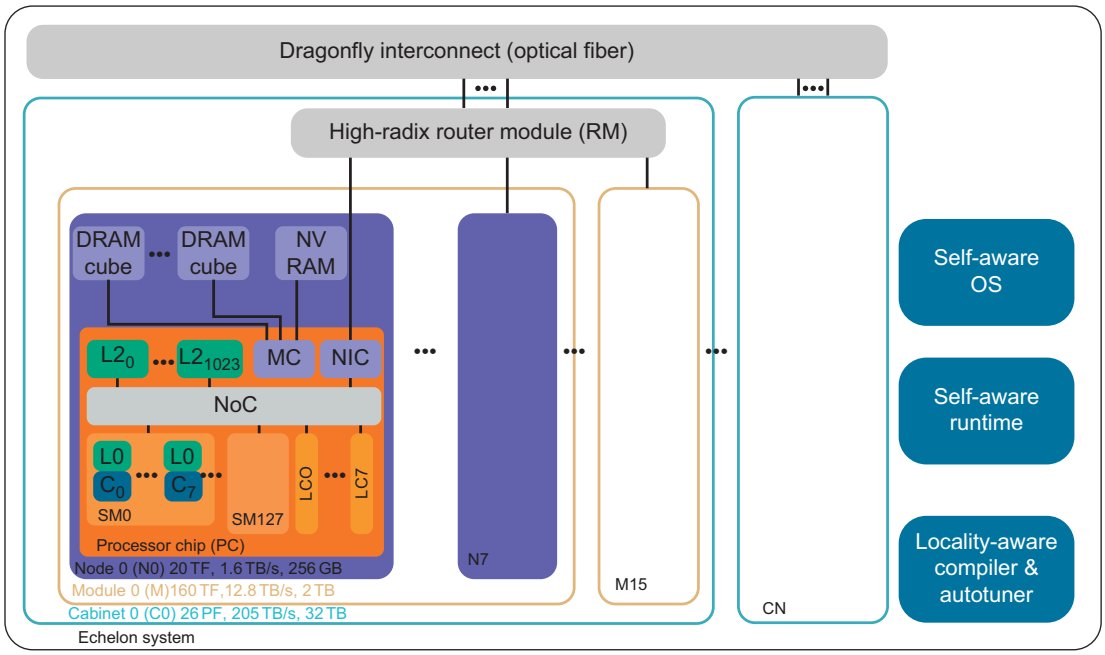

FIGURE 2.12

The architecture of NVIDIA Echelon system built with a hierarchical network of GPUs that can deliver 2.6 Pflops per cabinet, and takes at least N = 400 cabinets to achieve the desired Eflops performance.

(*Courtesy of Bill Dally, Reprinted with Permission [10]*)

Each compute module features a performance of 160 Tflops and 12.8 TB/s over 2 TB of memory. Thus, a single cabinet can house 128 GPU nodes or 16,000 processor cores. Each cabinet has the potential to deliver 2.6 Pflops over 32 TB memory and 205 TB/s bandwidth. The *N* cabinets are interconnected by a Dragonfly network with optical fiber.

To achieve Eflops performance, we need to use at least $N = 400$ cabinets. In total, an Exascale system needs 327,680 processor cores in 400 cabinets. The Echelon system is supported by a self-aware OS and runtime system. The Echelon system is also designed to preserve locality with the support of compiler and autotuner. At present, NVIDIA Fermi (GF110) chip has 512 stream processors. Thus the Echelon design is about 25 times faster. It is highly likely that the Echelon will employ post-Maxwell NVIDIA GPU planned to appear in 2013 ~ 2014 time frame.

2.2.5.4 CUDA Parallel Programming

CUDA (Compute Unified Device Architecture) offers a parallel computing architecture developed by NVIDIA. CUDA is the computing engine in NVIDIA GPUs. This software is accessible to developers through standard programming languages. Programmers use C for CUDA C with NVIDIA extensions and certain restrictions. This CUDA C is compiled through a PathScale Open64 C compiler for parallel execution on a large number of GPU cores. Example 2.4 shows the advantage of using CUDA C in parallel processing.

Example 2.4 Parallel SAXPY Execution Using CUDA C Code on GPUs

SAXPY is a kernel operation frequently performed in matrix multiplication. It essentially performs repeated *multiply and add* operations to generate the dot product of two long vectors. The following `saxpy_serial` routine is written in standard C code. This code is only suitable for sequential execution on a single processor core.

```
Void saxpy_serial (int n, float a, float*x, float *
  { for (int i = 0; i < n; ++i), y[i] = a*x[i] + y[i] }
    // Invoke the serial SAXPY kernel
  saxpy_serial (n, 2.0, x, y);
```

The following `saxpy_parallel` routine is written in CUDA C code for parallel execution by 256 threads/block on many processing cores on the GPU chip. Note that `n` blocks are handled by `n` processing cores, where `n` could be on the order of hundreds of blocks.

```
_global__void saxpy_parallel (int n, float a, float*x, float *y)
  { Int i = blockIndex.x*blockDim.x + threadIndex.x; if (i < n) y [i] = a*x[i] + y[i] }
 // Invoke the parallel SAXPY kernel with 256 threads/block int nblocks = (n + 255)/256;
  saxpy_parallel <<< nblocks, 256 >>> (n, 2.0, x, y);
```

This is a good example of using CUDA C to exploit massive parallelism on a cluster of multicore and multithreaded processors using the CODA GPGPUs as building blocks.

2.2.5.5 CUDA Programming Interfaces

CUDA architecture shares a range of computational interfaces with two competitors: the *Khronos Group*'s *Open Computing Language* and Microsoft's *DirectCompute*. Third-party wrappers are also

available for using Python, Perl, FORTRAN, Java, Ruby, Lua, MATLAB, and IDL. CUDA has been used to accelerate nongraphical applications in computational biology, cryptography, and other fields by an order of magnitude or more. A good example is the BOINC distributed computing client. CUDA provides both a low-level API and a higher-level API. CUDA works with all NVIDIA GPUs from the G8X series onward, including the GeForce, Quadro, and Tesla lines. NVIDIA states that programs developed for the GeForce 8 series will also work without modification on all future NVIDIA video cards due to binary compatibility.

