Department of Computer Science and Engineering

CLOUD COMPUTING (BCS601)

Module-01

Distributed System Models and Enabling Technologies

Scalable Computing Over the Internet

Scalable Computing over the Internet refers to the ability to dynamically allocate and manage computing resources over the internet in a way that can handle growing demands. This involves distributing computational tasks across multiple systems (often using cloud computing or distributed computing platforms) to accommodate varying workloads.

Evolution of Computing Technology

- Over the last 60 years, computing has evolved through multiple platforms and environments.
- Shift from centralized computing to parallel and distributed systems.
- Modern computing relies on data-intensive and network-centric architectures.

The Age of Internet Computing

- High-performance computing (HPC) systems cater to large-scale computational needs.
- High-throughput computing (HTC) focuses on handling a high number of simultaneous tasks.
- The shift from Linpack Benchmark to HTC systems for measuring performance.

Platform Evolution

- **First Generation (1950-1970):** Mainframes like IBM 360 and CDC 6400.
- **Second Generation (1960-1980):** Minicomputers like DEC PDP 11 and VAX.
- **Third Generation (1970-1990):** Personal computers with VLSI microprocessors.
- **Fourth Generation (1980-2000):** Portable and wireless computing devices.
- **Fifth Generation (1990-present):** HPC and HTC systems in clusters, grids, and cloud computing.

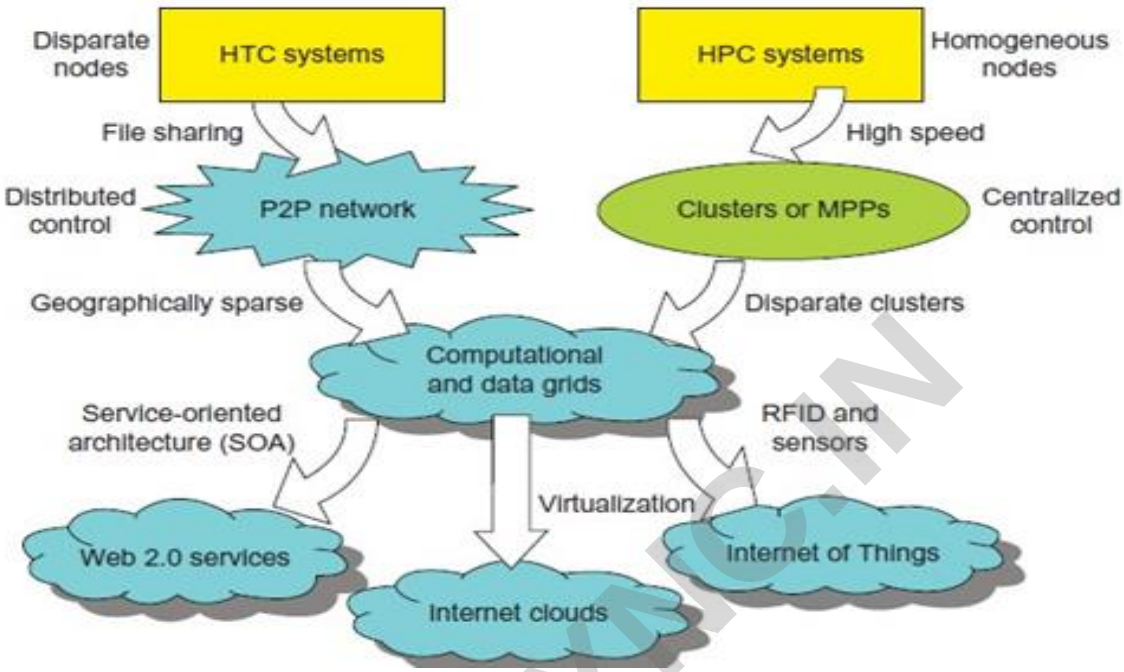


FIGURE 1.1 Evolutionary trend toward parallel, distributed, and cloud computing with clusters, MPPs, P2P networks, grids, clouds, web services, and the Internet of Things.

High-Performance Computing (HPC)

- Focused on raw speed, measured in floating-point operations per second (FLOPS).
- Used mainly in scientific, engineering, and industrial applications.
- Limited to a small number of specialized users.

High-Throughput Computing (HTC)

- Shift from HPC to HTC for market-oriented applications.
- Used in Internet searches, web services, and enterprise computing.
- Emphasis on cost reduction, energy efficiency, security, and reliability.

Emerging Computing Paradigms

- Service-Oriented Architecture (SOA): Enables Web 2.0 services.
- Virtualization: Key technology for cloud computing.
- Internet of Things (IoT): Enabled by RFID, GPS, and sensor technologies.
- Cloud Computing: Evolution of computing as a utility.

Computing Paradigm Distinctions

- Centralized Computing: All resources in one system.
- Parallel Computing: Processors work simultaneously in a shared-memory or distributed-memory setup.
- Distributed Computing: Multiple autonomous computers communicate over a network.
- Cloud Computing: Uses both centralized and distributed computing over data centers.

Distributed System Families

- **Clusters:** Homogeneous compute nodes working together.
- **Grids:** Wide-area distributed computing infrastructures.
- **P2P Networks:** Client machines globally distributed for file sharing and content delivery.
- **Cloud Computing:** Utilizes clusters, grids, and P2P technologies.

Future Computing Needs and Design Objectives

- **Efficiency:** Maximizing parallelism, job throughput, and power efficiency.
- **Dependability:** Ensuring reliability and Quality of Service (QoS).
- **Adaptation:** Scaling to billions of requests over vast data sets.
- **Flexibility**: Supporting both HPC (scientific/engineering) and HTC (business) applications.

Scalable Computing Trends and New Paradigms

Computing Trends and Parallelism

- Technological progress drives computing applications, as seen in Moore's Law (processor speed doubling every 18 months) and Gilder's Law (network bandwidth doubling yearly).
- Commodity hardware advancements, driven by personal computing markets, have influenced large-scale computing.

Degrees of Parallelism (DoP):

- **Bit-level (BLP):** Transition from 4-bit to 64-bit CPUs.
- **Instruction-level (ILP):** Techniques like pipelining, superscalar processing, and multithreading.
- **Data-level (DLP):** SIMD and vector processing for efficient parallel execution.
- **Task-level (TLP):** Parallel tasks on multicore processors, though challenging to program.

- **Job-level (JLP):** High-level parallelism in distributed systems, integrating fine- grain parallelism.

Innovative Applications of Parallel and Distributed Systems

Transparency in **data access, resource allocation, job execution, and failure recovery** is essential.

Application domains:

- **Banking and finance:** Distributed transaction processing and data consistency challenges.
- **Science, engineering, healthcare, and web services:** Demand scalable and reliable computing.
- Challenges include **network saturation, security threats, and lack of software support.**

OR

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 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.

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.

Utility Computing and Cloud Adoption

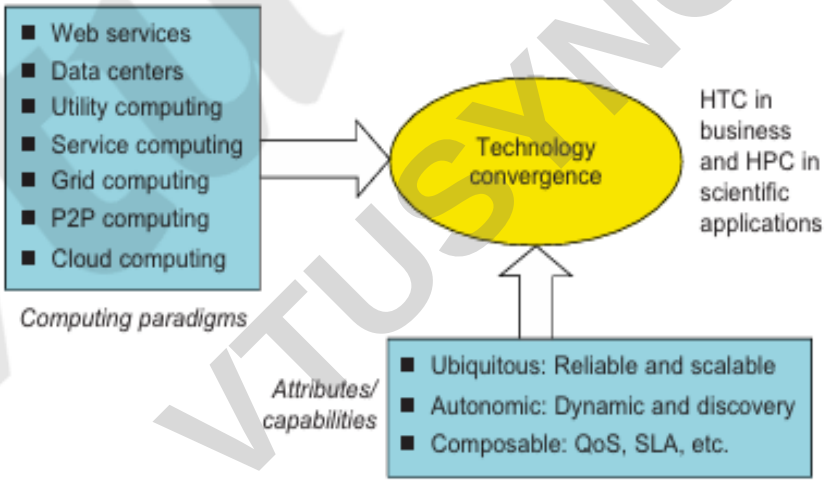


FIGURE 1.2

The vision of computer utilities in modern distributed computing systems.

(Modified from presentation slide by Raj Buyya, 2010)

- **Utility computing:** Provides computing resources as a paid service (grid/cloud platforms).
- **Cloud computing** extends utility computing, leveraging distributed resources and virtualized environments.
- **Challenges:** Efficient processors, scalable memory/storage, distributed OS, middleware, and new programming models.

Hype Cycle of Emerging Technologies

• New technologies go through five stages:
  o Innovation trigger → Peak of inflated expectations → Disillusionment → Enlightenment → Productivity plateau.
• **2010 Predictions:**
  - Cloud computing was expected to mature in 2-5 years.
  - 3D printing was 5-10 years from mainstream adoption.
  - Mesh network sensors were more than 10 years from maturity.
  - Broadband over power lines was expected to become obsolete.
• **Promising technologies:** Cloud computing, biometric authentication, interactive TV, speech recognition, predictive analytics, and media tablets.

The Internet of Things and Cyber-Physical Systems

The Internet of Things (IoT)

- IoT extends the Internet to everyday objects, interconnecting devices, tools, and computers via sensors, RFID, and GPS.
- **History:** Introduced in 1999 at MIT, IoT enables communication between objects and people.
- **IPv6 Impact:** With 2^{128} IP addresses, IoT can assign unique addresses to all objects, tracking up to 100 trillion static or moving objects.

Communication Models:

- **H2H (Human-to-Human)**
- **H2T (Human-to-Thing)**
- **T2T (Thing-to-Thing)**

Development & Challenges:

- IoT is in its early stages, mainly advancing in Asia and Europe.
- Cloud computing is expected to enhance efficiency, intelligence, and scalability in IoT interactions.

Smart Earth Vision: IoT aims to create intelligent cities, clean energy, better healthcare, and sustainable environments.

