Module-01

Distributed System Models and Enabling Technologies

Scalable Computing Over the Internet 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.

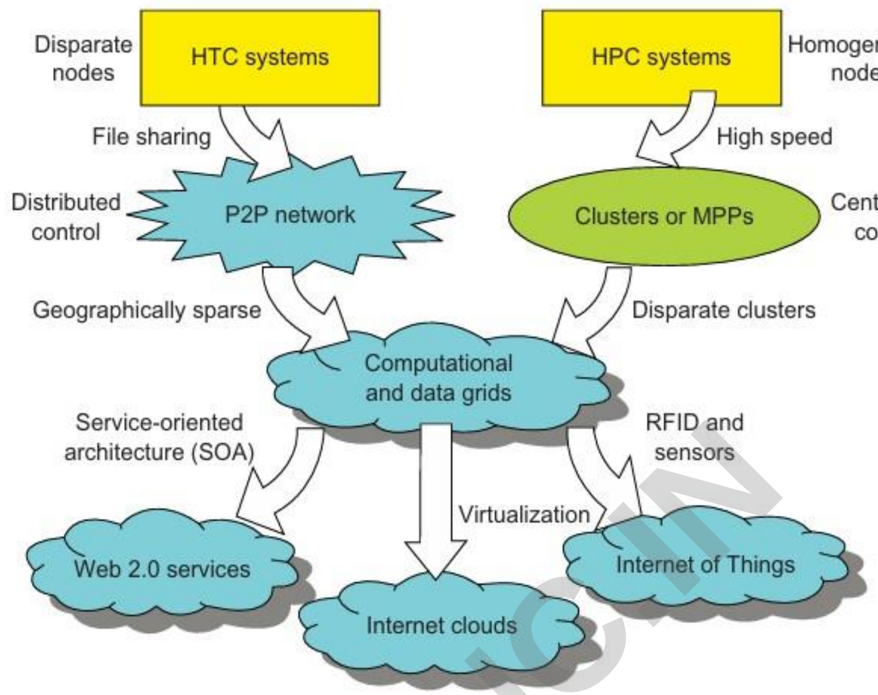


FIGURE 1.1

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

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

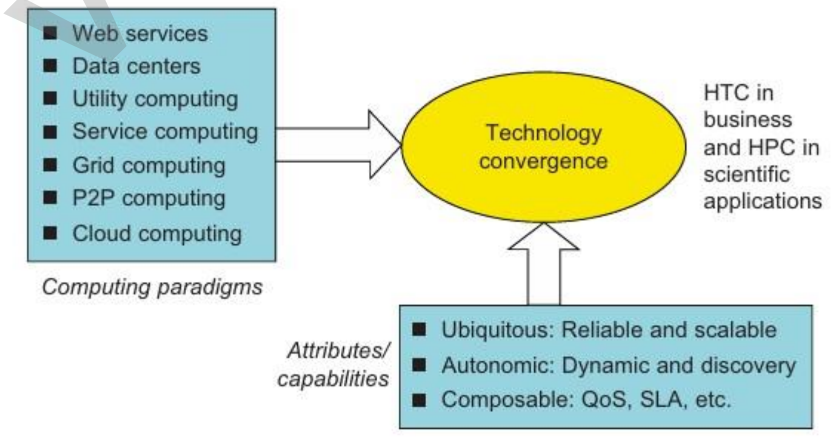


FIGURE 1.2

The vision of computer utilities in modern distributed computing systems.

- **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**:
  - **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**.

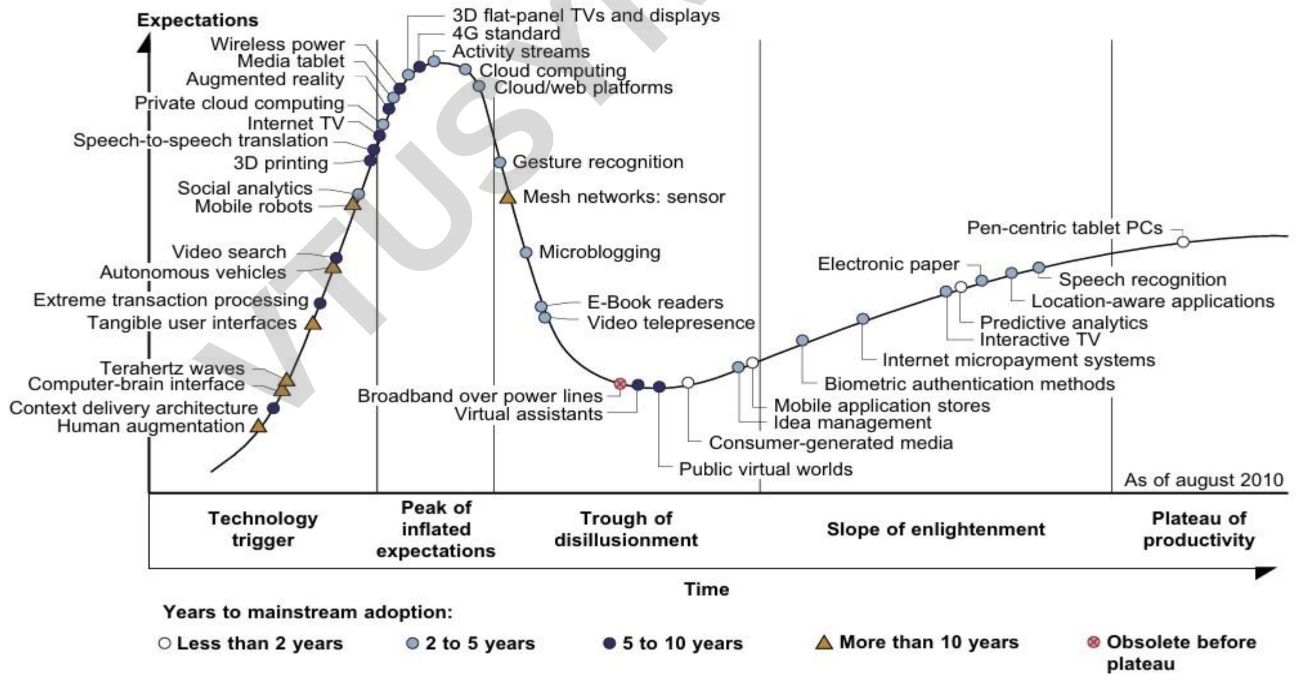
Communication Models:

- **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.



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).