2.2.5.6 Trends in CUDA Usage

Tesla and Fermi are two generations of CUDA architecture released by NVIDIA in 2007 and 2010, respectively. The CUDA version 3.2 is used for using a single GPU module in 2010. A newer CUDA version 4.0 will allow multiple GPUs to address use an unified virtual address space of shared memory. The next NVIDIA GPUs will be Kepler-designed to support C++. The Fermi has eight times the peak double-precision floating-point performance of the Tesla GPU (5.8 Gflops/W versus 0.7 Gflops/W). Currently, the Fermi has up to 512 CUDA cores on 3 billion transistors.

Future applications of the CUDA GPUs and the Echelon system may include the following:

- The search for extraterrestrial intelligence (SETI@Home)
- Distributed calculations to predict the native conformation of proteins
- Medical analysis simulations based on CT and MRI scan images
- Physical simulations in fluid dynamics and environment statistics
- Accelerated 3D graphics, cryptography, compression, and interconversion of video file formats
- Building the *single-chip cloud computer* (SCC) through virtualization in many-core architecture.

2.3 DESIGN PRINCIPLES OF COMPUTER CLUSTERS

Clusters should be designed for scalability and availability. In this section, we will cover the design principles of SSI, HA, fault tolerance, and rollback recovery in general-purpose computers and clusters of cooperative computers.

2.3.1 Single-System Image Features

SSI does not mean a single copy of an operating system image residing in memory, as in an SMP or a workstation. Rather, it means the *illusion* of a single system, single control, symmetry, and transparency as characterized in the following list:

- **Single system** The entire cluster is viewed by users as one system that has multiple processors. The user could say, "Execute my application using five processors." This is different from a distributed system.
- **Single control** Logically, an end user or system user utilizes services from one place with a single interface. For instance, a user submits batch jobs to one set of queues; a system administrator configures all the hardware and software components of the cluster from one control point.

- **Symmetry** A user can use a cluster service from any node. In other words, all cluster services and functionalities are symmetric to all nodes and all users, except those protected by access rights.
- **Location-transparent** The user is not aware of the where abouts of the physical device that eventually provides a service. For instance, the user can use a tape drive attached to any cluster node as though it were physically attached to the local node.

The main motivation to have SSI is that it allows a cluster to be used, controlled, and maintained as a familiar workstation is. The word "single" in "single-system image" is sometimes synonymous with "global" or "central." For instance, a global file system means a single file hierarchy, which a user can access from any node. A single point of control allows an operator to monitor and configure the cluster system. Although there is an illusion of a single system, a cluster service or functionality is often realized in a distributed manner through the cooperation of multiple components. A main requirement (and advantage) of SSI techniques is that they provide both the performance benefits of distributed implementation and the usability benefits of a single image.

From the viewpoint of a process *P*, cluster nodes can be classified into three types. The *home node* of a process *P* is the node where *P* resided when it was created. The *local node* of a process *P* is the node where *P* currently resides. All other nodes are *remote nodes* to *P*. Cluster nodes can be configured to suit different needs. A *host node* serves user logins through Telnet, rlogin, or even FTP and HTTP. A *compute node* is one that performs computational jobs. An *I/O node* is one that serves file I/O requests. If a cluster has large shared disks and tape units, they are normally physically attached to I/O nodes.

There is one home node for each process, which is fixed throughout the life of the process. At any time, there is only one local node, which may or may not be the host node. The local node and remote nodes of a process may change when the process migrates. A node can be configured to provide multiple functionalities. For instance, a node can be designated as a host, an I/O node, and a compute node at the same time. The illusion of an SSI can be obtained at several layers, three of which are discussed in the following list. Note that these layers may overlap with one another.

- **Application software layer** Two examples are parallel web servers and various parallel databases. The user sees an SSI through the application and is not even aware that he is using a cluster. This approach demands the modification of workstation or SMP applications for clusters.
- **Hardware or kernel layer** Ideally, SSI should be provided by the operating system or by the hardware. Unfortunately, this is not a reality yet. Furthermore, it is extremely difficult to provide an SSI over heterogeneous clusters. With most hardware architectures and operating systems being proprietary, only the manufacturer can use this approach.
- **Middleware layer** The most viable approach is to construct an SSI layer just above the OS kernel. This approach is promising because it is platform-independent and does not require application modification. Many cluster job management systems have already adopted this approach.

Each computer in a cluster has its own operating system image. Thus, a cluster may display multiple system images due to the stand-alone operations of all participating node computers. Determining how to merge the multiple system images in a cluster is as difficult as regulating many individual personalities in a community to a single personality. With different degrees of resource sharing, multiple systems could be integrated to achieve SSI at various operational levels.

2.3.1.1 Single Entry Point

Single-system image (SSI) is a very rich concept, consisting of single entry point, single file hierarchy, single I/O space, single networking scheme, single control point, single job management system, single memory space, and single process space. The single entry point enables users to log in (e.g., through Telnet, rlogin, or HTTP) to a cluster as one virtual host, although the cluster may have multiple physical host nodes to serve the login sessions. The system transparently distributes the user's login and connection requests to different physical hosts to balance the load. Clusters could substitute for mainframes and supercomputers. Also, in an Internet cluster server, thousands of HTTP or FTP requests may come simultaneously. Establishing a single entry point with multiple hosts is not a trivial matter. Many issues must be resolved. The following is just a partial list:

- **Home directory** Where do you put the user's home directory?
- **Authentication** How do you authenticate user logins?
- **Multiple connections** What if the same user opens several sessions to the same user account?
- **Host failure** How do you deal with the failure of one or more hosts?

Example 2.5 Realizing a Single Entry Point in a Cluster of Computers

Figure 2.13 illustrates how to realize a single entry point. Four nodes of a cluster are used as host nodes to receive users' login requests. Although only one user is shown, thousands of users can connect to the cluster in the same fashion. When a user logs into the cluster, he issues a standard UNIX command such as *telnet* cluster.cs.hku.hk, using the symbolic name of the cluster system.

The DNS translates the symbolic name and returns the IP address 159.226.41.150 of the least-loaded node, which happens to be node Host1. The user then logs in using this IP address. The DNS periodically receives load information from the host nodes to make load-balancing translation decisions. In the ideal case, if 200 users simultaneously log in, the login sessions are evenly distributed among our hosts with 50 users each. This allows a single host to be four times more powerful.