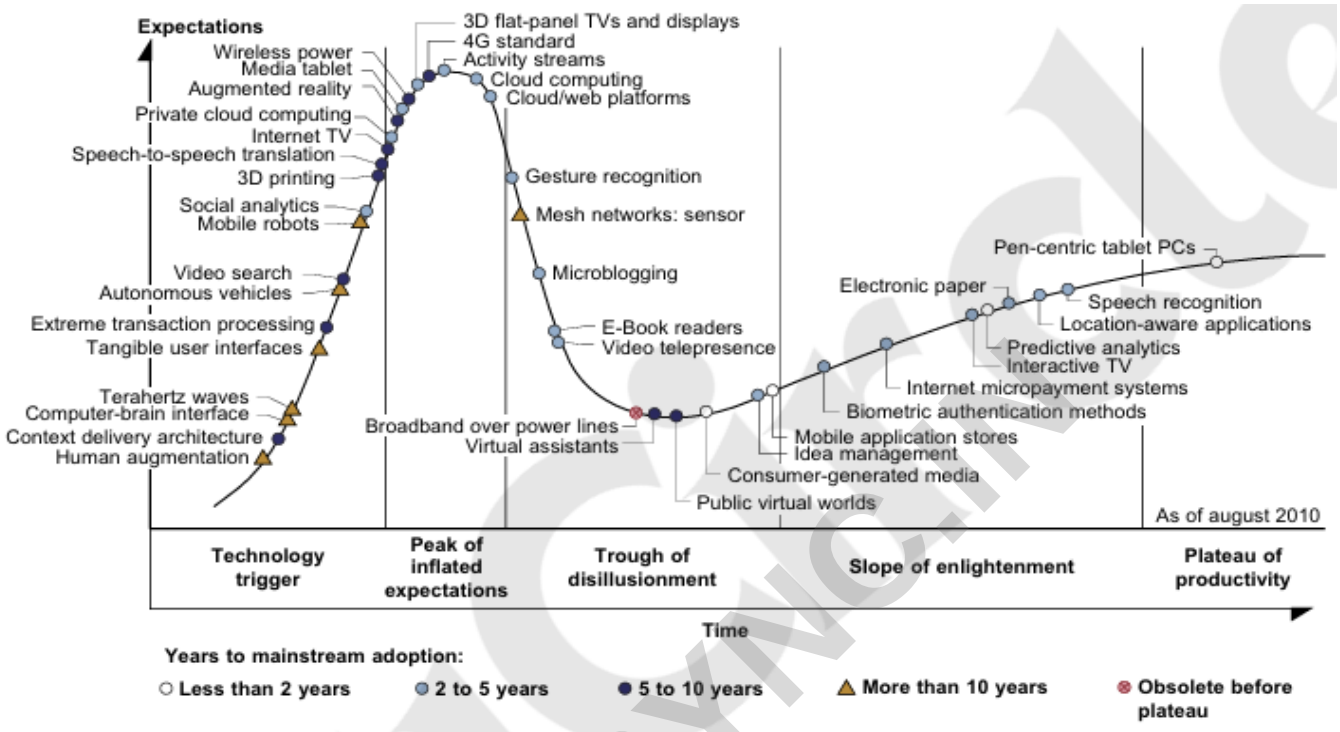


FIGURE 1.3
Hype cycle for Emerging Technologies, 2010.

Cyber-Physical Systems (CPS)

CPS integrates computation, communication, and control (3C) into a closed intelligent feedback system between the physical and digital worlds.

Features:

- IoT vs. CPS: IoT focuses on networked objects, while CPS focuses on VR applications in the real world.
- CPS enhances automation, intelligence, and interactivity in physical environments.

Development:

- Actively researched in the United States.
- Expected to revolutionize real-world interactions just as the Internet transformed virtual interactions.

Technologies for Network Based Systems

Introduction to Distributed Computing Technologies

Discusses hardware, software, and network technologies for distributed computing. Focuses on designing distributed operating systems for handling massive parallelism.

Advances in CPU Processors

- Modern CPUs use multicore architecture (dual, quad, six, or more cores).
- Instruction-Level Parallelism (ILP) and Thread-Level Parallelism (TLP) improve performance.

• **Processor speed evolution:**

- 1 MIPS (VAX 780, 1978) → 1,800 MIPS (Intel Pentium 4, 2002) → 22,000 MIPS (Sun Niagara 2,2008).

• **Moore's Law** holds for CPU growth, but clock rates are limited (~5 GHz max) due to heat and power constraints.

• **Modern CPU technologies include:**

- Superscalar architecture, dynamic branch prediction, speculative execution.
- Multithreaded CPUs (e.g., Intel i7, AMD Opteron, Sun Niagara, IBM Power 6).

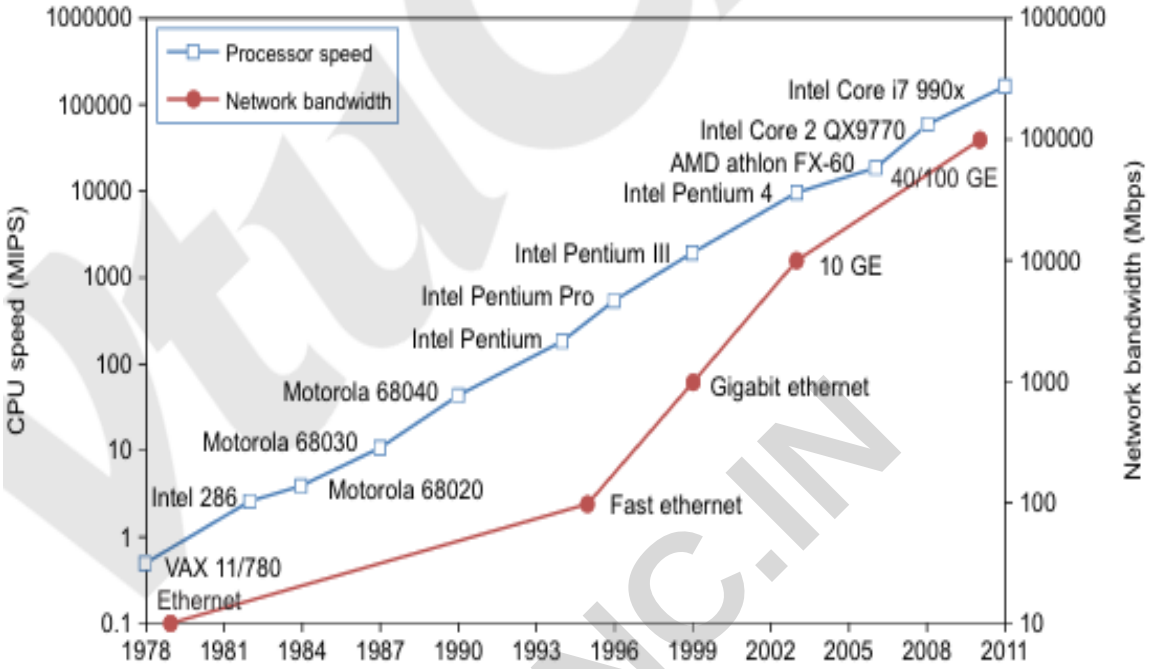


FIGURE 1.4

Improvement in processor and network technologies over 33 years.

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

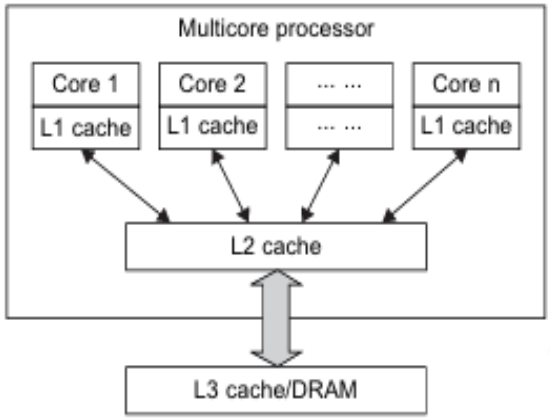


FIGURE 1.5

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.

The hierarchy of caches plays a crucial role in optimizing the performance of multicore CPUs and many-core GPUs by reducing memory access latency.

Cache Hierarchy Overview:

Caches are small, fast storage layers closer to the processor than the main memory. A typical hierarchy includes:

- **L1 Cache:** Closest to the processor core; fastest but smallest in size.
- **L2 Cache:** Larger than L1 but slightly slower; shared by one or more cores.
- **L3 Cache:** Even larger and slower; usually shared among all cores in a processor.

How it Works:

- **Data Locality:** Frequently accessed data is stored in the caches, exploiting spatial and temporal locality to speed up access.
- **Multilevel Approach:** The cache hierarchy ensures faster access to data by maintaining different levels of speed and size. L1 handles immediate needs, while L2 and L3 back it up.
- **Coherence and Sharing:** In multicore systems, maintaining cache coherence is vital. Protocols like MESI (Modified, Exclusive, Shared, Invalid) ensure data consistency across cores.

In GPUs:

- GPUs also employ cache hierarchies, but their design is optimized for parallel workloads.
    - **Shared Memory:** A fast, programmable memory shared among threads within a core.
    - **L1 and L2 Caches:** Designed to handle specific workloads like texture and global memory accesses efficiently.

In both CPUs and GPUs, cache hierarchies reduce dependence on the slower main memory, improving overall performance.

Multicore CPU and Many-Core GPU Architectures

- CPUs may scale to hundreds of cores but face memory wall limitations.
- GPUs (Graphics Processing Units) are designed for massive parallelism and data-level parallelism (DLP).
- x86-based processors dominate HPC and HTC systems.
- Trend towards heterogeneous processors combining CPU and GPU cores on a single chip.