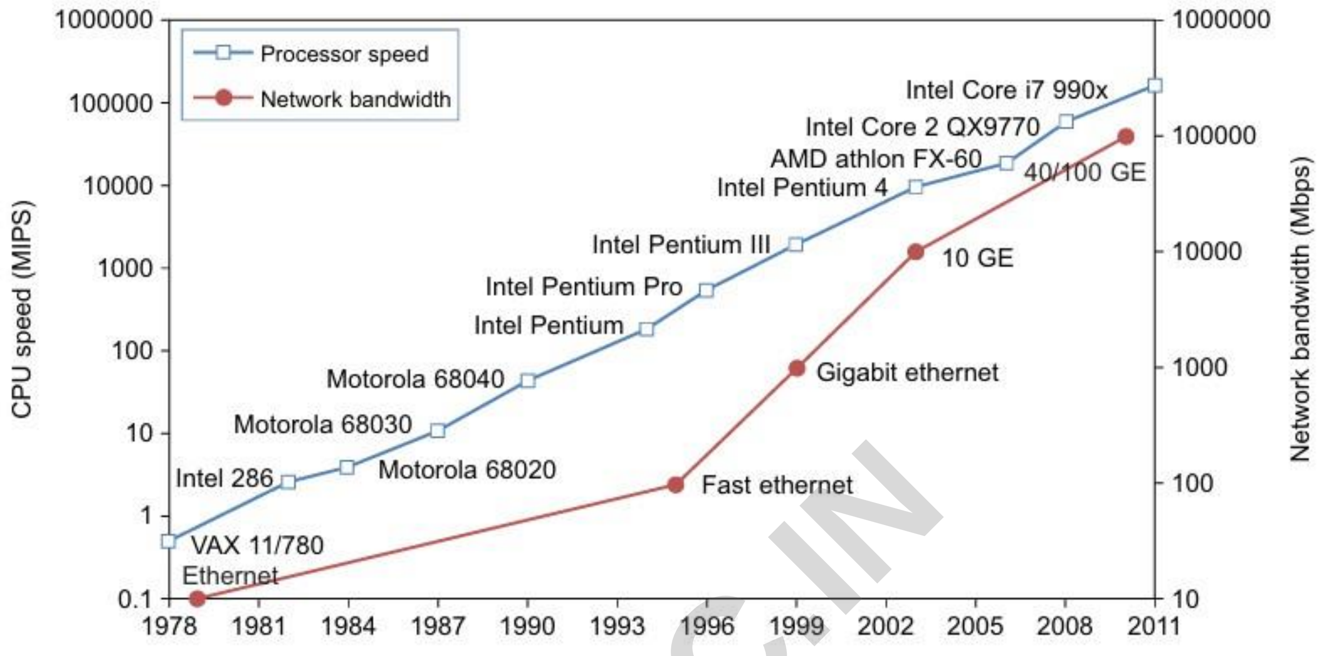


FIGURE 1.4

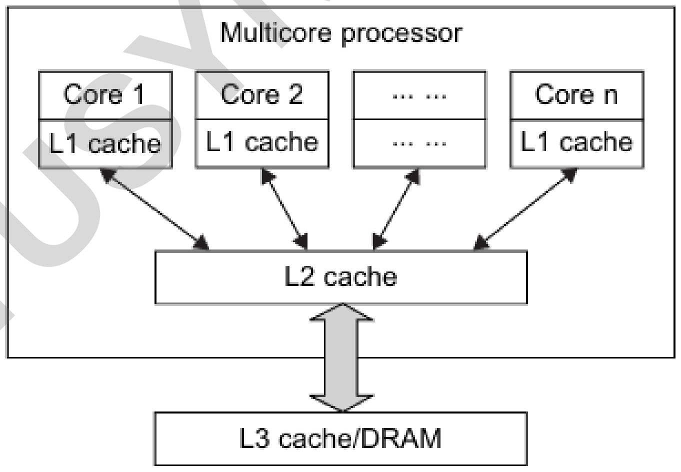


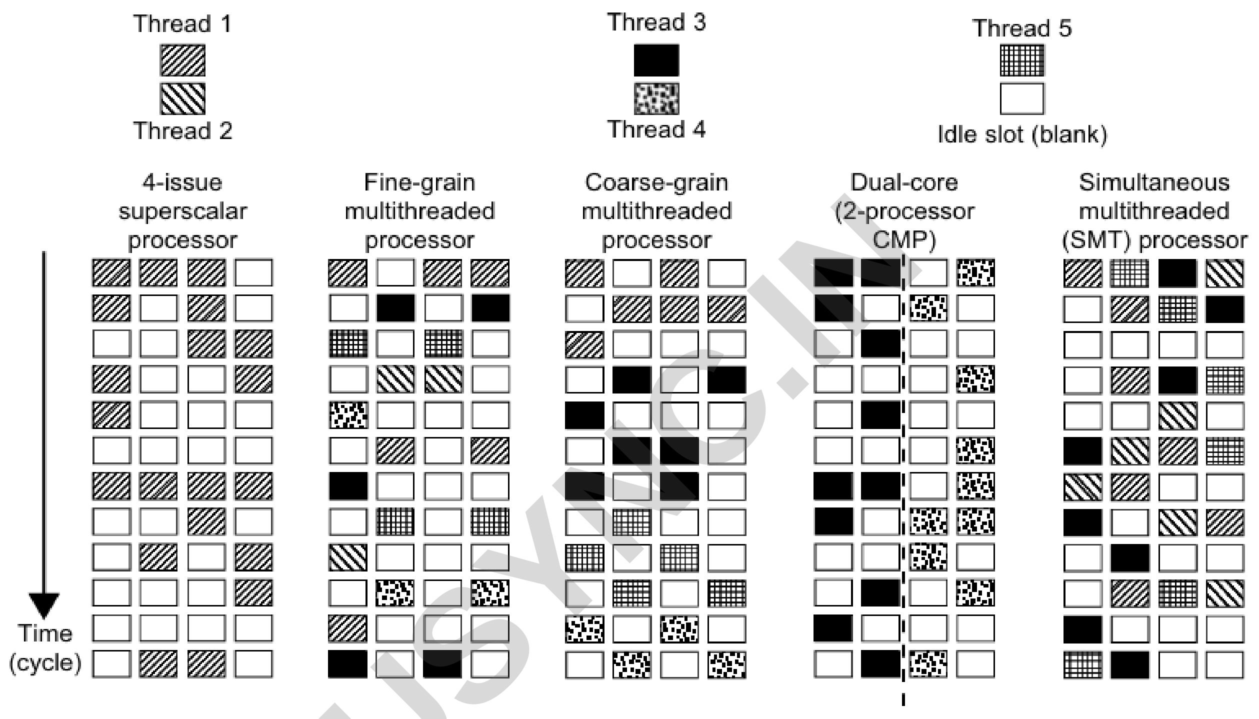
FIGURE 1.5

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



Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Idle slot (blank)
4-issue superscalar processor
Fine-grain multithreaded processor
Coarse-grain multithreaded processor
Dual-core (2-processor CMP)
Simultaneous multithreaded (SMT) processor
Time (cycle)

FIGURE 1.6

Types of processor architectures:

- **Superscalar** – Single-threaded with multiple functional units.
- **Fine-grain multithreading** – Switches between threads per cycle.
- **Coarse-grain multithreading** – Executes multiple instructions per thread before switching.
- **Chip Multiprocessor (CMP)** – Multicore processor executing multiple threads.
- **Simultaneous Multithreading (SMT)** – Executes instructions from different threads in parallel.

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

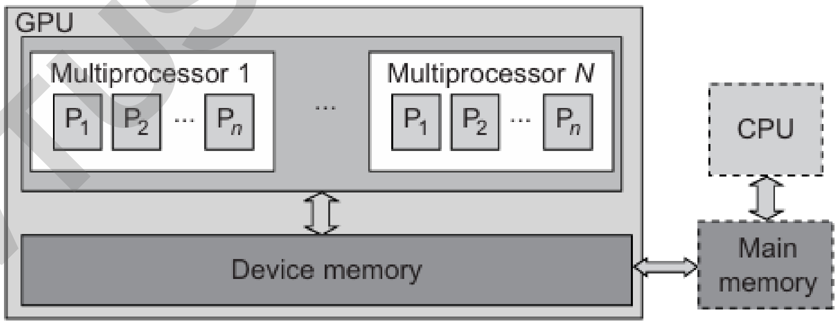


FIGURE 1.7

- **CPU offloads floating-point computations** to **GPU**.
- **CUDA programming (by NVIDIA)** enables large-scale parallel computing.