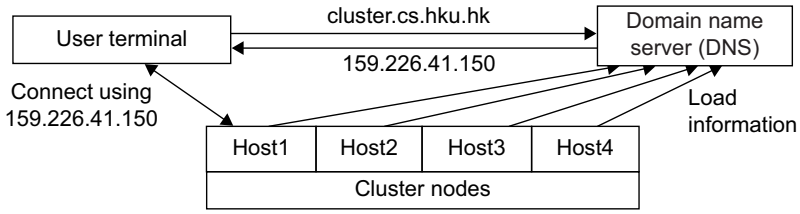

FIGURE 2.13

Realizing a single entry point using a load-balancing domain name system (DNS).

(*Courtesy of Hwang and Xu [14]*)

2.3.1.2 Single File Hierarchy

We use the term "single file hierarchy" in this book to mean the illusion of a single, huge file system image that transparently integrates local and global disks and other file devices (e.g., tapes). In other words, all files a user needs are stored in some subdirectories of the root directory /, and they can be accessed through ordinary UNIX calls such as *open*, *read*, and so on. This should not be confused with the fact that multiple file systems can exist in a workstation as subdirectories of the root directory.

The functionalities of a single file hierarchy have already been partially provided by existing distributed file systems such as *Network File System (NFS)* and *Andrew File System (AFS)*. From the viewpoint of any process, files can reside on three types of locations in a cluster, as shown in Figure 2.14.

Local storage is the disk on the local node of a process. The disks on remote nodes are *remote storage*. A s*table storage* requires two aspects: It is *persistent*, which means data, once written to the stable storage, will stay there for a sufficiently long time (e.g., a week), even after the cluster shuts down; and it is fault-tolerant to some degree, by using redundancy and periodic backup to tapes.

Figure 2.14 uses stable storage. Files in stable storage are called *global files*, those in local storage *local files*, and those in remote storage *remote files.* Stable storage could be implemented as one centralized, large RAID disk. But it could also be distributed using local disks of cluster nodes. The first approach uses a large disk, which is a single point of failure and a potential performance bottleneck. The latter approach is more difficult to implement, but it is potentially more economical, more efficient, and more available. On many cluster systems, it is customary for the system to make visible to the user processes the following directories in a single file hierarchy: the usual *system directories* as in a traditional UNIX workstation, such as /usr and /usr/local; and the user's *home directory* ~/ that has a small disk quota (1–20 MB). The user stores his code files and other files here. But large data files must be stored elsewhere.

- A *global directory* is shared by all users and all processes. This directory has a large disk space of multiple gigabytes. Users can store their large data files here.
- On a cluster system, a process can access a special directory on the local disk. This directory has medium capacity and is faster to access than the global directory.

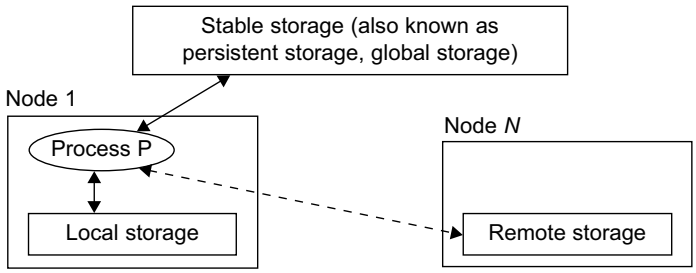

FIGURE 2.14

Three types of storage in a single file hierarchy. Solid lines show what process P can access and the dashed line shows what P may be able to access.

(*Courtesy of Hwang and Xu [14]*)

2.3.1.3 Visibility of Files

The term "visibility" here means a process can use traditional UNIX system or library calls such as *fopen*, *fread*, and *fwrite* to access files. Note that there are multiple local scratch directories in a cluster. The local scratch directories in remote nodes are not in the single file hierarchy, and are not directly visible to the process. A user process can still access them with commands such as *rcp* or some special library functions, by specifying both the node name and the filename.

The name "scratch" indicates that the storage is meant to act as a scratch pad for temporary information storage. Information in the local scratch space could be lost once the user logs out. Files in the global scratch space will normally persist even after the user logs out, but will be deleted by the system if not accessed in a predetermined time period. This is to free disk space for other users. The length of the period can be set by the system administrator, and usually ranges from one day to several weeks. Some systems back up the global scratch space to tapes periodically or before deleting any files.

2.3.1.4 Support of Single-File Hierarchy

It is desired that a single file hierarchy have the SSI properties discussed, which are reiterated for file systems as follows:

- **Single system** There is just one file hierarchy from the user's viewpoint.
- **Symmetry** A user can access the global storage (e.g., */scratch*) using a cluster service from any node. In other words, all file services and functionalities are symmetric to all nodes and all users, except those protected by access rights.
- **Location-transparent** The user is not aware of the whereabouts of the physical device that eventually provides a service. For instance, the user can use a RAID attached to any cluster node as though it were physically attached to the local node. There may be some performance differences, though.

A cluster file system should maintain *UNIX semantics*: Every file operation (*fopen*, *fread*, *fwrite*, *fclose*, etc.) is a transaction. When an *fread* accesses a file after an *fwrite* modifies the same file, the *fread* should get the updated value. However, existing distributed file systems do not completely follow UNIX semantics. Some of them update a file only at close or flush. A number of alternatives have been suggested to organize the global storage in a cluster. One extreme is to use a single file server that hosts a big RAID. This solution is simple and can be easily implemented with current software (e.g., NFS). But the file server becomes both a performance bottleneck and a single point of failure. Another extreme is to utilize the local disks in all nodes to form global storage. This could solve the performance and availability problems of a single file server.

2.3.1.5 Single I/O, Networking, and Memory Space

To achieve SSI, we desire a single control point, a single address space, a single job management system, a single user interface, and a single process control, as depicted in Figure 2.17. In this example, each node has exactly one network connection. Two of the four nodes each have two I/O devices attached.

Single Networking: A properly designed cluster should behave as one system (the shaded area). In other words, it is like a big workstation with four network connections and four I/O devices

attached. Any process on any node can use any network and I/O device as though it were attached to the local node. Single networking means any node can access any network connection.

Single Point of Control: The system administrator should be able to configure, monitor, test, and control the entire cluster and each individual node from a single point. Many clusters help with this through a system console that is connected to all nodes of the cluster. The system console is normally connected to an external LAN (not shown in Figure 2.15) so that the administrator can log in remotely to the system console from anywhere in the LAN to perform administration work.

Note that single point of control does not mean all system administration work should be carried out solely by the system console. In reality, many administrative functions are distributed across the cluster. It means that controlling a cluster should be no more difficult than administering an SMP or a mainframe. It implies that administration-related system information (such as various configuration files) should be kept in one logical place. The administrator monitors the cluster with one graphics tool, which shows the entire picture of the cluster, and the administrator can zoom in and out at will.