Multithreading Technology

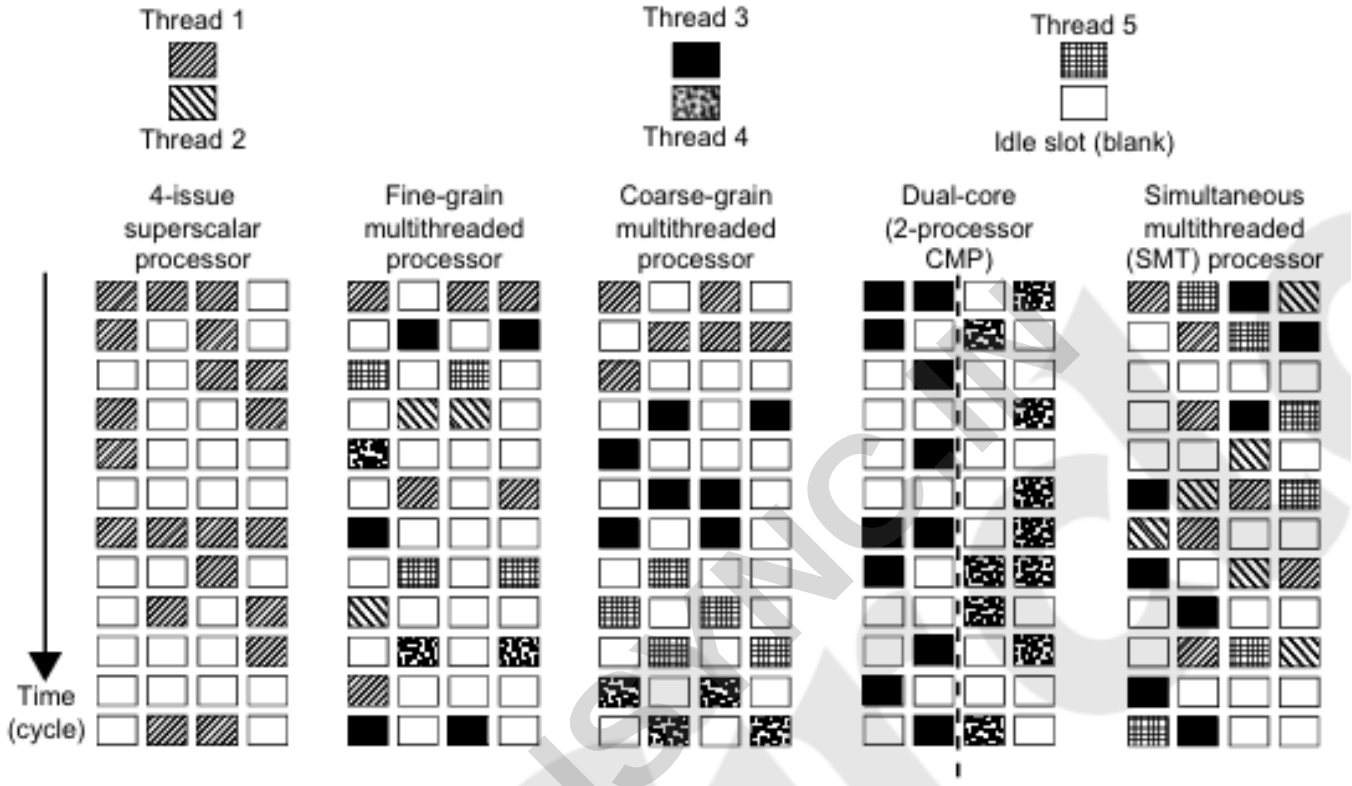


FIGURE 1.6

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

- Each processor category employs different scheduling patterns and techniques to exploit Instruction-Level Parallelism (ILP) and Thread-Level Parallelism (TLP).
- Fine-grain and coarse-grain multithreading focus on reducing idle cycles, while superscalar, CMP, and SMT enhance parallel execution in various ways.
- Blank slots in the functional units represent inefficiency, with SMT generally showing the least idle time.

1. **Four-Issue Superscalar Processor:**
   - Single-threaded processor with four functional units.
   - Executes only instructions from a single thread at any given time.
   - Limited by the availability of instructions from the same thread.
2. **Fine-Grain Multithreaded Processor:**
   - Switches between different threads at every clock cycle.

- Ensures maximum utilization of functional units by avoiding stalls due to single-thread dependencies.
- Suitable for workloads with many independent threads.

3. **Coarse-Grain Multithreaded Processor:**
   - Executes multiple instructions from the same thread over several cycles before switching to another thread.
   - Reduces thread-switching overhead but may leave functional units idle during thread stalls.
4. **Dual-Core (2-Processor Chip Multiprocessing - CMP):**
   - Contains two independent processing cores, each functioning as a two-way superscalar processor.
   - Executes instructions from different threads on separate cores, fully exploiting thread-level parallelism (TLP).
5. **Simultaneous Multithreaded (SMT) Processor:**
   - Allows multiple threads to execute simultaneously on a single core by sharing functional units.
   - Maximizes throughput by interleaving instructions from different threads in the same cycle.

GPU Computing and Exascale Systems

• GPUs were initially graphics accelerators, now widely used for HPC and AI.

• First GPU: NVIDIA GeForce 256 (1999).

• Modern GPUs have hundreds of cores, e.g., NVIDIA CUDA Tesla.

• GPGPU (General-Purpose GPU Computing) enables parallel processing beyond graphics.

How GPUs Work

• Early GPUs functioned as CPU coprocessors.

• Modern GPUs have 128+ cores, each handling multiple threads.

• GPUs optimize throughput, CPUs optimize latency.

• Used in supercomputers, AI, deep learning, gaming, and mobile devices.

GPU Programming Model

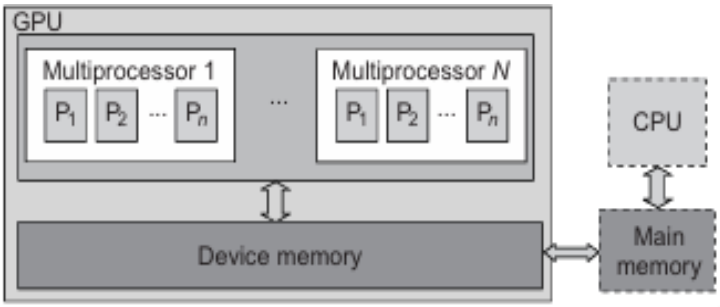


FIGURE 1.7

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

This section highlights the interaction between a CPU and a GPU for parallel execution of floating-point operations:

1. **CPU as the Controller:**
   - The CPU, with its multicore architecture, has limited parallelism.
   - It offloads floating-point kernel computations to the GPU for massive data processing.
2. **GPU as the Workhorse:**
   - The GPU features a many-core architecture with hundreds of simple processing cores grouped into multiprocessors.
   - Each core supports multiple threads, enabling extensive parallel execution.
3. **Key Process:**
   - The CPU instructs the GPU to perform large-scale computations.
   - Efficient communication requires matching bandwidths between the on-board main memory and the GPU's on-chip memory.
4. **NVIDIA CUDA Framework:**
   - This process is commonly implemented using CUDA programming.
   - Examples of GPUs for such tasks include NVIDIA's GeForce 8800, Tesla, and Fermi GPUs.

The setup ensures that the computational workload is distributed effectively, leveraging the strengths of both CPU and GPU architectures.

Example : the NVIDIA Fermi GPU Chip with 512 CUDA Cores

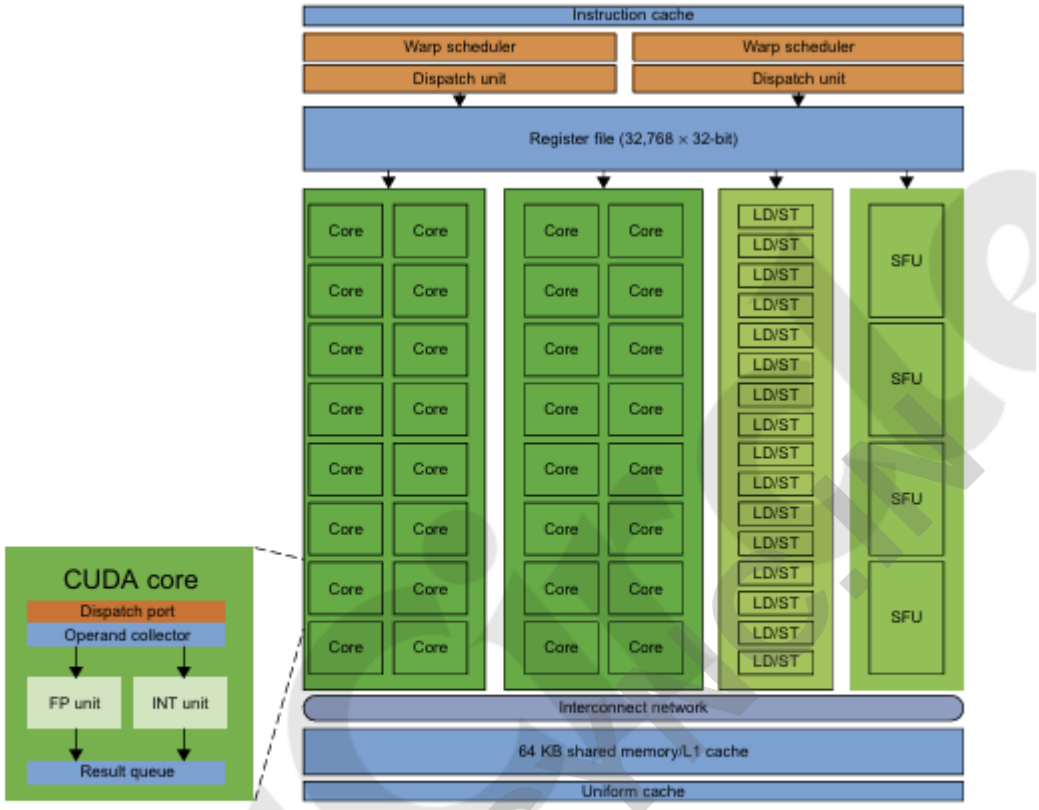


FIGURE 1.8

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].

The NVIDIA Fermi GPU architecture is designed for high performance, efficiency, and programmability. Here's a quick summary:

- **Structure:**
  - Built with 16 streaming multiprocessors (SMs), each containing 32 CUDA cores, for a total of 512 CUDA cores.
  - Each SM supports parallel execution, enabling massive computational power.
- **Key Features:**
  - Optimized for floating-point operations and parallel processing.
  - Allows for high memory bandwidth between on-chip GPU memory and external device memory.
  - Enhanced programmability using NVIDIA's CUDA platform.

Power Efficiency of the GPU

- **Power Efficiency:**
  - GPUs consume significantly less power per instruction compared to CPUs.
  - This efficiency makes GPUs more suitable for large-scale, power-constrained systems like exascale computing.
- **Performance and Parallelism:**
  - GPUs are optimized for high throughput and parallel processing, with explicit on-chip memory management.
  - CPUs are designed for latency optimization in caches and memory, limiting their parallelism.
- **Performance/Power Ratio:**
  - In 2010, GPUs achieved a performance of 5 Gflops/watt per core, far exceeding CPUs at less than 1 Gflop/watt.
  - Exascale computing requires an estimated 60 Gflops/watt per core, highlighting the need for further GPU advancements.
- **Challenges and Future Directions:**
  - Power constraints and data movement dominate system limitations.
  - Solutions include optimizing storage hierarchy, designing application-specific memory, developing self-aware OS and runtime support, and building locality-aware compilers for GPU-based massively parallel processors (MPPs).

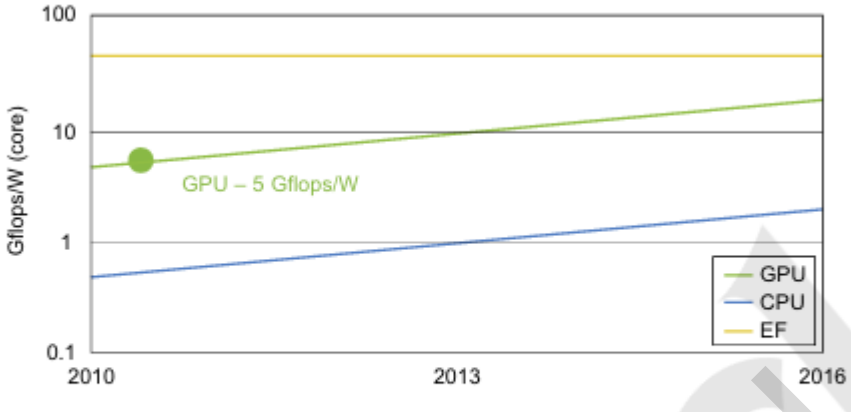


FIGURE 1.9

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.