Example 1.1 the NVIDIA Fermi GPU Chip with 512 CUDA Cores

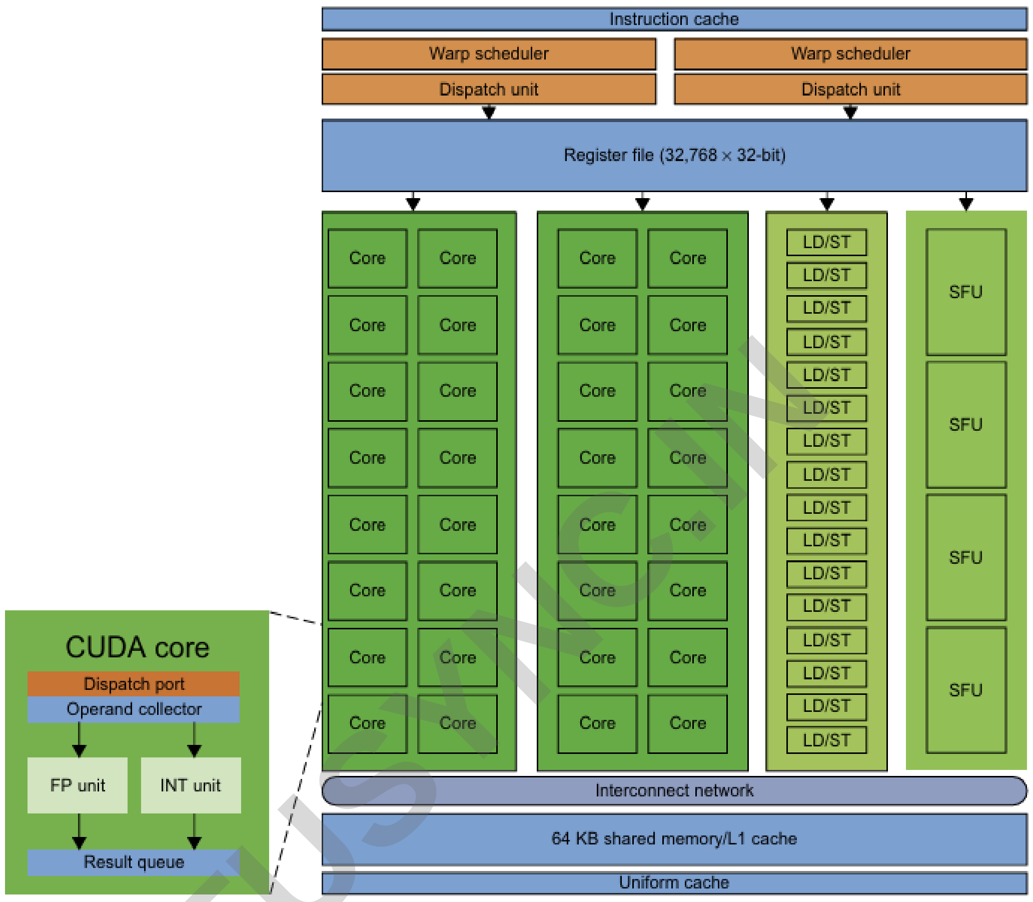


FIGURE 1.8

Power Efficiency of GPUs

- GPUs offer **better performance per watt** than CPUs.
- **Energy consumption**:
  - **CPU:** ~2 nJ per instruction.
  - **GPU:** ~200 pJ per instruction (10x more power-efficient).
- **Challenges in future computing**:
  - Power consumption constraints.
  - Optimization of **storage hierarchy** and **memory management**.

- Need for **self-aware OS, locality-aware compilers, and auto-tuners** for GPU-based computing.

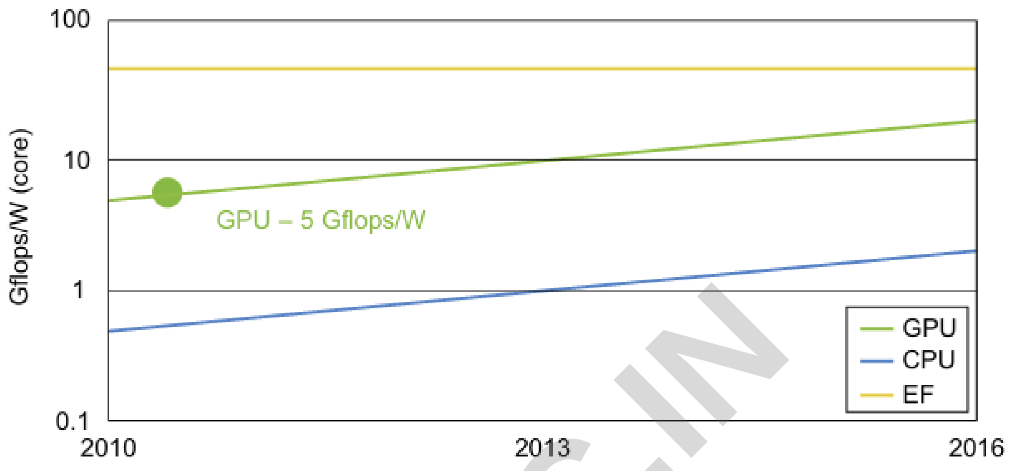


FIGURE 1.9

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.

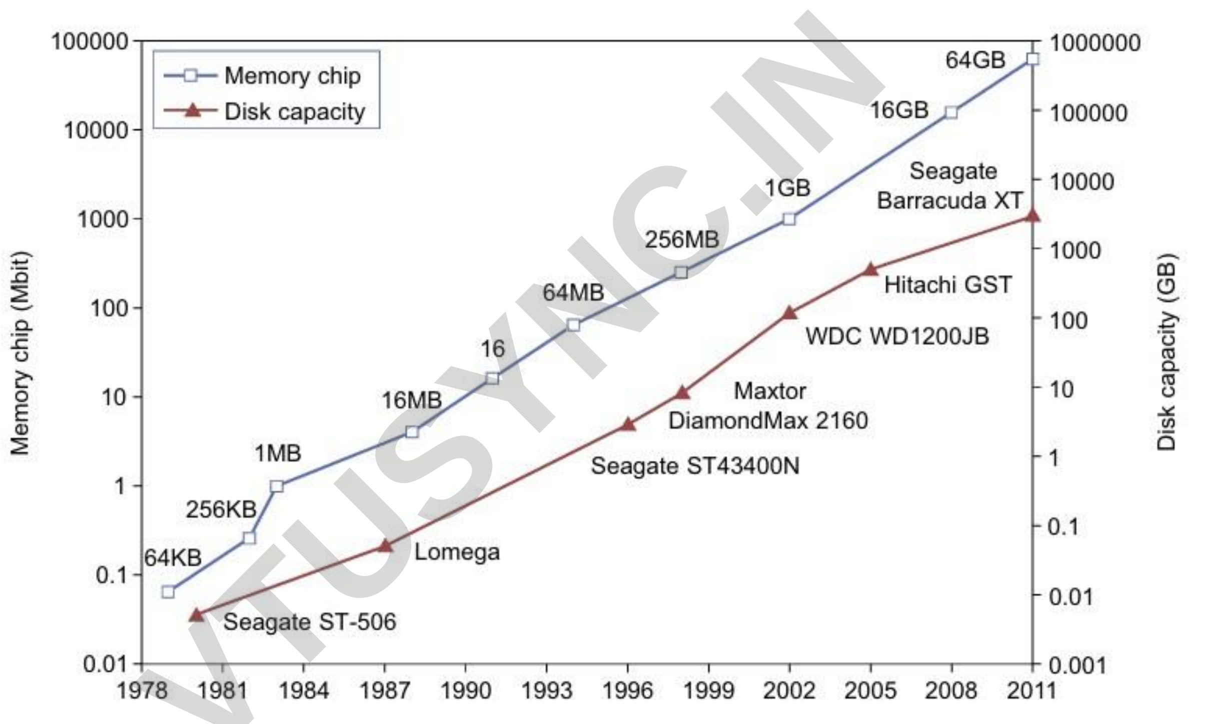


FIGURE 1.10

System-Area Interconnects

Small clusters use **Ethernet switches** or **Local Area Networks (LANs)**.

Types of storage networks:

- **Storage Area Network (SAN)** – connects servers to network storage.
- **Network Attached Storage (NAS)** – allows client hosts direct disk access.

Smaller clusters use **Gigabit Ethernet** with copper cables.

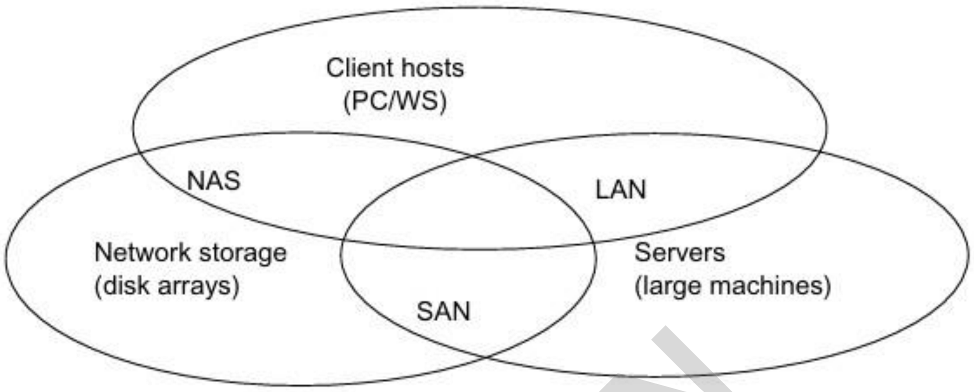


FIGURE 1.11

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

- Traditional computers have a single OS, tightly coupling applications to hardware.
- Virtual Machines (VMs) provide flexibility, resource utilization, software manageability, and security.
- Virtualization enables access to computing, storage, and networking resources dynamically.
- Middleware like Virtual Machine Monitors (VMMs) or hypervisors manage VMs.