Single point of control (or *single point of management*) is one of the most challenging issues in constructing a cluster system. Techniques from distributed and networked system management can be transferred to clusters. Several de facto standards have already been developed for network management. An example is *Simple Network Management Protocol (SNMP)*. It demands an efficient cluster management package that integrates with the availability support system, the file system, and the job management system.

Single Memory Space: *Single memory space* gives users the illusion of a big, centralized main memory, which in reality may be a set of distributed local memory spaces. PVPs, SMPs, and DSMs have an edge over MPPs and clusters in this respect, because they allow a program to utilize all global or local memory space. A good way to test if a cluster has a single memory space is to run a *sequential* program that needs a memory space larger than any single node can provide.

Suppose each node in Figure 2.15 has 2 GB of memory available to users. An ideal single memory image would allow the cluster to execute a sequential program that needs 8 GB of memory. This would enable a cluster to operate like an SMP system. Several approaches have

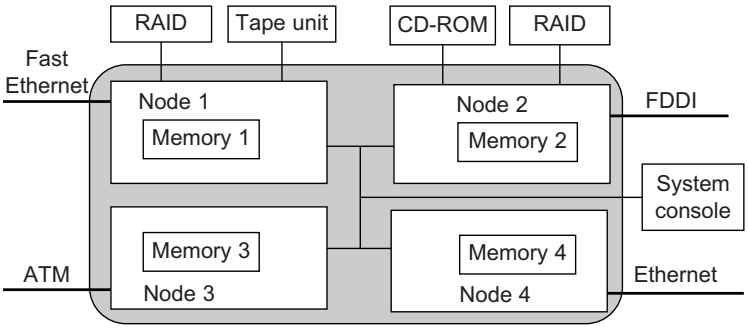

FIGURE 2.15

A cluster with single networking, single I/O space, single memory, and single point of control.

(*Courtesy of Hwang and Xu [14]*)

been attempted to achieve a single memory space on clusters. Another approach is to let the compiler distribute the data structures of an application across multiple nodes. It is still a challenging task to develop a single memory scheme that is efficient, platform-independent, and able to support sequential binary codes.

Single I/O Address Space: Assume the cluster is used as a web server. The web information database is distributed between the two RAIDs. An HTTP daemon is started on each node to handle web requests, which come from all four network connections. A single I/O space implies that any node can access the two RAIDs. Suppose most requests come from the ATM network. It would be beneficial if the functions of the HTTP on node 3 could be distributed to all four nodes. The following example shows a distributed RAID-x architecture for I/O-centric cluster computing [9].

Example 2.6 Single I/O Space over Distributed RAID for I/O-Centric Clusters

A distributed disk array architecture was proposed by Hwang, et al. [9] for establishing a single I/O space in I/O-centric cluster applications. Figure 2.16 shows the architecture for a four-node Linux PC cluster, in which three disks are attached to the SCSI bus of each host node. All 12 disks form an integrated RAID-x with a single address space. In other words, all PCs can access both local and remote disks. The addressing scheme for all disk blocks is interleaved horizontally. Orthogonal stripping and mirroring make it possible to have a RAID-1 equivalent capability in the system.

The shaded blocks are images of the blank blocks. A disk block and its image will be mapped on different physical disks in an orthogonal manner. For example, the block B_0 is located on disk D0. The image block M_0 of block B_0 is located on disk D3. The four disks D0, D1, D2, and D3 are attached to four servers, and thus can be accessed in parallel. Any single disk failure will not lose the data block, because its image is available in recovery. All disk blocks are labeled to show image mapping. Benchmark experiments show that this RAID-x is scalable and can restore data after any single disk failure. The distributed RAID-x has improved aggregate I/O bandwidth in both parallel read and write operations over all physical disks in the cluster.

2.3.1.6 Other Desired SSI Features

The ultimate goal of SSI is for a cluster to be as easy to use as a desktop computer. Here are additional types of SSI, which are present in SMP servers:

- **Single job management system** All cluster jobs can be submitted from any node to a single job management system.
- **Single user interface** The users use the cluster through a single graphical interface. Such an interface is available for workstations and PCs. A good direction to take in developing a cluster GUI is to utilize web technology.
- **Single process space** All user processes created on various nodes form a single process space and share a uniform process identification scheme. A process on any node can create (e.g., through a UNIX fork) or communicate with (e.g., through signals, pipes, etc.) processes on remote nodes.