Memory, Storage, and Wide-Area Networking

Memory Technology

• DRAM capacity growth: 16 KB (1976) → 64 GB (2011), increasing 4× every 3 years.

• Memory wall problem: CPU speed increases faster than memory access speed, creating a performance gap.

• Hard drive capacity growth: 260 MB (1981) → 250 GB (2004) → 3 TB (2011), increasing 10× every 8 years.

• Challenge: Faster CPUs and larger memory lead to CPU-memory bottlenecks.

Disks and Storage Technology

- Disk arrays exceeded 3 TB in capacity beyond 2011.
- Flash memory & SSDs are revolutionizing HPC (High-Performance Computing) and HTC (High-Throughput Computing).
- SSD lifespan: 300,000–1 million write cycles per block, making them durable for years.

Storage trends:

- Tape units → obsolete
- Disks → function as tape units
- Flash storage → replacing traditional disks
- Memory → functions as cache

Challenges: Power consumption, cooling, and cost of large storage systems.

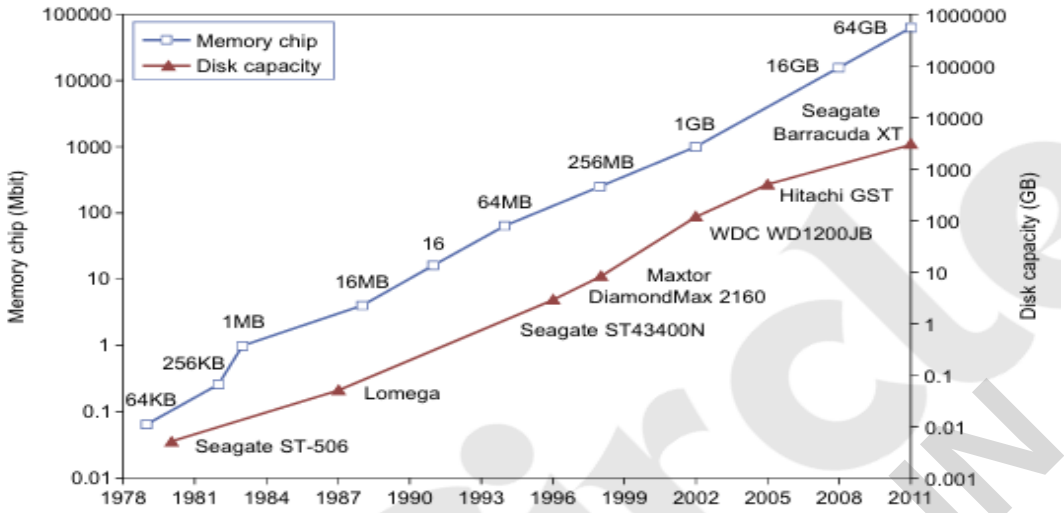


FIGURE 1.10

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

System-Area Interconnects

These three network types are frequently found in large clusters built with commercial components, showcasing flexibility in cluster design.

LAN (Local Area Network):

- Typically connects client hosts to large servers.
- Provides communication within a limited area, such as a cluster.

SAN (Storage Area Network):

- Links servers to network storage like disk arrays.
- Allows efficient and centralized data storage management.

NAS (Network Attached Storage):

- Directly connects client hosts to storage devices (disk arrays).
- Offers shared storage access to multiple clients.

Small Clusters:

- Often built using a multiport Gigabit Ethernet switch and copper cables.
- Suitable when large distributed storage is not needed.

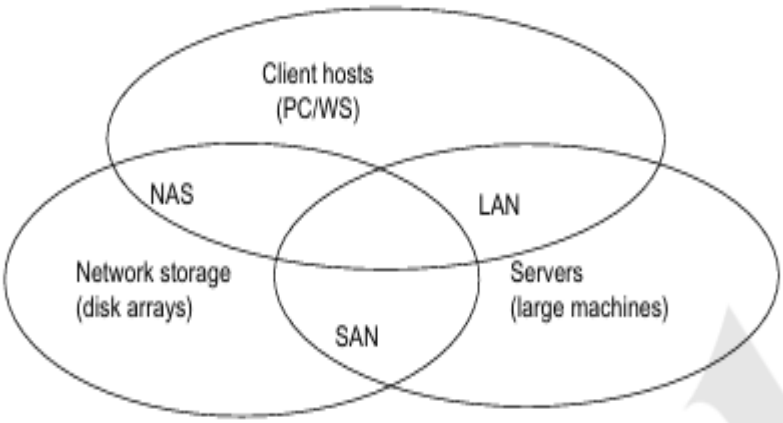


FIGURE 1.11

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.

Wide-Area Networking

Ethernet speed evolution:

- 10 Mbps (1979) → 1 Gbps (1999) → 40-100 Gbps (2011) → projected 1 Tbps (2013).
- Network performance grows 2× per year, surpassing Moore's Law for CPUs.
- High-bandwidth networking enables large-scale distributed computing.
- IDC 2010 report: InfiniBand & Ethernet will dominate HPC interconnects.
- Most data centers use Gigabit Ethernet for server clusters.

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.

Conventional Computers:

- Operate with a single OS image.
- Software is tightly linked to specific hardware, limiting flexibility.

Virtual Machines (VMs):

- Solve issues like resource underutilization and software inflexibility.
- Provide virtualized resources: processors, memory, and I/O.

Virtualization in Large Systems:

- Essential for clusters, grids, and clouds.
- Enables efficient, dynamic access to computing and storage.

Purpose of Virtualization:

- Aggregates resources for a unified system view.
- Improves manageability, flexibility, and efficiency.

Examples of Virtualized Resources:

- Virtual Machines (VMs).
- Virtual storage and networking.

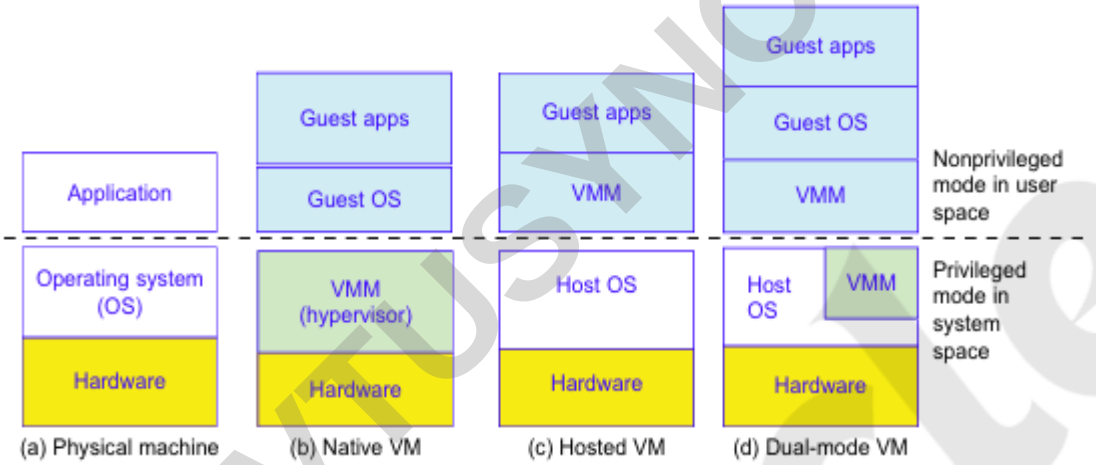


FIGURE 1.12

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)

(a) Physical Machine:

- The traditional setup where a single OS directly manages the hardware.
- No virtualization; applications and the OS depend entirely on the physical machine.

(b) Native VM Architecture:

- A virtual machine monitor (VMM or hypervisor) operates directly on the hardware.
- Efficient and high-performing, often used for managing resources in large-scale systems.

(c) Hosted VM Architecture:

- A VMM runs on top of a host operating system, treating VMs as applications.

- Simpler to set up but less efficient due to the extra layer introduced by the host OS.

(d) Dual-Mode VM Architecture:

- Combines features of both native and hosted architectures.
- Some tasks are handled directly by the VMM on hardware, while others pass through a host OS.

Virtual Machines

Host Machine and Hardware:

- The physical hardware serves as the base for virtualization.
- Example: An x86 desktop running Windows OS as the host.

Virtual Machine (VM):

- Built with virtual resources managed by a guest OS to run specific applications.
- Provides hardware independence, allowing VMs to be ported across platforms.

Middleware Layer - Virtual Machine Monitor (VMM):

- Positioned between the host and VM.
- Controls virtualization and resource management.

VM Architectures:

- **(b) Native (Bare-Metal) VM:** Hypervisor (e.g., XEN) operates directly on hardware in privileged mode. Guest OS could differ, like Linux on Windows hardware.
- **(c) Hosted VM:** VMM runs in nonprivileged mode on a host OS. No need to modify the host.
- **(d) Dual-Mode VM:** Splits VMM functions between user level and supervisor level. May require minor modifications to the host OS.

Key Benefits:

- Supports multiple VMs on the same hardware.
- Facilitates portability and flexibility with virtual appliances bundled with their dedicated OS and applications.

VM Primitive Operations

VM Operations :

- **Multiplexing:** VMs are distributed across multiple hardware machines for efficient resource use.

- **Suspension:** A VM can be paused and stored in stable storage for later use.
- **Provision (Resumption):** A suspended VM can be resumed on a new hardware platform.
- **Live Migration:** A VM can be moved from one hardware platform to another without disruption.

Benefits:

- **Hardware Independence:** VMs can run on any available hardware platform, regardless of the underlying OS or architecture.
- **Flexibility:** Enables seamless porting of distributed applications.
- **Enhanced Efficiency:** Consolidates multiple server functions on a single hardware platform, reducing server sprawl.
- **Improved Utilization:** VMware estimates server utilization increases from 5–15% to 60–80% with this approach.