VM Architectures

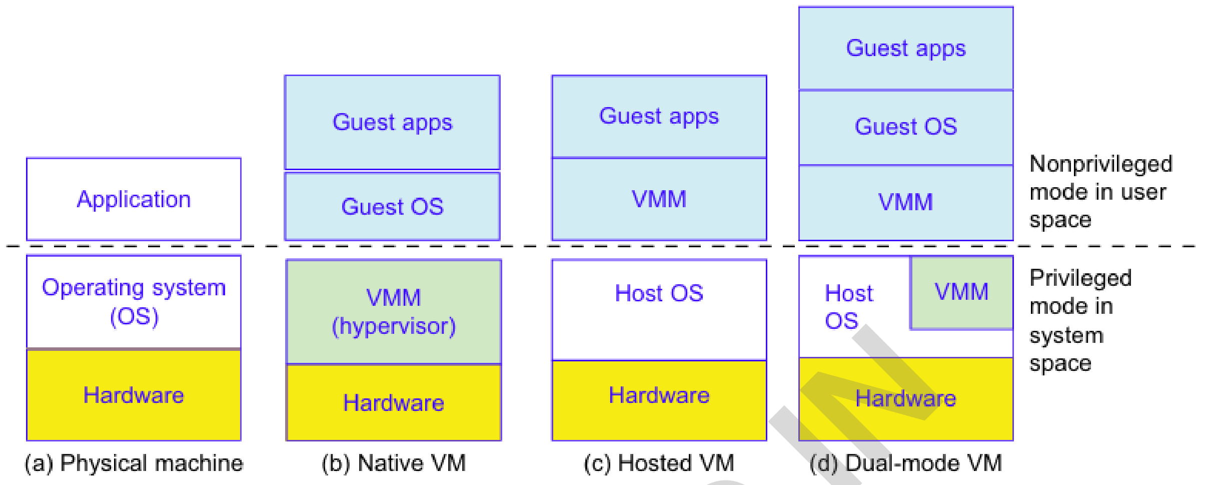


FIGURE 1.12

- **Host Machine:** Physical hardware runs an OS (e.g., Windows).
- **Native VM (Bare-Metal):** A hypervisor directly manages hardware, running guest OS (e.g., XEN on Linux).
- **Host VM:** The VMM runs in non-privileged mode without modifying the host OS.
- **Hybrid VM:** VMM operates at both user and supervisor levels, requiring host OS modifications.
- **Advantages:** OS independence, application portability, hardware abstraction.

VM Primitive Operations

- **Multiplexing:** Multiple VMs run on a single hardware machine.
- **Suspension & Storage:** VM state is saved for later use.
- **Resumption:** Suspended VM can be restored on a different machine.
- **Migration:** VM moves across platforms, enhancing flexibility.
- **Benefits:** Improved resource utilization, reduced server sprawl, increased efficiency (VMware reports 60–80% server utilization).

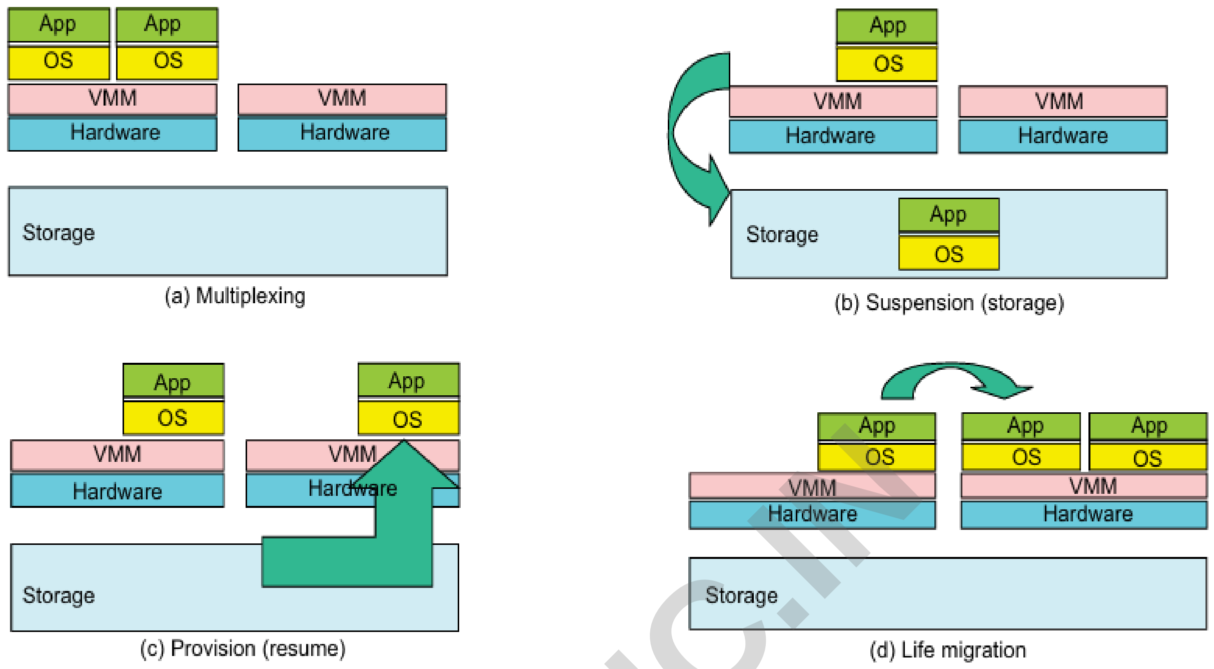


FIGURE 1.13

Virtual Infrastructures

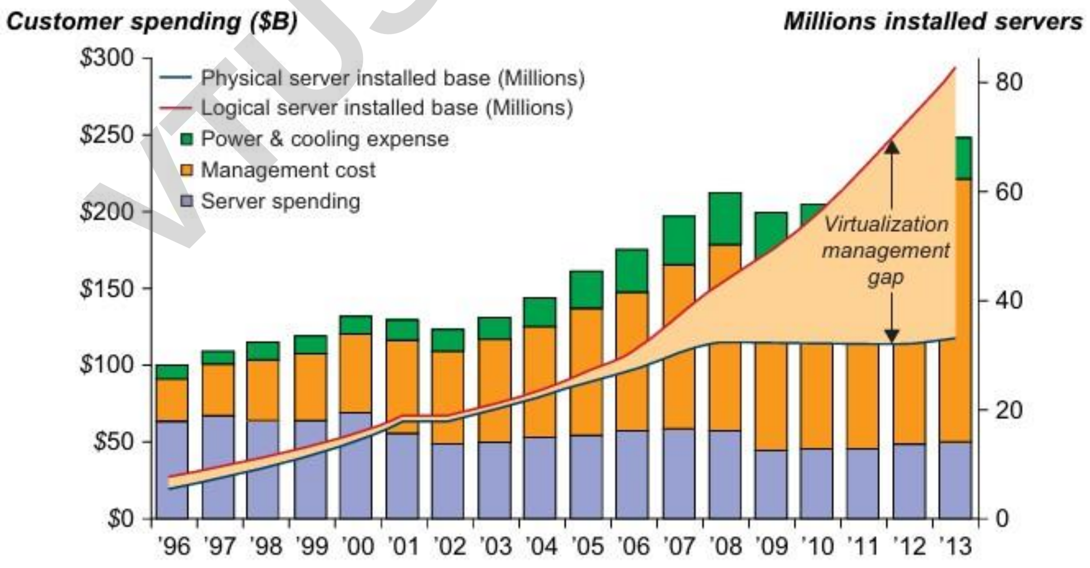


FIGURE 1.14

- Maps physical resources (compute, storage, networking) to virtualized applications.
- Separates hardware from software, reducing costs and increasing efficiency.
- Supports cloud computing through dynamic resource mapping.

Data Center 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 Center Growth & Cost Breakdown

Large data centers contain thousands of servers.

Cost Distribution (2009 IDC Report):

- 30% for IT equipment (servers, storage).
- 60% for maintenance and management (cooling, power, etc.).
- 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.