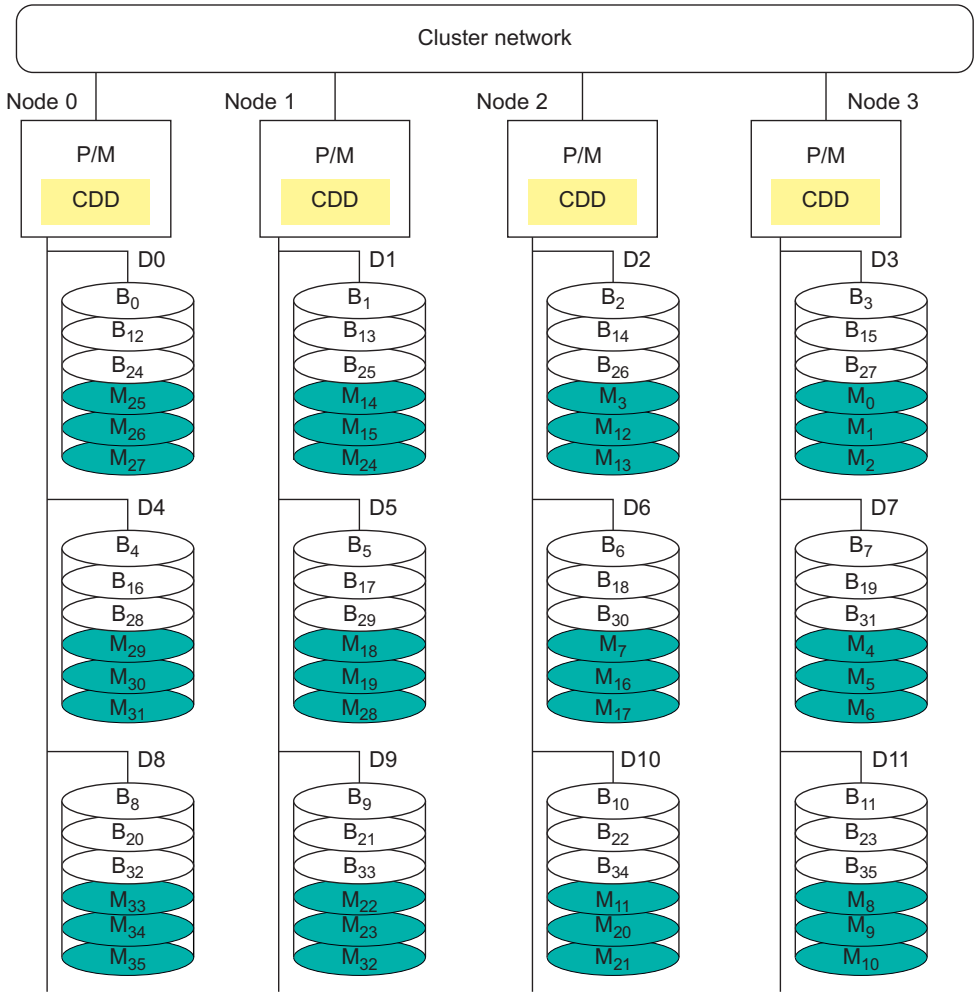

FIGURE 2.16

Distributed RAID architecture with a single I/O space over 12 distributed disks attached to 4 host computers in the cluster (Di stands for Disk i, B_j for disk block j, M_j an image for blocks B_j, P/M for processor/memory node, and CDD for cooperative disk driver.)

(*Courtesy of Hwang, Jin, and Ho [13]*)

- **Middleware support for SSI clustering** As shown in Figure 2.17, various SSI features are supported by middleware developed at three cluster application levels:
- **Management level** This level handles user applications and provides a job management system such as GLUnix, MOSIX, *Load Sharing Facility (LSF)*, or Codine.
- **Programming level** This level provides single file hierarchy (NFS, xFS, AFS, Proxy) and distributed shared memory (TreadMark, Wind Tunnel).

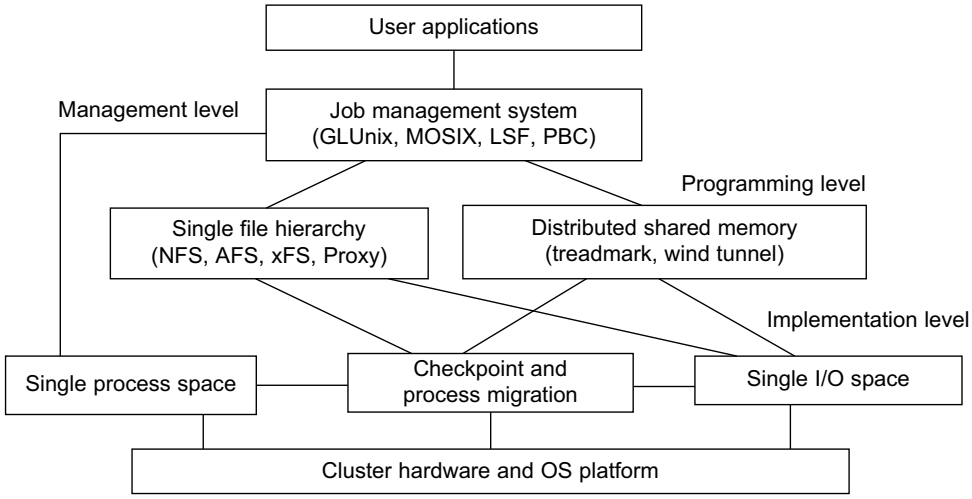

FIGURE 2.17

Relationship among clustering middleware at the job management, programming, and implementation levels.
(*Courtesy of K. Hwang, H. Jin, C.L. Wang and Z. Xu [16]*)

- **Implementation level** This level supports a single process space, checkpointing, process migration, and a single I/O space. These features must interface with the cluster hardware and OS platform. The distributed disk array, RAID-x, in Example 2.6 implements a single I/O space.

2.3.2 High Availability through Redundancy

When designing robust, highly available systems three terms are often used together: *reliability*, *availability*, and *serviceability (RAS)*. Availability is the most interesting measure since it combines the concepts of reliability and serviceability as defined here:

- *Reliability* measures how long a system can operate without a breakdown.
- *Availability* indicates the percentage of time that a system is available to the user, that is, the percentage of system uptime.
- *Serviceability* refers to how easy it is to service the system, including hardware and software maintenance, repair, upgrades, and so on.

The demand for RAS is driven by practical market needs. A recent Find/SVP survey found the following figures among Fortune 1000 companies: An average computer is down nine times per year with an average downtime of four hours. The average loss of revenue per hour of downtime is $82,500. With such a hefty penalty for downtime, many companies are striving for systems that offer *24/365 availability*, meaning the system is available 24 hours per day, 365 days per year.

2.3.2.1 Availability and Failure Rate

As Figure 2.18 shows, a computer system operates normally for a period of time before it fails. The failed system is then repaired, and the system returns to normal operation. This operate-repair cycle

then repeats. A system's reliability is measured by the *mean time to failure* (MTTF), which is the average time of normal operation before the system (or a component of the system) fails. The metric for serviceability is the *mean time to repair* (MTTR), which is the average time it takes to repair the system and restore it to working condition after it fails. The *availability* of a system is defined by:

$$Availability = \text{MTTF}/(\text{MTTF} + \text{MTTR}) \tag{2.1}$$

2.3.2.2 Planned versus Unplanned Failure

When studying RAS, we call any event that prevents the system from normal operation a *failure*. This includes:

- **Unplanned failures** The system breaks, due to an operating system crash, a hardware failure, a network disconnection, human operation errors, a power outage, and so on. All these are simply called failures. The system must be repaired to correct the failure.
- **Planned shutdowns** The system is not broken, but is periodically taken off normal operation for upgrades, reconfiguration, and maintenance. A system may also be shut down for weekends or holidays. The MTTR in Figure 2.18 for this type of failure is the planned downtime.

Table 2.5 shows the availability values of several representative systems. For instance, a conventional workstation has an availability of 99 percent, meaning it is up and running 99 percent of the time or it has a downtime of 3.6 days per year. An optimistic definition of availability does not consider planned downtime, which may be significant. For instance, many supercomputer installations have a planned downtime of several hours per week, while a telephone system cannot tolerate a downtime of a few minutes per year.