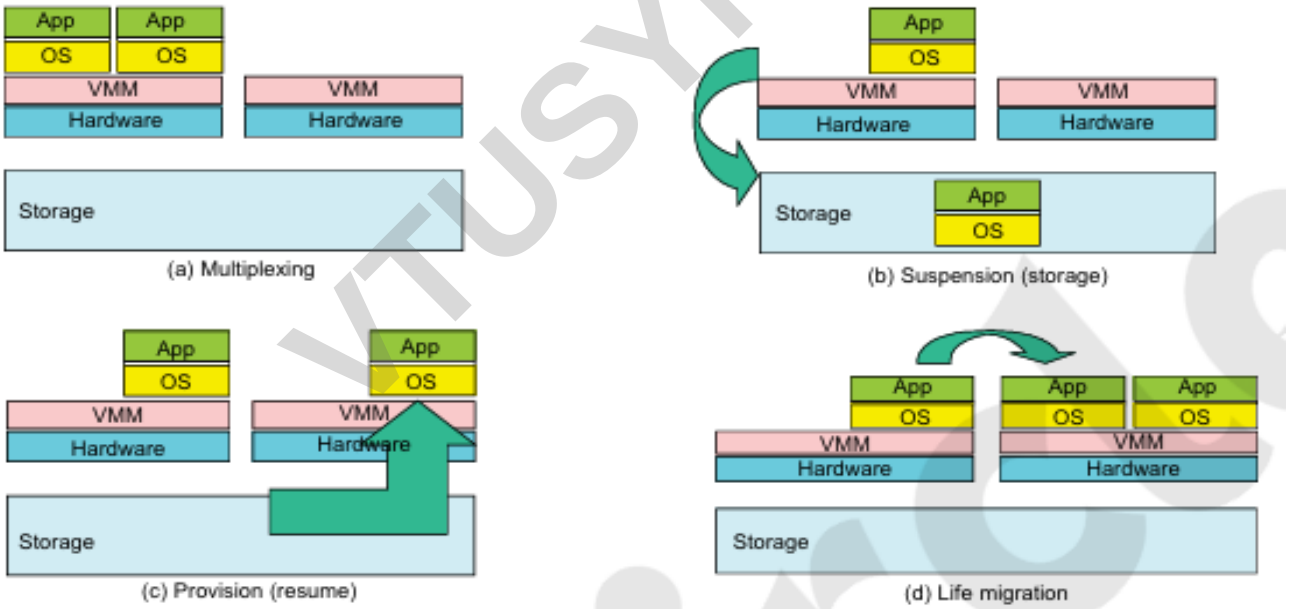


FIGURE 1.13
VM multiplexing, suspension, provision, and migration in a distributed computing environment.
(Courtesy of M. Rosenblum, Keynote address, ACM ASPLOS 2006 [41])

Virtual Infrastructures

Virtual Infrastructure:

- Connects physical resources (compute, storage, networking) to distributed applications using VMs.
- Separates hardware and software for dynamic resource mapping.

Benefits:

- Reduces costs, increases efficiency, and improves responsiveness.
- Example: Server consolidation and containment using virtualization.

Future Support:

- Virtualization is pivotal for clusters, clouds, and grids, as discussed in later chapters.

Data Center Trends

- Shows customer spending, server installation growth, and cost breakdown over the years.
- Highlights the "virtualization management gap," emphasizing the growing importance of virtualization technologies.

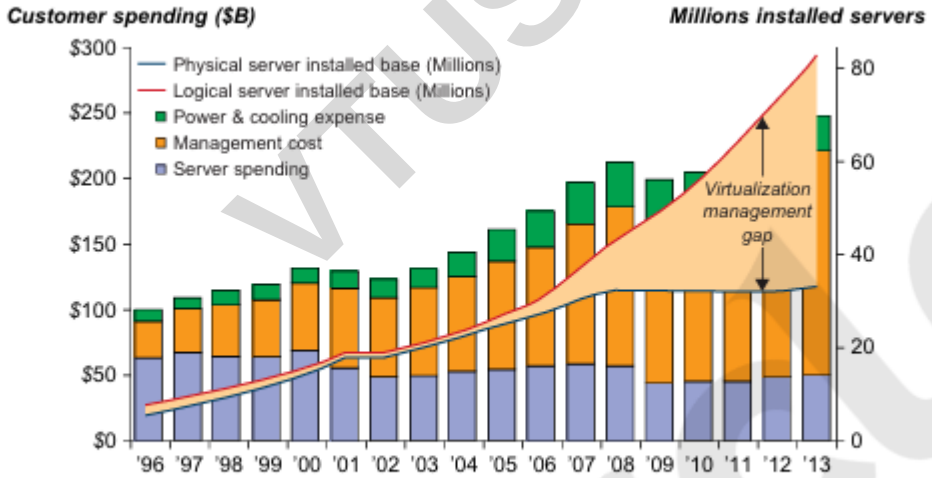


FIGURE 1.14
Growth and cost breakdown of data centers over the years.

Data Centre Virtualization for Cloud Computing

• **Cloud Architecture:** Uses commodity hardware (x86 processors, low-cost storage, Gigabit Ethernet).

• **Design Priorities**: Cost-efficiency over raw performance, focusing on storage and energy savings.

Data Centre Growth & Cost Breakdown

Large data centres contain thousands of servers.

Cost Distribution (2009 IDC Report):

- o 30% for IT equipment (servers, storage).
- o 60% for maintenance and management (cooling, power, etc.).
- o Electricity & cooling costs increased from 5% to 14% in 15 years.

Low-Cost Design Philosophy

Uses commodity x86 servers and Ethernet networks instead of expensive hardware.

Software manages network traffic, fault tolerance, and scalability.

Convergence of Technologies Enabling Cloud Computing

1. Hardware Virtualization & Multi-Core Chips: Allows dynamic configurations.
2. Utility & Grid Computing: Forms the foundation of cloud computing.
3. SOA, Web 2.0, and Mashups: Advances in web technologies drive cloud adoption.
4. Autonomic Computing & Data Center Automation: Enhances efficiency.

Impact of Cloud Computing on Data Science & E-Research

• Data Deluge: Massive data from sensors, web, simulations, requiring advanced data management.

• E-Science Applications: Used in biology, chemistry, physics, and social sciences.

• MapReduce & Iterative MapReduce: Enable parallel processing of big data.

• Multicore & GPU Clusters: Boost computational power for scientific research.

• Cloud Computing & Data Science Convergence: Revolutionizes computing architecture and programming models.

System Models for Distributed and Cloud Computing

Distributed and Cloud Computing Systems

- Built over a large number of autonomous computer nodes.
- Nodes are interconnected using SANs, LANs, or WANs in a hierarchical manner.

- Clusters of clusters can be created using WANs for large-scale systems.
- These systems are highly scalable, supporting web-scale connectivity.

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

Node Interconnection:

- Built over large numbers of autonomous computer nodes.
- Nodes are interconnected hierarchically via SANs, LANs, or WANs.
- LAN switches connect hundreds of machines; WANs link local clusters into massive systems.

Massive Systems:

- Highly scalable, supporting web-scale connectivity.
- Involve hundreds, thousands, or millions of computers, working collectively or collaboratively.

Classification (Table 1.2):

- **Clusters:** Homogeneous nodes for high-performance tasks like supercomputing (e.g., Google Search Engine, IBM Road Runner).
- **Peer-to-Peer (P2P) Networks:** Flexible client networks for file sharing, social networking (e.g., BitTorrent, Skype).
- **Grids:** Distributed resource pooling for global problem-solving (e.g., TeraGrid, ChinaGrid).

- **Cloud Platforms:** Virtualized servers for utility computing and dynamic resource provisioning (e.g., AWS, Microsoft Azure).

Applications and Trends:

- Clusters dominate supercomputing (e.g., 417 of Top 500 supercomputers in 2009).
- P2P faces resistance due to copyright issues.
- Clouds offer cost-effective and simple solutions, gaining popularity.

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.

Cluster Architecture

Cluster Architecture:

- Built around a low-latency, high-bandwidth interconnection network, such as SAN (e.g., Myrinet) or LAN (e.g., Ethernet).
- Hierarchical construction allows for scalable clusters using Gigabit Ethernet, Myrinet, or InfiniBand switches.

Interconnection and Internet Access:

- The cluster is connected to the Internet via a virtual private network (VPN) gateway.
- The gateway's IP address is used to locate the cluster.

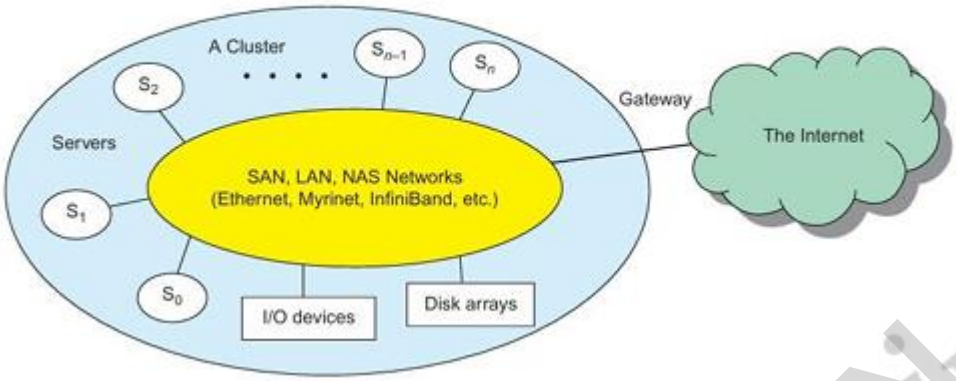
Node Management:

- Most clusters have loosely coupled node computers, each managed by its own operating system (OS).
- This results in multiple system images, as nodes operate autonomously under different OS controls.

Shared Resources:

- Shared I/O devices and disk arrays support the cluster.
- Despite being composed of multiple nodes, the cluster acts as a single computational unit connected to the Internet.

Cluster Architecture



Single-System Image

Greg Pfister highlights that an ideal cluster should integrate multiple system images into a **single-system image (SSI)**. This SSI creates an illusion, via software or hardware, where a collection of resources appears as one unified, powerful resource. SSI enables clusters to function as a single machine for users, supporting shared CPUs, memory, and I/O across cluster nodes. In contrast, a cluster with multiple system images consists of independent computers.

Hardware, Software, and Middleware Support

- Cluster nodes include PCs, workstations, servers, or SMP.
- MPI and PVM used for message passing.
- Most clusters run on Linux OS.
- Middleware is essential for SSI, high availability (HA), and distributed shared memory (DSM).
- Virtualization allows dynamic creation of virtual clusters.

Major Cluster Design Issues

- No unified cluster-wide OS for resource sharing.
- Middleware is required for cooperative computing and high performance.
- Benefits of clusters: Scalability, efficient message passing, fault tolerance, and job management

Grid Computing Infrastructures

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

Over the past 30 years, there has been significant progression in computing services:

- **Internet services** like Telnet allow local computers to connect to remote ones.
- **Web services** like HTTP enable remote access to web pages.
- **Grid computing** facilitates simultaneous interaction among applications running on distant computers.