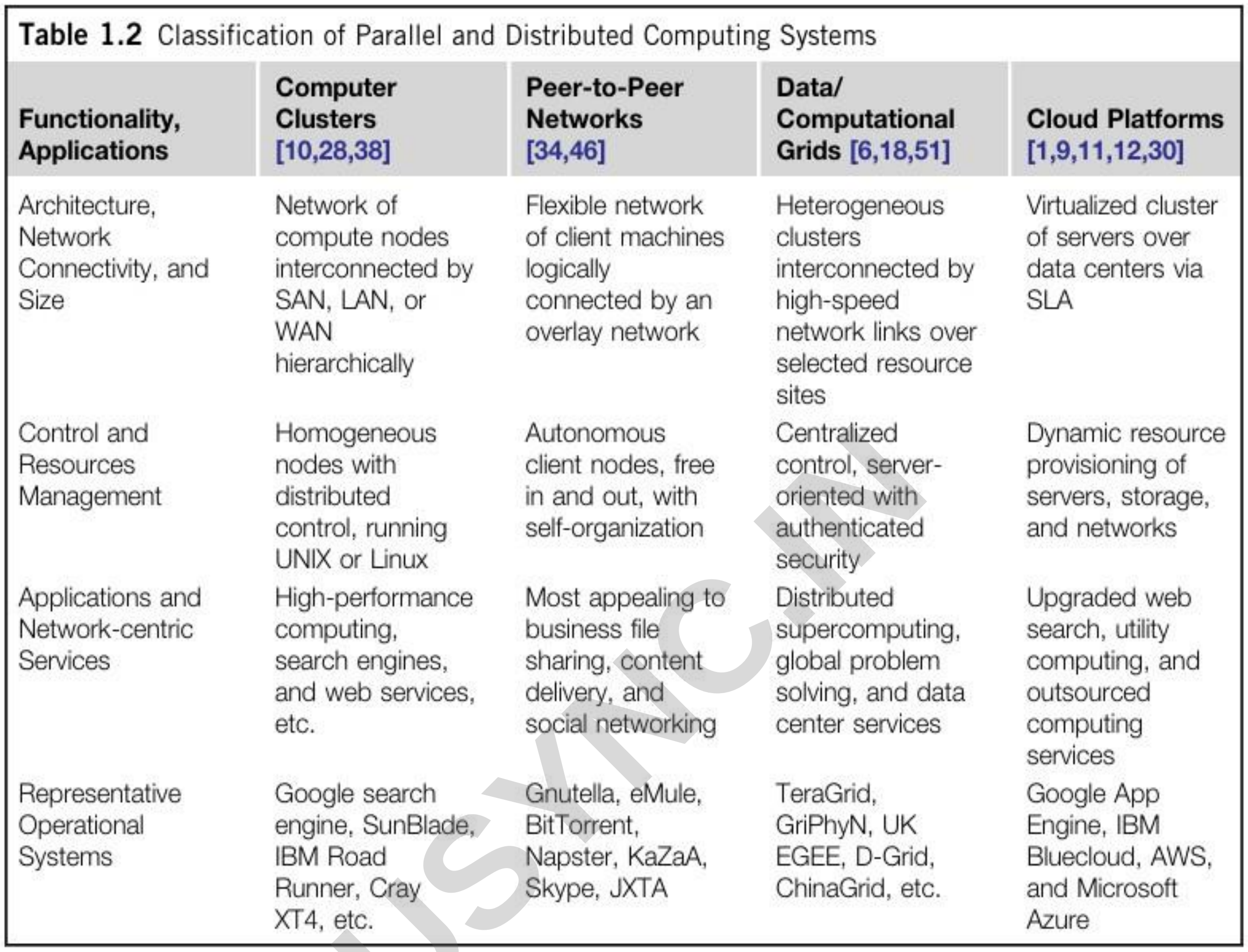


Table 1.2 Classification of Parallel and Distributed Computing Systems

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

Classification of Massive Systems

- Four major types: **Clusters, P2P Networks, Computing Grids, and Internet Clouds**.
- Involves hundreds, thousands, or even millions of participating nodes.
- **Clusters**: Popular in supercomputing applications.
- **P2P Networks**: Used in business applications but face copyright concerns.
- **Grids**: Underutilized due to middleware and application inefficiencies.
- **Cloud Computing**: Cost-effective and simple for providers and users.

Clusters of Cooperative Computers

A computing cluster consists of interconnected computers working as a single unit.

Handles **heavy workloads and large datasets** efficiently.

Cluster Architecture

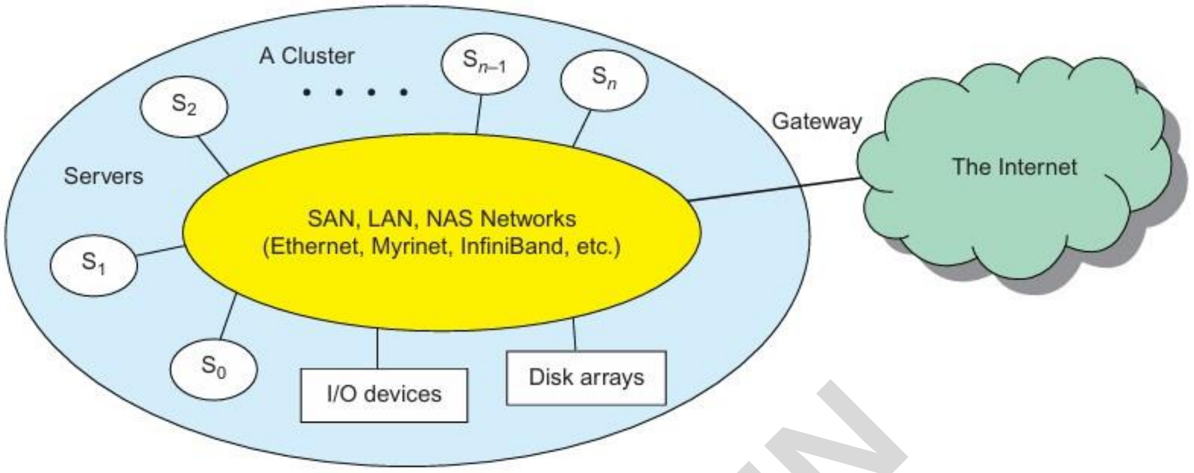


FIGURE 1.15

- Uses **low-latency, high-bandwidth interconnection networks** (e.g., SAN, LAN).
- Built using **Gigabit Ethernet, Myrinet, or InfiniBand switches**.
- Connected to the Internet via **VPN gateways**.
- Cluster nodes often run under **different OS**, leading to multiple system images.

Single-System Image (SSI)

- **Ideal cluster design** merges multiple system images into a single-system image.
- SSI makes a cluster appear as **a single machine** to users.
- Achieved using **middleware or OS extensions**.