2.3.2.3 Transient versus Permanent Failures

A lot of failures are *transient* in that they occur temporarily and then disappear. They can be dealt with without replacing any components. A standard approach is to roll back the system to a known

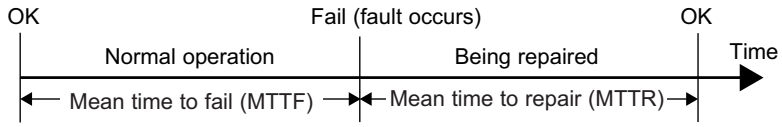

FIGURE 2.18

The operate-repair cycle of a computer system.

Table 2.5 Availability of Computer System Types

System Type	Availability (%)	Downtime in a Year
Conventional workstation	99	3.6 days
HA system	99.9	8.5 hours
Fault-resilient system	99.99	1 hour
Fault-tolerant system	99.999	5 minutes

state and start over. For instance, we all have rebooted our PC to take care of transient failures such as a frozen keyboard or window. *Permanent failures* cannot be corrected by rebooting. Some hardware or software component must be repaired or replaced. For instance, rebooting will not work if the system hard disk is broken.

2.3.2.4 Partial versus Total Failures

A failure that renders the entire system unusable is called a *total failure*. A failure that only affects part of the system is called a *partial failure* if the system is still usable, even at a reduced capacity. A key approach to enhancing availability is to make as many failures as possible partial failures, by systematically removing *single points of failure*, which are hardware or software components whose failure will bring down the entire system.

Example 2.7 Single Points of Failure in an SMP and in Clusters of Computers

In an SMP (Figure 2.19(a)), the shared memory, the OS image, and the memory bus are all single points of failure. On the other hand, the processors are not forming a single point of failure. In a cluster of workstations (Figure 2.19(b)), interconnected by Ethernet, there are multiple OS images, each residing in a workstation. This avoids the single point of failure caused by the OS as in the SMP case. However, the Ethernet network now becomes a single point of failure, which is eliminated in Figure 2.21(c), where a high-speed network is added to provide two paths for communication.

When a node fails in the clusters in Figure 2.19(b) and Figure 2.19(c), not only will the node applications all fail, but also all node data cannot be used until the node is repaired. The shared disk cluster in Figure 2.19(d) provides a remedy. The system stores persistent data on the shared disk, and periodically *checkpoints* to save intermediate results. When one WS node fails, the data will not be lost in this shared-disk cluster.

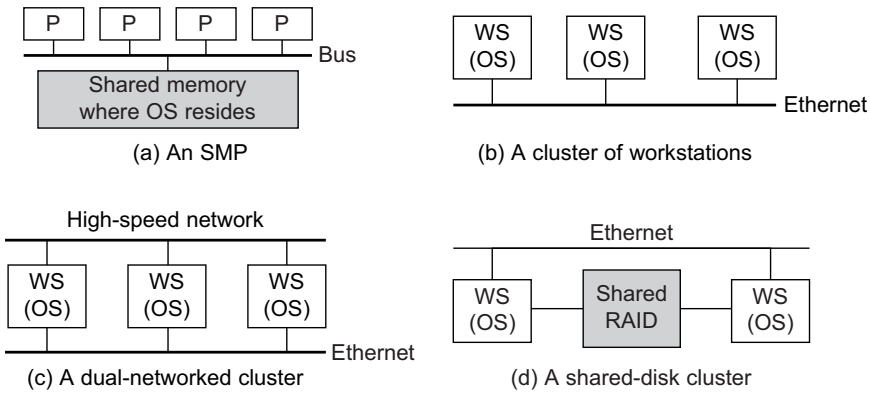

FIGURE 2.19

Single points of failure (SPF) in an SMP and in three clusters, where greater redundancy eliminates more SPFs in systems from (a) to (d).

(*Courtesy of Hwang and Xu [14]*)

2.3.2.5 Redundancy Techniques

Consider the cluster in Figure 2.19(d). Assume only the nodes can fail. The rest of the system (e.g., interconnect and the shared RAID disk) is 100 percent available. Also assume that when a node fails, its workload is switched over to the other node in zero time. We ask, what is the availability of the cluster if planned downtime is ignored? What is the availability if the cluster needs one hour/week for maintenance? What is the availability if it is shut down one hour/week, one node at a time?

According to Table 2.4, a workstation is available 99 percent of the time. The time both nodes are down is only 0.01 percent. Thus, the availability is 99.99 percent. It is now a fault-resilient system, with only one hour of downtime per year. The planned downtime is 52 hours per year, that is, $52 / (365 \times 24) = 0.0059$. The total downtime is now 0.59 percent + 0.01 percent = 0.6 percent. The availability of the cluster becomes 99.4 percent. Suppose we ignore the unlikely situation in which the other node fails while one node is maintained. Then the availability is 99.99 percent.

There are basically two ways to increase the availability of a system: increasing MTTF or reducing MTTR. Increasing MTTF amounts to increasing the reliability of the system. The computer industry has strived to make reliable systems, and today's workstations have MTTFs in the range of hundreds to thousands of hours. However, to further improve MTTF is very difficult and costly. Clusters offer an HA solution based on reducing the MTTR of the system. A multinode cluster has a lower MTTF (thus lower reliability) than a workstation. However, the failures are taken care of quickly to deliver higher availability. We consider several redundancy techniques used in cluster design.

2.3.2.6 Isolated Redundancy

A key technique to improve availability in any system is to use redundant components. When a component (the *primary* component) fails, the service it provided is taken over by another component (the *backup* component). Furthermore, the primary and the backup components should be *isolated* from each other, meaning they should not be subject to the same cause of failure. Clusters provide HA with redundancy in power supplies, fans, processors, memories, disks, I/O devices, networks, operating system images, and so on. In a carefully designed cluster, redundancy is also isolated. Isolated redundancy provides several benefits:

- First, a component designed with isolated redundancy is not a single point of failure, and the failure of that component will not cause a total system failure.
- Second, the failed component can be repaired while the rest of the system is still working.
- Third, the primary and the backup components can mutually test and debug each other.

The IBM SP2 communication subsystem is a good example of isolated-redundancy design. All nodes are connected by two networks: an Ethernet network and a high-performance switch. Each node uses two separate interface cards to connect to these networks. There are two communication protocols: a standard IP and a *user-space* (US) protocol; each can run on either network. If either network or protocol fails, the other network or protocol can take over.