Computational Grids

- **Grid Computing**: Functions like a power grid, combining computers, software/middleware, instruments, people, and sensors into an integrated infrastructure across LAN, WAN, or the Internet at various scales.
- Grids include workstations, servers, clusters, and supercomputers, while users can access them via PCs, laptops, or PDAs.
- Provides shared resources across resource sites owned by different organizations, enabling collaborative computational tasks.
- Built over IP broadband networks (LAN/WAN) and presented as a unified resource pool.
- Uses instruments like radio telescopes (e.g., SETI@Home for alien life searches) and supports scientific research (e.g., pulsar studies).
- Enterprises and consumers shape usage trends, offering computing, communication, content, and transaction services.
- **Global Examples**: National and international grids include TeraGrid (US), EGEE (Europe), and ChinaGrid, supporting scientific applications.

Grid Families

- **Grid Technology Evolution**: Requires new distributed computing models, software/middleware, network protocols, and hardware.
- **Industry Involvement**: National grid projects have led to platforms developed by companies like IBM, Microsoft, Sun, HP, Dell, and others.
- **Emerging Grid Services**: Grid service providers (GSPs) and applications have grown quickly, paralleling the rise of Internet and web services.
- **Grid System Categories**:
  - **Computational/Data Grids**: Operate primarily at a national level.
  - **P2P Grids**: Focus on peer-to-peer resource sharing.

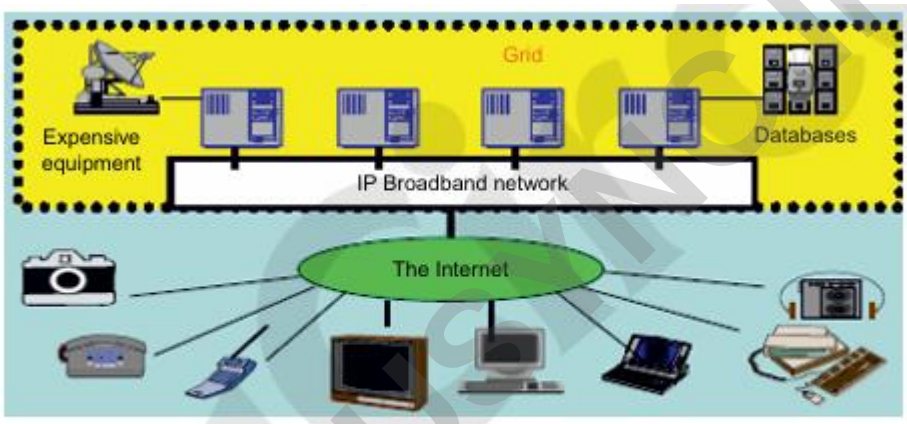


FIGURE 1.16
Computational grid or data grid providing computing utility, data, and information services through resource sharing and cooperation among participating organizations.

Table 1.4 Two Grid Computing Infrastructures and Representative Systems

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

Peer-to-Peer (P2P) Network Families

- Client-server architecture connects clients (PCs, workstations) to a central server for various applications.
- P2P (peer-to-peer) architecture is a distributed, client-oriented network model.
- P2P systems lack a central server and rely on peers to act as both clients and servers.
- P2P systems are described at the physical level and overlay networks at the logical level

P2P Systems

- In a **P2P system**, each node acts as both a client and a server, sharing system resources.
- Peer machines are simply client computers connected via the Internet, with the autonomy to join or leave freely.
- No master-slave relationship, central coordination, or global database exists.
- The system is **self-organizing** with distributed control.
- P2P networks form **ad hoc physical networks** across Internet domains, using protocols like TCP/IP and NAI.
- These networks dynamically change in size and topology as peers join or leave voluntarily.

Overlay Networks

- Data and files are distributed among participating peers.
- Peer IDs create an overlay network, defining logical connections between peers.
- The overlay network uses virtual links mapped to physical machines via IDs.
- Peer IDs are added or removed from the overlay network when peers join or leave.

Two types of overlay networks:

1. **Unstructured**: Random graph, uses flooding for queries, causing high traffic and unpredictable searches.
2. **Structured**: Follows specific topology and rules, with routing mechanisms for better efficiency.

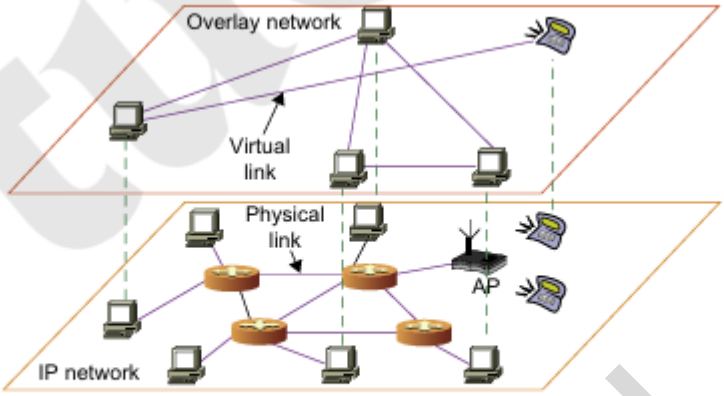


FIGURE 1.17

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

P2P Application Families

- **File Sharing**: Distributed file sharing of digital content (e.g., Gnutella, Napster, BitTorrent).
- **Collaboration**: Tools for chatting, instant messaging, and collaborative design (e.g., MSN, Skype).
- **Distributed Computing**: Specific applications for computing power, such as SETI@home with 25 Tflops over 3 million Internet hosts.
- **Application Platforms**: Systems like JXTA, .NET, and FightingAID@home offer support for naming, discovery, security, communication, and resource aggregation.

P2P Computing Challenges

- P2P systems face heterogeneity issues in hardware, software, and network requirements.
- Data locality, network proximity, and interoperability are key design objectives in P2P applications.
- Routing efficiency, fault tolerance, failure management, and load balancing significantly impact performance.
- Trust issues exist due to security, privacy, and copyright concerns among peers.
- P2P networks improve robustness by avoiding a single point of failure and replicating data across nodes.
- Lack of centralization makes management complex.
- Security risks arise as any client can potentially cause damage or abuse.
- P2P networks are suitable for low-security applications without sensitive data.

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.

Cloud Computing over the Internet

- Computational science is shifting to a data-intensive approach.
- Supercomputers need balanced systems, combining CPUs with petascale I/O and networking arrays.
- Future data handling will involve sending computations to data rather than moving data to workstations.
- IT trends are moving computing and data from desktops to large data centers.
- Cloud computing emerged to handle the data explosion, offering on-demand software, hardware, and data as a service.
- IBM defines a cloud as a pool of virtualized resources hosting diverse workloads, including backend jobs and interactive applications.
- Clouds allow rapid provisioning of virtual or physical machines for scaling workloads.
- They support redundant, self-recovering, and highly scalable models to handle hardware/software failures.
- Real-time resource monitoring in clouds enables rebalancing of allocations when necessary.

Internet Clouds

- Cloud computing uses virtualized platforms with elastic, on-demand resources.
- It dynamically provisions hardware, software, and data sets.
- The goal is to shift desktop computing to a service-oriented platform using server clusters and large databases.
- Cloud computing is cost-effective and simple, benefiting users and providers.
- Machine virtualization enables the cost-efficiency of cloud computing.
- The cloud is designed to support many applications simultaneously.
- A secure, trustworthy, and dependable ecosystem is essential for cloud computing.

- Some users view the cloud as a centralized resource pool; others see it as a distributed server cluster.

The Cloud Landscape

Traditional Distributed Computing Systems:

- Owned by autonomous domains (e.g., labs or companies) for on-premises needs.
- Performance bottlenecks: system maintenance, poor utilization, and high upgrade costs.

Cloud Computing:

- An on-demand computing paradigm addressing traditional system challenges.

Cloud Service Models:

1. **Infrastructure as a Service (IaaS)**: Provides servers, storage, networks, and data center resources. Users deploy and run VMs but don't manage the cloud infrastructure.
2. **Platform as a Service (PaaS)**: Enables deployment of user-built applications on a cloud platform. Includes middleware, databases, tools, and APIs.
3. **Software as a Service (SaaS)**: Browser-based software for business processes (CRM, ERP, HR, etc.), eliminating upfront costs for users and reducing hosting costs for providers.

Cloud Deployment Modes:

- Private, public, managed, and hybrid, each with differing security implications.

Cloud Advantages:

- Protected locations and energy efficiency.
- Improved peak-load capacity utilization.
- Separation of infrastructure maintenance and application development.
- Reduced costs compared to traditional systems.
- Support for application development.
- Efficient data discovery and service distribution.
- Addressing privacy, security, and reliability issues.
- Flexible service agreements and pricing models.

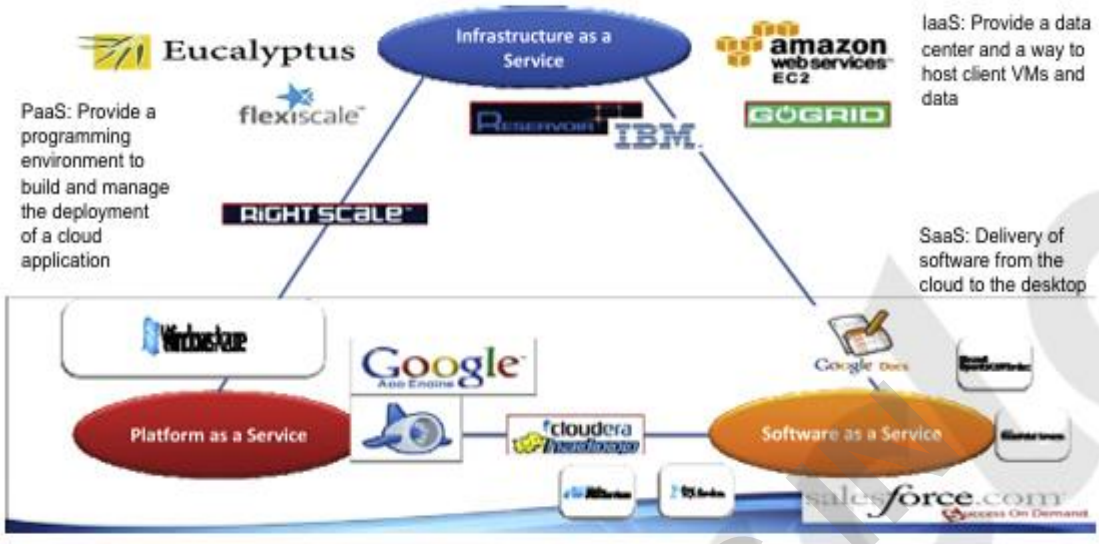


FIGURE 1.19
Three cloud service models in a cloud landscape of major providers.

SOFTWARE ENVIRONMENTS FOR DISTRIBUTED SYSTEMS AND CLOUDS

Service-Oriented Architecture (SOA)

- ✓ In grids/web services, Java, and CORBA:
- ✓ An entity is a **service**, a **Java object**, or a **CORBA distributed object**, respectively.
- ✓ These architectures build on the traditional seven **Open Systems Interconnection (OSI) layers** for networking abstractions.