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

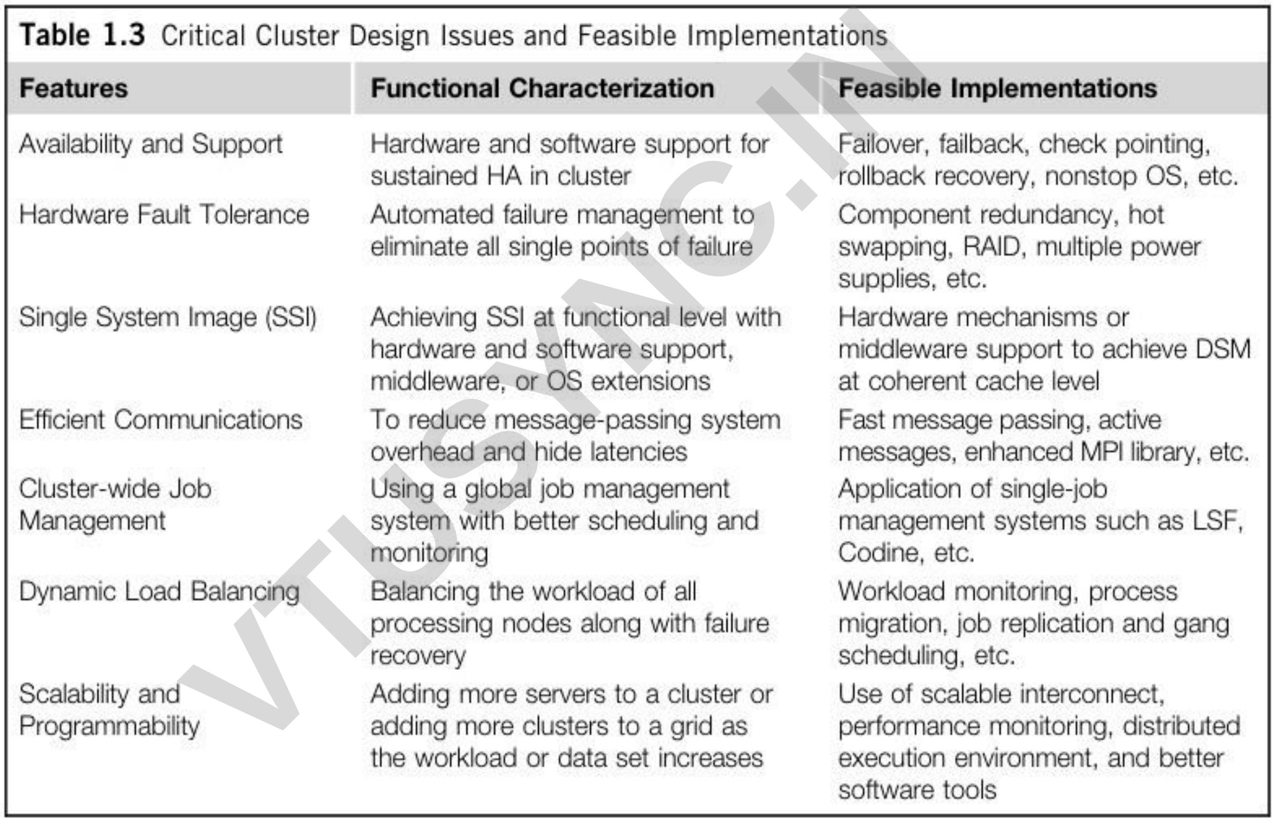


Table 1.3 Critical Cluster Design Issues and Feasible Implementations

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

- **Evolution**: Internet → Web → Grid computing.
- Enables **interaction among applications running on distant computers**.
- Grid computing supports a **rapidly growing IT-based economy**.

Computational Grids

- Similar to an **electric power grid**, integrates **computers, software, middleware, and users**.
- Constructed across **LANs, WANs, or the Internet** at various scales.
- **Virtual platforms for supporting virtual organizations**.
- Computers used: **Workstations, servers, clusters, supercomputers**.
- **Personal devices (PCs, laptops, PDAs)** can also access grid systems.

Example of a Computational Grid

- Built over multiple **resource sites owned by different organizations**.
- Offers **diverse computing resources** (e.g., workstations, large servers, Linux clusters).
- Uses **broadband IP networks** (LANs, WANs) to integrate computing, communication, and content.
- **Special applications**:
  - **SETI@Home** (search for extraterrestrial life).
  - **Astrophysics research (pulsars)**.
- Examples of large grids:
  - **NSF TeraGrid (USA)**.
  - **EGEE (Europe)**.
  - **ChinaGrid (China)**.

Grid Families

Definition: Grid computing integrates distributed resources to solve large-scale computing problems.

Types:

Computational/Data Grids: Built at a national level for large-scale computing and data sharing.

P2P Grids: Decentralized and self-organizing without a central control.

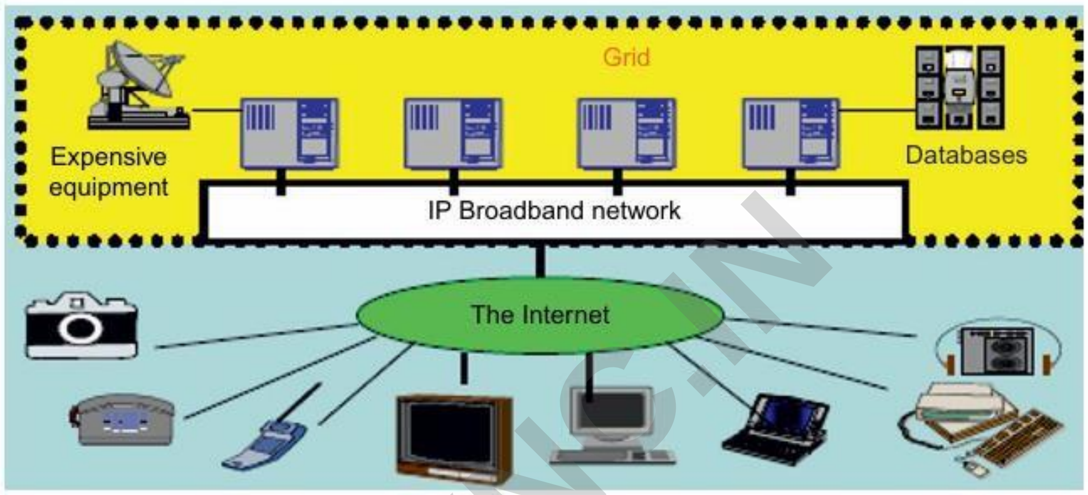


FIGURE 1.16

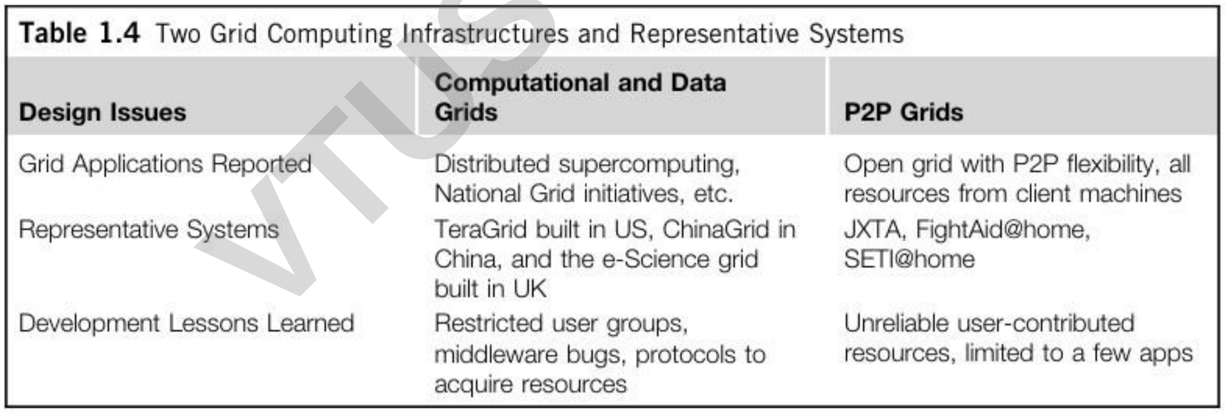


Table 1.4 Two Grid Computing Infrastructures and Representative Systems

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 Model: Traditional architecture where clients connect to a central server for computing resources.

P2P Architecture: Decentralized, with each node acting as both a client and a server.

P2P Systems

Decentralized Control: No master-slave relationship; no global view of the system.

Self-Organizing: Peers join and leave voluntarily.

Ad Hoc Network: Uses the Internet (TCP/IP, NAI protocols). Overlay

Networks

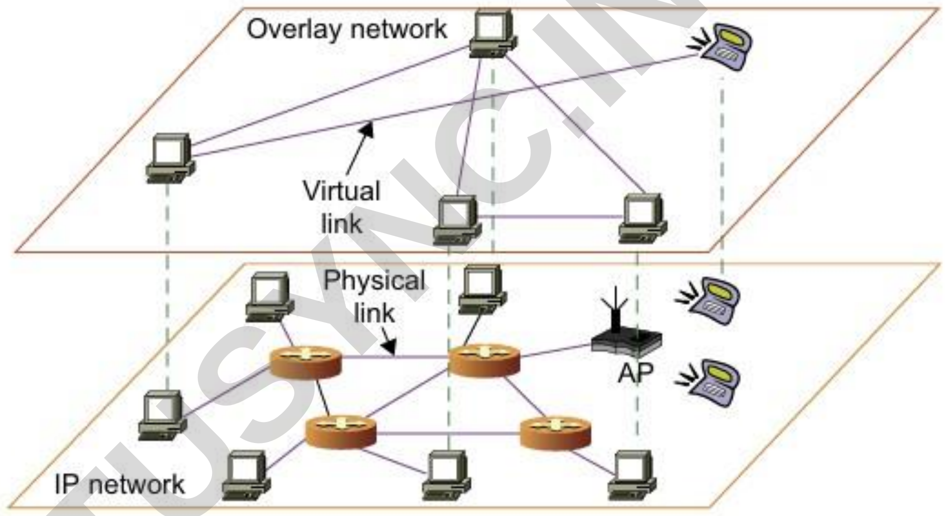


FIGURE 1.17

Virtual network over a physical P2P network.

Types:

Unstructured Overlay: Random connections, flooding-based search.