2.3.2.7 N-Version Programming to Enhance Software Reliability

A common isolated-redundancy approach to constructing a mission-critical software system is called *N-version programming*. The software is implemented by N isolated teams who may not even know the others exist. Different teams are asked to implement the software using different algorithms, programming languages, environment tools, and even platforms. In a fault-tolerant system, the

N versions all run simultaneously and their results are constantly compared. If the results differ, the system is notified that a fault has occurred. But because of isolated redundancy, it is extremely unlikely that the fault will cause a majority of the *N* versions to fail at the same time. So the system continues working, with the correct result generated by majority voting. In a highly available but less mission-critical system, only one version needs to run at a time. Each version has a built-in self-test capability. When one version fails, another version can take over.

2.3.3 Fault-Tolerant Cluster Configurations

The cluster solution was targeted to provide availability support for two server nodes with three ascending levels of availability: *hot standby*, *active takeover*, and *fault-tolerant*. In this section, we will consider the *recovery time*, *failback feature*, and *node activeness*. The level of availability increases from standby to active and fault-tolerant cluster configurations. The shorter is the recovery time, the higher is the cluster availability. *Failback* refers to the ability of a recovered node returning to normal operation after repair or maintenance. *Activeness* refers to whether the node is used in active work during normal operation.

- **Hot standby server clusters** In a *hot standby* cluster, only the primary node is actively doing all the useful work normally. The standby node is powered on (hot) and running some monitoring programs to communicate heartbeat signals to check the status of the primary node, but is not actively running other useful workloads. The primary node must mirror any data to shared disk storage, which is accessible by the standby node. The standby node requires a second copy of data.
- **Active-takeover clusters** In this case, the architecture is symmetric among multiple server nodes. Both servers are primary, doing useful work normally. Both failover and failback are often supported on both server nodes. When a node fails, the user applications fail over to the available node in the cluster. Depending on the time required to implement the failover, users may experience some delays or may lose some data that was not saved in the last checkpoint.
- **Failover cluster** This is probably the most important feature demanded in current clusters for commercial applications. When a component fails, this technique allows the remaining system to take over the services originally provided by the failed component. A failover mechanism must provide several functions, such as *failure diagnosis*, *failure notification*, and *failure recovery*. Failure diagnosis refers to the detection of a failure and the location of the failed component that caused the failure. A commonly used technique is *heartbeat*, whereby the cluster nodes send out a stream of heartbeat messages to one another. If the system does not receive the stream of heartbeat messages from a node, it can conclude that either the node or the network connection has failed.

Example 2.8 Failure Diagnosis and Recovery in a Dual-Network Cluster

A cluster uses two networks to connect its nodes. One node is designated as the *master node*. Each node has a *heartbeat daemon* that periodically (every 10 seconds) sends a heartbeat message to the master

node through both networks. The master node will detect a failure if it does not receive messages for a beat (10 seconds) from a node and will make the following diagnoses:

- A node's connection to one of the two networks failed if the master receives a heartbeat from the node through one network but not the other.
- The node failed if the master does not receive a heartbeat through either network. It is assumed that the chance of both networks failing at the same time is negligible.

The failure diagnosis in this example is simple, but it has several pitfalls. What if the master node fails? Is the 10-second heartbeat period too long or too short? What if the heartbeat messages are dropped by the network (e.g., due to network congestion)? Can this scheme accommodate hundreds of nodes? Practical HA systems must address these issues. A popular trick is to use the heartbeat messages to carry load information so that when the master receives the heartbeat from a node, it knows not only that the node is alive, but also the resource utilization status of the node. Such load information is useful for load balancing and job management.

Once a failure is diagnosed, the system notifies the components that need to know the failure event. Failure notification is needed because the master node is not the only one that needs to have this information. For instance, in case of the failure of a node, the DNS needs to be told so that it will not connect more users to that node. The resource manager needs to reassign the workload and to take over the remaining workload on that node. The system administrator needs to be alerted so that she can initiate proper actions to repair the node.

2.3.3.1 Recovery Schemes

Failure recovery refers to the actions needed to take over the workload of a failed component. There are two types of recovery techniques. In *backward recovery*, the processes running on a cluster periodically save a consistent state (called a *checkpoint*) to a stable storage. After a failure, the system is reconfigured to isolate the failed component, restores the previous checkpoint, and resumes normal operation. This is called *rollback*.

Backward recovery is relatively easy to implement in an application-independent, portable fashion, and has been widely used. However, rollback implies wasted execution. If execution time is crucial, such as in real-time systems where the rollback time cannot be tolerated, a *forward recovery* scheme should be used. With such a scheme, the system is not rolled back to the previous checkpoint upon a failure. Instead, the system utilizes the failure diagnosis information to reconstruct a valid system state and continues execution. Forward recovery is application-dependent and may need extra hardware.

Example 2.9 MTTF, MTTR, and Failure Cost Analysis

Consider a cluster that has little availability support. Upon a node failure, the following sequence of events takes place:

1. The entire system is shut down and powered off.
2. The faulty node is replaced if the failure is in hardware.
3. The system is powered on and rebooted.
4. The user application is reloaded and rerun from the start.

Assume one of the cluster nodes fails every 100 hours. Other parts of the cluster never fail. Steps 1 through 3 take two hours. On average, the mean time for step 4 is two hours. What is the availability of the cluster? What is the yearly failure cost if each one-hour downtime costs $82,500?

Solution: The cluster's MTTF is 100 hours; the MTTR is 2 + 2 = 4 hours. According to Table 2.5, the availability is 100/104 = 96.15 percent. This corresponds to 337 hours of downtime in a year, and the failure cost is $82500 × 337, that is, more than $27 million.

Example 2.10 Availability and Cost Analysis of a Cluster of Computers

Repeat Example 2.9, but assume that the cluster now has much increased availability support. Upon a node failure, its workload automatically fails over to other nodes. The failover time is only six minutes. Meanwhile, the cluster has *hot swap* capability: The faulty node is taken off the cluster, repaired, replugged, and rebooted, and it rejoins the cluster, all without impacting the rest of the cluster. What is the availability of this ideal cluster, and what is the yearly failure cost?

Solution: The cluster's MTTF is still 100 hours, but the MTTR is reduced to 0.1 hours, as the cluster is available while the failed node is being repaired. From to Table 2.5, the availability is 100/100.5 = 99.9 percent. This corresponds to 8.75 hours of downtime per year, and the failure cost is $82,500, a 27M/722K = 38 times reduction in failure cost from the design in Example 3.8.