Base Software Environment:

- ✓ .NET or Apache Axis for web services.
- ✓ Java Virtual Machine (JVM) for Java.
- ✓ Broker networks for CORBA.
- ✓ On top of the base software environment, a **higher-level environment** is built, tailored to distributed computing features.
- ✓ This includes **entity interfaces** and **inter-entity communication** that reconstruct the top four OSI layers, but at the **entity level** rather than the bit level.

Layered Architecture for Web Services and Grids

- Entity interfaces include WSDL, Java methods, and CORBA IDL for distributed systems.
- High-level communication systems use SOAP, RMI, and IIOP for RPC, fault recovery, and routing.
- Middleware like WebSphere MQ or JMS supports routing and virtualization.

- Fault tolerance is achieved via WSRM, adapting OSI layer features for entity abstractions.
- Security is implemented using OSI concepts like IPsec and secure sockets.
- Higher-level services provide registries, metadata, and entity management.
- Discovery tools like JNDI, UDDI, LDAP, and ebXML enable service discovery.
- Management services include CORBA Life Cycle, Enterprise JavaBeans models, and Jini's lifetime model.
- Shared memory models simplify information exchange.
- Distributed systems improve performance using multiple CPUs and support software reuse.
- Older approaches like CORBA are being replaced by SOAP, XML, and REST in modern systems.

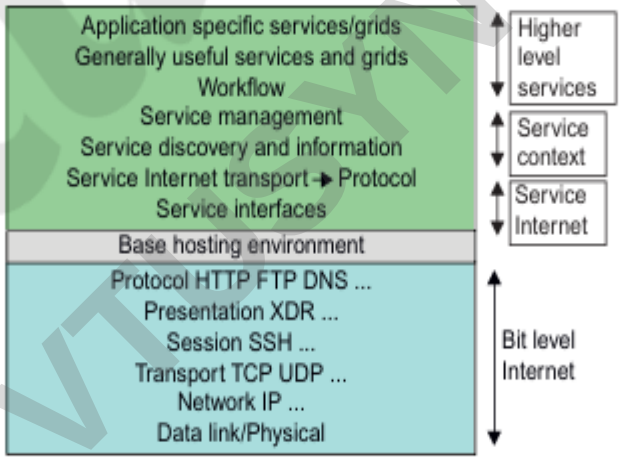


FIGURE 1.20

Layered achitecture for web services and the grids.

Web Services and Tools

- Loose coupling and heterogeneous implementations make services more attractive than distributed objects.
- Service architectures include **web services** and **REST systems**, each with distinct approaches to interoperability.

Web Services:

- Fully specify the service and environment using **SOAP** messages.
- SOAP acts as a universal distributed operating system, but efficiency and agreement on protocols remain challenging.

 REST Systems:

- Prioritize simplicity and delegate complex problems to application-specific software.
- Minimal information is included in the header; the message body carries required data.
- REST is suitable for rapid technology environments and uses "XML over HTTP" instead of SOAP.

 Distributed entities in **CORBA** and **Java** are linked via RPCs.

- In Java, method calls are replaced by **Remote Method Invocation (RMI)**.
- CORBA uses C++-style syntax for linking object interfaces.

 Grids and Clouds:

- Grids refer to either single services or collections of services.
- Sensors represent entities producing data, while grids and clouds handle multiple message-based inputs and outputs.

The Evolution of SOA

- Service-Oriented Architecture (SOA) has evolved to support grids, clouds, interclouds, and systems of systems.
- Sensors (e.g., ZigBee, Bluetooth, WiFi devices) collect raw data as **sensor services (SS)**.
- Sensor services interact with grids, databases, and various clouds, such as compute, storage, filter, and discovery clouds.
- **Filter services (fs)** process raw data, removing unwanted information to address specific requests.
- A collection of filter services forms a **filter cloud**.
- SOA processes raw data into information, knowledge, and intelligence for decision-making.
- Distributed systems use a **web interface/portal** to transform sensor data through compute, storage, filter, and discovery clouds.
- Example portals include **OGFCE** and **HUBzero**, which employ web service and Web 2.0 technologies.

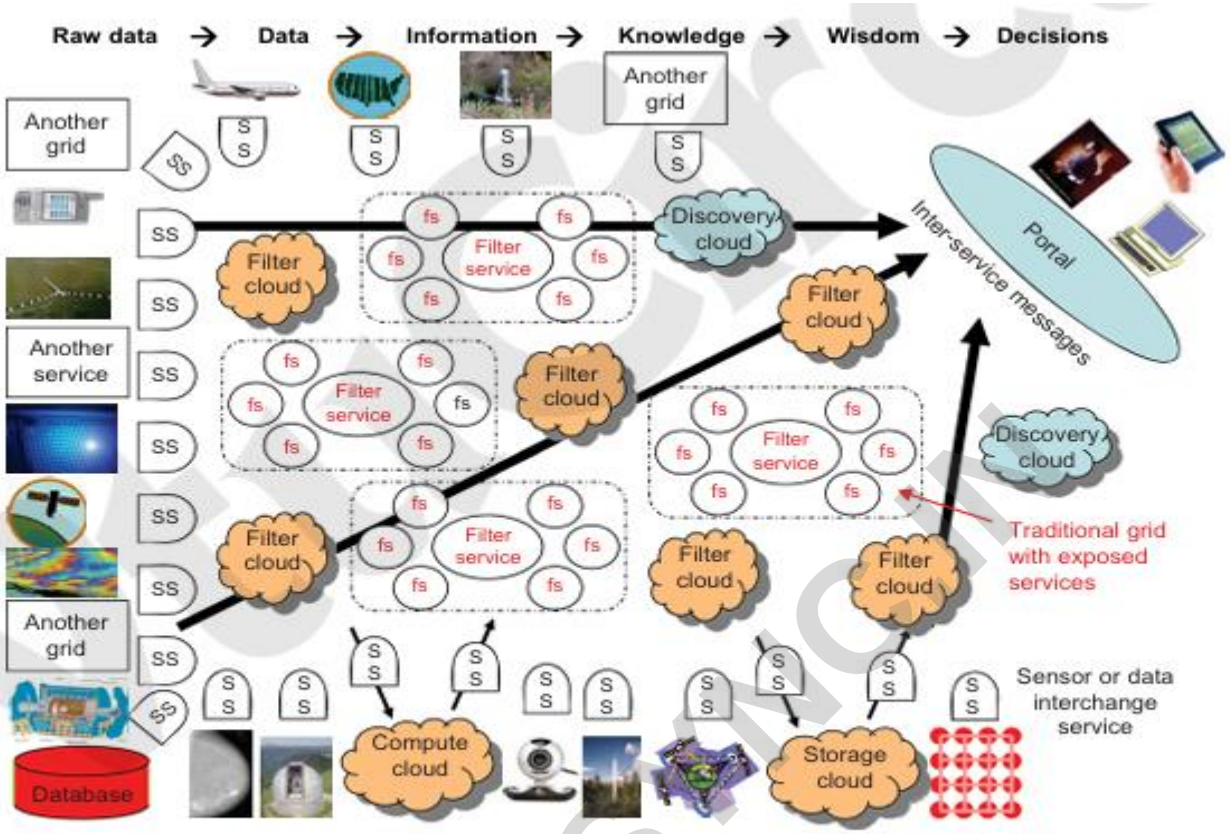


FIGURE 1.21
The evolution of SOA: grids of clouds and grids, where "SS" refers to a sensor service and "fs" to a filter or transforming service.

Grids versus Clouds

- Service-Oriented Architecture (SOA) has evolved to support grids, clouds, grids of clouds, clouds of grids, and interclouds (clouds of clouds).
- **Sensor services (SS)** collect raw data; sensors include ZigBee devices, Bluetooth devices, WiFi access points, personal computers, and wireless phones.
- SS devices interact with grids, databases, and various clouds, such as compute, storage, filter, and discovery clouds.
- **Filter services (fs)** process raw data by eliminating unwanted information, forming **filter clouds** to respond to specific requests.
- SOA transforms massive raw data into useful information, knowledge, and wisdom for intelligent decision-making.
- Distributed systems often require web interfaces or portals to process and present transformed data.

- Data streams from sensors pass through sequences of compute, storage, filter, and discovery clouds before converging at portals accessed by users.
- Example portals include **OGFCE** and **HUBzero**, which use web service (portlet) and Web 2.0 (gadget) technologies.

Grids versus Clouds

- The boundary between grids and clouds is increasingly blurred.
- Workflow technologies in web services coordinate and orchestrate services, supporting business models like two-phase transactions.
- Workflow standards and approaches include BPEL, Pegasus, Taverna, Kepler, Trident, and Swift.
- Grids use **static resources**, while clouds focus on **elastic resources**.
- Key difference: **Dynamic resource allocation** via virtualization and autonomic computing.
- Grids can be built from multiple clouds, offering better support for **negotiated resource allocation**.
- These combinations result in systems like **clouds of clouds**, **grids of clouds**, **clouds of grids**, or **inter-clouds** in SOA architectures.

Trends toward Distributed Operating Systems

- Distributed systems are **loosely coupled**, with multiple system images due to independent operating systems on node machines.
- A **distributed OS** promotes resource sharing and efficient communication among nodes.
- Such a system is typically a **closed system**, relying on **message passing** and **Remote Procedure Calls (RPCs)** for internode communication.
- A distributed OS is essential for improving the **performance, efficiency, and flexibility** of distributed applications.

Distributed Operating Systems

- **Network OS**: Built over heterogeneous OS platforms, offers the lowest transparency, and relies on file sharing for communication.
- **Middleware**: Provides limited resource sharing, similar to MOSIX/OS for clustered systems.
- **Truly Distributed OS**: Achieves high resource usage and system transparency.

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

Amoeba versus DCE

- **DCE** is a middleware-based system for distributed computing environments.
- **Amoeba** was developed academically at Free University in the Netherlands.
- **OSF** promoted DCE for distributed computing.
- Amoeba, DCE, and MOSIX2 remain research prototypes, mainly used in academia.
- No successful commercial OS products emerged from these research systems.
- New web-based operating systems are needed to support resource virtualization in distributed environments.
- These distributed OSs should distribute functionalities across available servers.

- Conventional OSs run on centralized platforms, while distributed OSs require a lightweight **microkernel approach** (e.g., Amoeba).
- Alternatively, distributed OSs could extend existing systems like DCE by modifying **UNIX**.
- The trend is to relieve users of most resource management responsibilities.