Structured Overlay: Organized topology, efficient routing.

P2P Application Families

File Sharing: E.g., BitTorrent, Napster.

Collaboration: E.g., Skype, MSN. **Distributed**

Computing: E.g., SETI@home. **P2P**

Platforms: E.g., JXTA, .NET.

P2P Computing Challenges

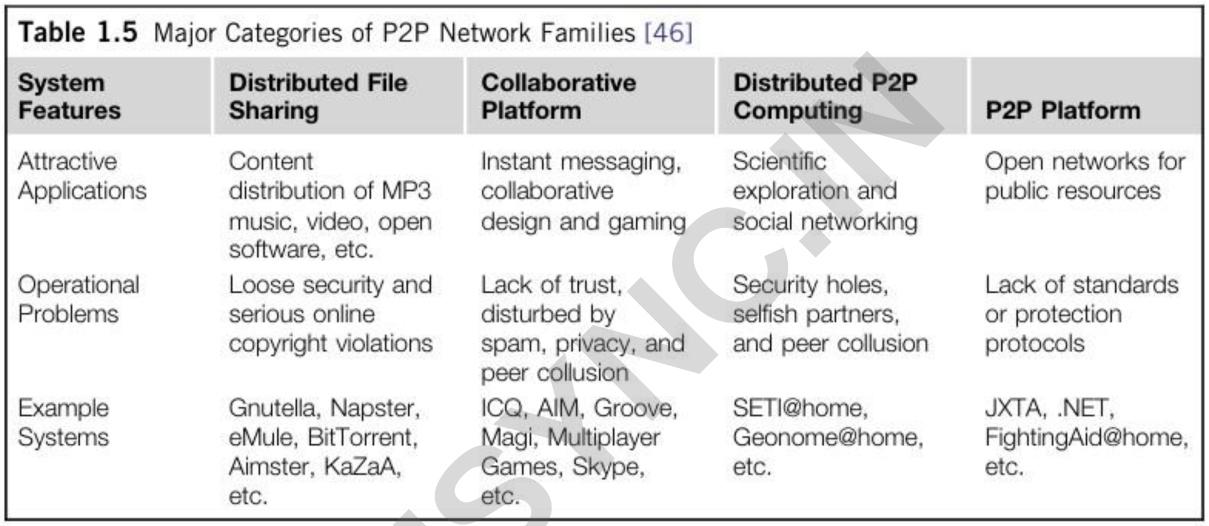


Table 1.5 Major Categories of P2P Network Families [46]

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.

- **Heterogeneity**: Hardware, software, and network incompatibilities.
- **Scalability**: Handling increased workloads.
- **Security & Privacy**: Lack of trust, copyright concerns.
- **Performance Issues**: Data location, routing efficiency, load balancing.
- **Fault Tolerance**: Replication prevents single points of failure.

Cloud Computing Over the Internet

Cloud computing is revolutionizing computational science by enabling large-scale data processing with efficient resource allocation.

It shifts computing and data storage from desktops to centralized data centers, offering on-demand services.

Cloud Computing

- Moves computing from desktops to large data centers.
- Enables on-demand software, hardware, and data as a service.
- Supports scalability, redundancy, and self-recovering systems.

Internet Clouds

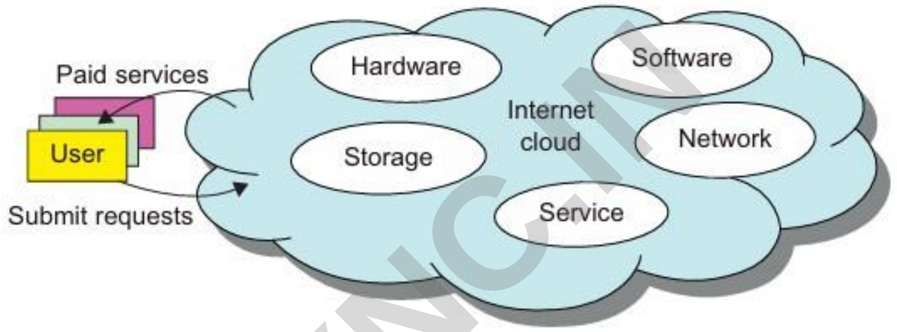


FIGURE 1.18

- Uses virtualized platforms with dynamic resource provisioning.
- Provides cost-effective solutions for both users and providers.
- Ensures security, trust, and dependability in cloud operations.

The Cloud Landscape

Cloud computing addresses challenges in traditional distributed computing systems, such as maintenance, poor utilization, and high costs.

It provides scalable, on-demand computing resources through various service models.

Challenges in Traditional Systems

- Require constant maintenance.
- Suffer from poor resource utilization.
- Have high costs for hardware and software upgrades.

Cloud Computing as a Solution

- Provides an on-demand computing paradigm.
- Offers scalable and cost-efficient alternatives to traditional systems.

Cloud Service Models

- **Infrastructure as a Service (IaaS)**: Provides virtualized computing resources (servers, storage, networking). Users manage applications but not infrastructure.
- **Platform as a Service (PaaS):** Offers a virtualized development platform with middleware, databases, and APIs (e.g., Java, Python, Web 2.0).
- **Software as a Service (SaaS)**: Delivers software applications via browsers (e.g., CRM, ERP, HR systems) without upfront infrastructure investment.

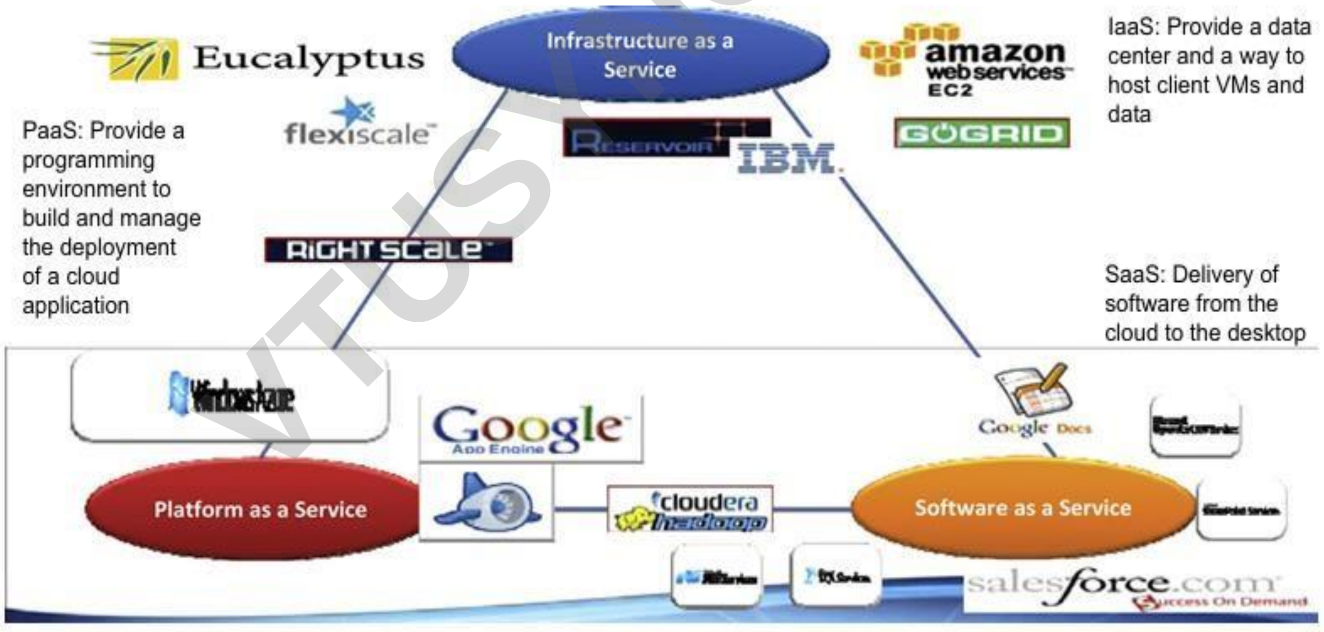

Infrastructure as a Service
Eucalyptus
amazon webservices EC2
IaaS: Provide a data center and a way to host client VMs and data
PaaS: Provide a programming environment to build and manage the deployment of a cloud application
flexiscale
Reservoir
IBM
GOGRID
RightScale
SaaS: Delivery of software from the cloud to the desktop
Windows Azure
Google App Engine
Google Docs
Platform as a Service
cloudera hadoop
Software as a Service
salesforce.com Success On Demand

FIGURE 1.19

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

Cloud Deployment Models

- **Private Cloud**: Exclusive use by a single organization, ensuring high security.
- **Public Cloud**: Services available to multiple users with lower costs.
- **Hybrid Cloud**: Combines private and public cloud benefits.
- **Managed Cloud**: Maintained by third-party providers for efficient management.