2.3.4 Checkpointing and Recovery Techniques

Checkpointing and recovery are two techniques that must be developed hand in hand to enhance the availability of a cluster system. We will start with the basic concept of checkpointing. This is the process of periodically saving the state of an executing program to stable storage, from which the system can recover after a failure. Each program state saved is called a *checkpoint*. The disk file that contains the saved state is called the *checkpoint file*. Although all current checkpointing software saves program states in a disk, research is underway to use node memories in place of stable storage in order to improve performance.

Checkpointing techniques are useful not only for availability, but also for program debugging, process migration, and load balancing. Many job management systems and some operating systems support checkpointing to a certain degree. The Web Resource contains pointers to numerous checkpoint-related web sites, including some public domain software such as Condor and Libckpt. Here we will present the important issues for the designer and the user of checkpoint software. We will first consider the issues that are common to both sequential and parallel programs, and then we will discuss the issues pertaining to parallel programs.

2.3.4.1 Kernel, Library, and Application Levels

Checkpointing can be realized by the operating system at the *kernel level*, where the OS transparently checkpoints and restarts processes. This is ideal for users. However, checkpointing is not supported in most operating systems, especially for parallel programs. A less transparent approach links the user code with a checkpointing library in the user space. Checkpointing and restarting are

handled by this runtime support. This approach is used widely because it has the advantage that user applications do not have to be modified.

A main problem is that most current checkpointing libraries are static, meaning the application source code (or at least the object code) must be available. It does not work if the application is in the form of executable code. A third approach requires the user (or the compiler) to insert checkpointing functions in the application; thus, the application has to be modified, and the transparency is lost. However, it has the advantage that the user can specify where to checkpoint. This is helpful to reduce checkpointing overhead. Checkpointing incurs both time and storage overheads.

2.3.4.2 Checkpoint Overheads

During a program's execution, its states may be saved many times. This is denoted by the time consumed to save one checkpoint. The storage overhead is the extra memory and disk space required for checkpointing. Both time and storage overheads depend on the size of the checkpoint file. The overheads can be substantial, especially for applications that require a large memory space. A number of techniques have been suggested to reduce these overheads.

2.3.4.3 Choosing an Optimal Checkpoint Interval

The time period between two checkpoints is called the *checkpoint interval*. Making the interval larger can reduce checkpoint time overhead. However, this implies a longer computation time after a failure. Wong and Franklin [28] derived an expression for optimal checkpoint interval as illustrated in Figure 2.20.

$$\textit{Optimal checkpoint interval} = \text{Square root}\,(MTTF \times t_c)/h \tag{2.2}$$

Here, MTTF is the system's *mean time to failure*. This MTTF accounts the time consumed to save one checkpoint, and h is the average percentage of normal computation performed in a checkpoint interval before the system fails. The parameter h is always in the range. After a system is restored, it needs to spend $h \times$ (checkpoint interval) time to recompute.

2.3.4.4 Incremental Checkpoint

Instead of saving the full state at each checkpoint, an *incremental checkpoint* scheme saves only the portion of the state that is changed from the previous checkpoint. However, care must be taken regarding *old* checkpoint files. In full-state checkpointing, only one checkpoint file needs to be kept on disk. Subsequent checkpoints simply overwrite this file. With incremental checkpointing, old files needed to be kept, because a state may span many files. Thus, the total storage requirement is larger.

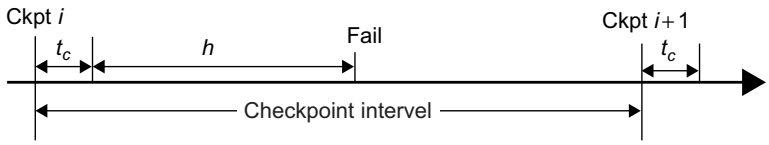

FIGURE 2.20

Time parameters between two checkpoints.

(*Courtesy of Hwang and Xu [14]*)

2.3.4.5 Forked Checkpointing

Most checkpoint schemes are blocking in that the normal computation is stopped while checkpointing is in progress. With enough memory, checkpoint overhead can be reduced by making a copy of the program state in memory and invoking another asynchronous thread to perform the checkpointing concurrently. A simple way to overlap checkpointing with computation is to use the UNIX *fork( )* system call. The forked child process duplicates the parent process's address space and checkpoints it. Meanwhile, the parent process continues execution. Overlapping is achieved since checkpointing is disk-I/O intensive. A further optimization is to use the copy-on-write mechanism.

2.3.4.6 User-Directed Checkpointing

The checkpoint overheads can sometimes be substantially reduced if the user inserts code (e.g., library or system calls) to tell the system when to save, what to save, and what not to save. What should be the exact contents of a checkpoint? It should contain just enough information to allow a system to recover. The state of a process includes its data state and control state. For a UNIX process, these states are stored in its address space, including the text (code), the data, the stack segments, and the process descriptor. Saving and restoring the full state is expensive and sometimes impossible.

For instance, the process ID and the parent process ID are not restorable, nor do they need to be saved in many applications. Most checkpointing systems save a partial state. For instance, the code segment is usually not saved, as it does not change in most applications. What kinds of applications can be checkpointed? Current checkpoint schemes require programs to be *well behaved*, the exact meaning of which differs in different schemes. At a minimum, a well-behaved program should not need the exact contents of state information that is not restorable, such as the numeric value of a process ID.

2.3.4.7 Checkpointing Parallel Programs

We now turn to checkpointing parallel programs. The state of a parallel program is usually much larger than that of a sequential program, as it consists of the set of the states of individual processes, plus the state of the communication network. Parallelism also introduces various timing and consistency problems.

Example 2.11 Checkpointing a Parallel Program

Figure 2.21 illustrates checkpointing of a three-process parallel program. The arrows labeled *x*, *y*, and *z* represent point-to-point communication among the processes. The three thick lines labeled *a*, *b*, and *c* represent three *global snapshots* (or simply *snapshots*), where a global snapshot is a set of checkpoints

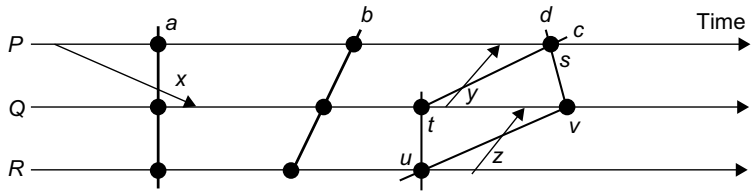

FIGURE 2.21

Consistent and inconsistent checkpoints in a parallel program.

(*Courtesy of Hwang and Xu [14]*)