MOSIX2 for Linux Clusters

- **MOSIX2** is a distributed OS that operates with a virtualization layer in Linux.
- It provides a **partial single-system image** to user applications.
- Supports **sequential** and **parallel applications**, and migrates software processes among Linux nodes.
- Manages **Linux clusters** or grids of multiple clusters with resource discovery and process migration.
- Allows **flexible grid management**, enabling resource sharing among cluster owners.
- MOSIX-enabled grids can scale indefinitely, provided there is **trust among cluster owners**.
- It is being applied to manage resources in **Linux clusters**, **GPU clusters**, **grids**, and **clouds** (when VMs are used).

Transparency in Programming Environments

- Transparent computing infrastructure separates **user data**, **applications**, **OS**, and **hardware** into four distinct levels.
- **User data** is owned independently of applications.
- The OS provides **clear interfaces** and standard system calls to application programmers.
- Future cloud infrastructure will decouple hardware from the OS through **standard interfaces**.
- Users can choose different **OSes** to run on their preferred hardware devices.
- Cloud applications as **SaaS** enable users to switch between services without binding data to specific applications.

Parallel and Distributed Programming Models

- Four programming models for distributed computing aim for **scalable performance** and **application flexibility**.

    - **MPI (Message Passing Interface)**:
- A library for C or FORTRAN to create parallel programs on distributed systems.
- Supports synchronous/asynchronous communication and I/O operations in user programs.

MapReduce:

- A web programming model for scalable data processing on large clusters.
- Includes Map and Reduce functions for processing large data sets.

Hadoop:

- A software library for running large applications on vast data sets.
- Scalable, efficient, and reliable, often used in business and cloud computing.
- Service clouds rely on extending tools like **Hadoop**, **EC2**, and **S3** for distributed computing over storage systems.

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

Message-Passing Interface (MPI)

- **MPI (Message Passing Interface)**: A primary standard for developing parallel and concurrent programs on distributed systems.
- Provides a library of subprograms callable from **C** or **FORTRAN** for writing parallel programs.
- Designed to support clusters, grid systems, and P2P systems with enhanced web services and utility computing applications.
- **PVM (Parallel Virtual Machine)**: Another low-level primitive for distributed programming.
- Both MPI and PVM are discussed in detail by Hwang and Xu [28].

Hadoop Library

- Hadoop, originally developed by a Yahoo! group, is a software platform for writing and running applications over vast distributed data.
- It can **scale easily** to store and process petabytes of web data.
- Hadoop is **economical**, offering an open-source version of MapReduce that reduces overhead.
- It is **efficient**, achieving high parallelism across many commodity nodes.
- It is **reliable**, automatically maintaining multiple data copies for redeployment after unexpected system failures.

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

Open Grid Services Architecture (OGSA)

- Grid infrastructure development is motivated by **large-scale distributed computing applications** requiring extensive resource and data sharing.
- **OGSA (Open Grid Services Architecture)** is a common standard for public grid services.
- **Genesis II** is an implementation of OGSA.
- Key features of Genesis II include:
  - A **distributed execution environment**.
  - **Public Key Infrastructure (PKI)** services using a local certificate authority (CA).
  - **Trust management** and security policies in grid computing.

Globus Toolkits and Extensions

- **Globus** is a middleware library developed by the **U.S. Argonne National Laboratory** and **USC Information Science Institute**.
- Implements **OGSA standards** for resource discovery, allocation, and security in grid environments.
- Supports **multisite mutual authentication** using PKI certificates.
- The current version, **GT 4**, has been in use since 2008.
- IBM has extended Globus for **business applications**.

PERFORMANCE, SECURITY, AND ENERGY EFFICIENCY

Performance Metrics and Scalability Analysis

- **Performance metrics** are essential for assessing distributed systems.
- This section focuses on various dimensions of **scalability** and **performance laws**.

- System scalability will be evaluated in relation to **OS images** and the **limiting factors** affecting performance.

Performance Metrics

- **Processor and network performance** are estimated using metrics like CPU speed in MIPS and network bandwidth in Mbps.
- **System throughput** is measured in MIPS, Tflops, or TPS.
- Additional performance measures include **job response time** and **network latency**.
- A preferred interconnection network has **low latency** and **high bandwidth**.
- **System overhead** factors include OS boot time, compile time, I/O data rate, and runtime support system.
- Other performance metrics involve **QoS for Internet/web services**, system **availability** and **dependability**, and **security resilience** against network attacks.

Dimensions of Scalability

- Scalable performance requires backward compatibility with existing hardware and software.
- Overdesign can be cost-ineffective, with scaling influenced by practical factors.
- Size scalability involves adding processors, cache, memory, storage, or I/O channels to boost performance or functionality.
- Software scalability includes upgrading OS, compilers, libraries, and application software while ensuring compatibility with larger systems.
- Application scalability aligns problem size with machine size for efficiency and cost-effectiveness, often by enlarging problem size.
- Technology scalability adapts to new component and networking technologies, considering time (generation updates), space (packaging and energy), and heterogeneity (compatibility across diverse hardware or software).

Scalability versus OS Image Count

- Scalable performance improves speed by adding processors, servers, memory, disk capacity, or I/O channels.
- OS image refers to independent OS instances in clusters, grids, P2P networks, or clouds.
- SMP (Symmetric Multiprocessor) has a single system image, limited to a few hundred processors by 2010, with scalability constrained by packaging and interconnects.
- NUMA (Nonuniform Memory Access) uses SMP nodes with shared memory, scaling to thousands of processors, such as a 2,048-processor NUMA machine with 32 OS images.
- Clusters consist of loosely coupled SMP servers or high-end machines, offering higher scalability than NUMA. Total processors or cores exceed the number of OS images by orders of magnitude.

- Clouds, being virtualized clusters, scaled up to a few thousand VMs by 2010.
- Grids include clusters, mainframes, supercomputers, or MPP systems, with fewer OS images than processors.
- P2P networks can scale to millions of peer nodes, with performance relying on public network QoS.
- Comparisons of scalability should consider similar networking levels across P2P, clouds, and clusters.

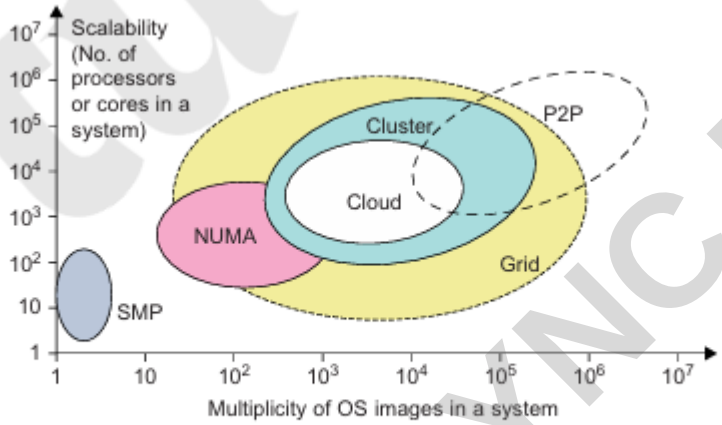


FIGURE 1.23
System scalability versus multiplicity of OS images based on 2010 technology.

Amdahl's Law

Amdahl's Law explains how much faster a program can run when split across multiple processors.

- Some part of the program (called the **sequential bottleneck**) must always run on a single processor, and it can't be parallelized. This is represented by α\alphaα.
- The rest of the program can run in parallel across nnn processors. This is

  (1−α)(1 - \alpha)(1−α).

The total time for the program on nnn processors is:

Tparallel=αT+(1−α)TnT_{\text{parallel}} = \alpha T + \frac{(1 - \alpha) T}{n}Tparallel =αT+n(1−α)T

Where TTT is the original time on a single processor.

Speedup (how much faster the program runs) is:

S=TαT+(1−α)Tn=1α+(1−α)nS = \frac{T}{\alpha T + \frac{(1 - \alpha) T}{n}} = \frac{1}{\alpha + \frac{(1 - \alpha)}{n}}S=αT+n(1−α)TT=α+n(1−α)1

As you add more processors, the speedup increases, but there's a limit. The maximum speedup happens when α\alphaα (the part that can't be parallelized) is zero.

For example, if 25% of the code can't be parallelized (α=0.25\alpha = 0.25α=0.25), the best speedup you'll ever get is 4, even with hundreds of processors.

Problem with Fixed Workload

In Amdahl's Law, the assumption is made that the workload remains the same for both sequential and parallel execution, with a fixed problem size or data set. This is known as **fixed-workload speedup**.

The **system efficiency** EEE is defined as the speedup SSS divided by the number of processors nnn, as follows:

E=Sn=1αn+(1−α)E = \frac{S}{n} = \frac{1}{\alpha n + (1 - \alpha)}E=nS=αn+(1−α)1

In many cases, especially with large clusters, the system efficiency becomes quite low. For instance, when executing the program on 256 processors, the efficiency is:

E=1(0.25×256)+0.75≈1.5%E = \frac{1}{(0.25 \times 256) + 0.75} \approx 1.5\%E=(0.25×256)+0.751≈1.5%

This low efficiency occurs because only a few processors (e.g., 4) are actively used, while the majority of the processors remain idle. As the cluster size increases, the parallel processing becomes less efficient, mainly due to the sequential bottleneck (the part of the code that can't be parallelized).

Gustafson's Law

To improve efficiency when using large clusters, **scaled-workload speedup** is used, which scales the problem size to match the cluster's capabilities. This approach, proposed by John Gustafson (1988), adjusts the workload based on the number of processors.

Scaled-Workload Speedup:

- Let WWW be the original workload. When using nnn processors, the workload is scaled to W′=αW+(1−α)nWW' = \alpha W + (1 - \alpha) n WW′=αW+(1−α)nW, where only the parallelizable portion of the workload is scaled by nnn.

- The total execution time for the scaled workload W′W'W′ on nnn processors gives the **scaled-workload speedup** S′S'S′:

S′=W′W=αW+(1−α)nWW=α+(1−α)nS' = \frac{W'}{W} = \frac{\alpha W + (1 - \alpha) n W}{W} = \alpha + (1 - \alpha) nS′=WW′=WαW+(1−α)nW=α+(1−α)n

This is known as **Gustafson's Law**.

Efficiency with Scaled Workload:

The efficiency E′E'E′ for the scaled workload on nnn processors is:

E′=S′n=αn+(1−α)E' = \frac{S'}{n} = \frac{\alpha}{n} + (1 - \alpha)E′=nS′=nα+(1−α)

For example, if the workload is scaled for a 256-node cluster with α=0.25\alpha = 0.25α=0.25, the efficiency is:

E′=0.25256+0.75≈0.751E' = \frac{0.25}{256} + 0.75 \approx 0.751E′=2560.25+0.75≈0.751

When to Apply Amdahl's vs. Gustafson's Law:

- **Amdahl's Law** should be used for **fixed workloads**, where the problem size doesn't change.
- **Gustafson's Law** should be used when the **workload is scaled** according to the number of processors, as it leads to better efficiency for large clusters.