Security Considerations

Different service level agreements (SLAs) define security responsibilities.

Security is shared among cloud providers, users, and third-party software providers.

Benefits of Cloud Computing

1. Efficient location-based data centers with better energy management.
2. Better resource utilization through peak-load sharing.
3. Reduces infrastructure maintenance efforts.
4. Significantly lowers computing costs.
5. Facilitates cloud-based programming and development.
6. Enhances service and data discovery, along with content distribution.
7. Addresses privacy, security, and reliability challenges.
8. Supports flexible service agreements, business models, and pricing policies.

Software Environments for Distributed Systems and Clouds

This section explores the software environments used in distributed and cloud computing, focusing on **Service-Oriented Architecture (SOA)**, web services, REST, and the evolving relationship between grids and clouds.

Service-Oriented Architecture (SOA) in Distributed Systems

Entities in SOA:

- **Grids/Web Services** → Services

- **Java** → Java Objects
- **CORBA** → Distributed Objects

SOA builds on the OSI networking model, using middleware like .NET, Apache Axis, and Java Virtual Machine.

Higher-level environments handle entity interfaces and inter-entity communication, rebuilding the OSI layers at the software level.

Layered Architecture for Web Services and Grids

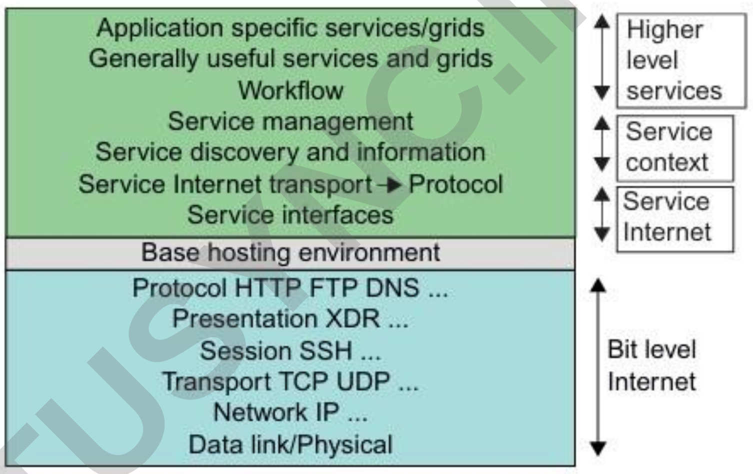


FIGURE 1.20

Entity interfaces include Web Services Description Language (WSDL), Java methods, and CORBA IDL.

Communication systems: SOAP (Web services), RMI (Java), IIOP (CORBA).

Middleware support: WebSphere MQ, Java Message Service (JMS) for messaging, fault tolerance, and security.

Service discovery models: JNDI (Java), UDDI, LDAP, ebXML, CORBA Trading Service.

Management services: CORBA Life Cycle, Enterprise JavaBeans, Jini lifetime model, and web services frameworks.

Web Services vs. REST Architecture **Web**

Services (SOAP-based):

- Fully specifies service behavior and environment.
- Uses SOAP messages for universal distributed OS functionality.
- Implementation challenges due to complexity.

REST (Representational State Transfer):

- Focuses on simplicity and lightweight communication.
- Uses "XML over HTTP" for rapid technology environments.
- Suitable for modern web applications.

Evolution of SOA

- **SOA enables integration across** grids, clouds, interclouds, and IoT.
- **Sensors (SS) collect raw data**, which is processed through compute, storage, filter, and discovery clouds.
- **Filter clouds** eliminate unnecessary data to refine information for decision-making.
- **Portals (e.g., OGFCE, HUBzero)** serve as access points for users.

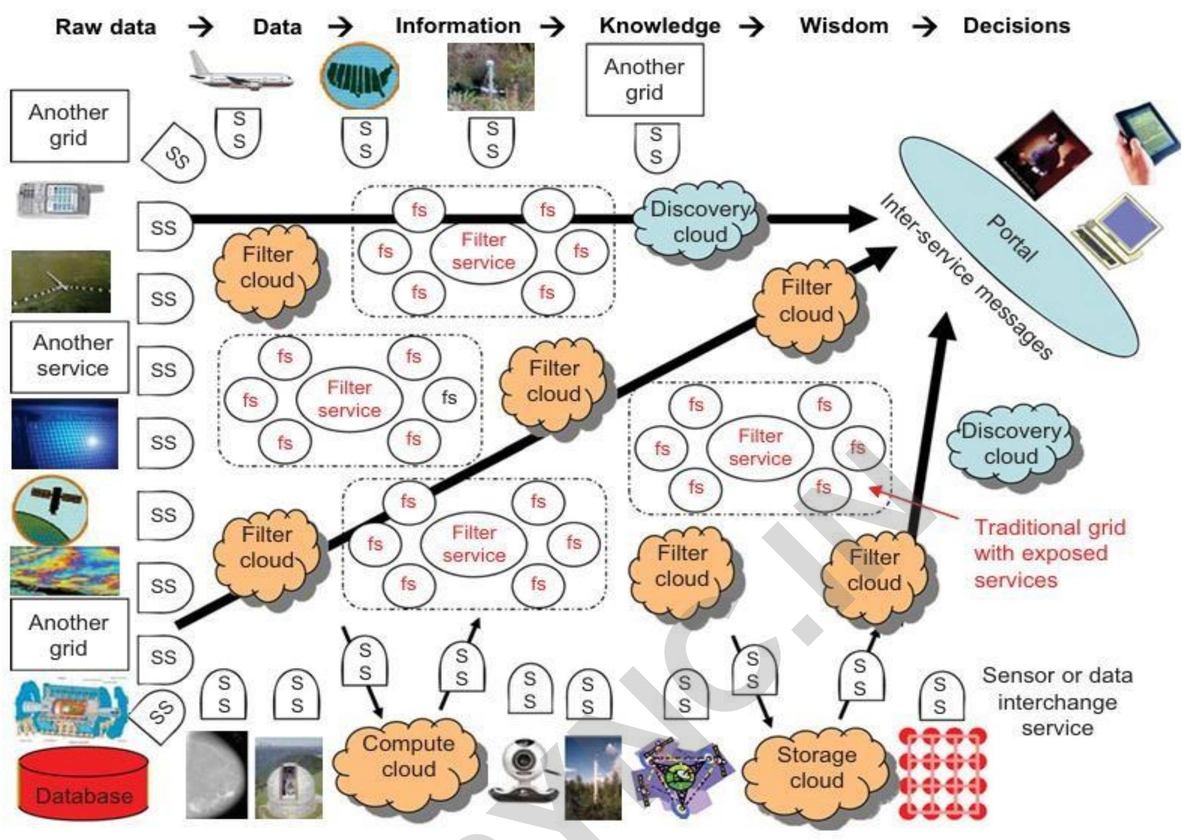

Raw data → Data → Information → Knowledge → Wisdom → Decisions
Another grid
Another service
SS
fs
Filter service
Filter cloud
Discovery cloud
Compute cloud
Storage cloud
Database
Portal
Inter-service messages
Traditional grid with exposed services
Sensor or data interchange service

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 vs. Clouds

Grid Computing:

- Uses **static resources** allocated in advance.
- Focuses on distributed computing with a defined structure.

Cloud Computing:

- Uses **elastic resources** that scale dynamically.
- Supports virtualization and autonomic computing.

Hybrid Approach:

- Grids can be built out of multiple clouds for **better resource allocation**.
- Models include **cloud of clouds, grid of clouds, and interclouds**.

Workflow Coordination in Distributed Systems:

- Technologies like BPEL Web Services, Pegasus, Taverna, Kepler, Trident, and Swift help manage distributed services.

Trends Toward Distributed Operating Systems

Distributed Operating Systems (DOS)

- Distributed systems have **multiple system images** due to independent OS on each node.
- A **distributed OS** enhances resource sharing and fast communication using **message passing** and **RPCs**.
- It improves **performance, efficiency, and flexibility** of distributed applications.

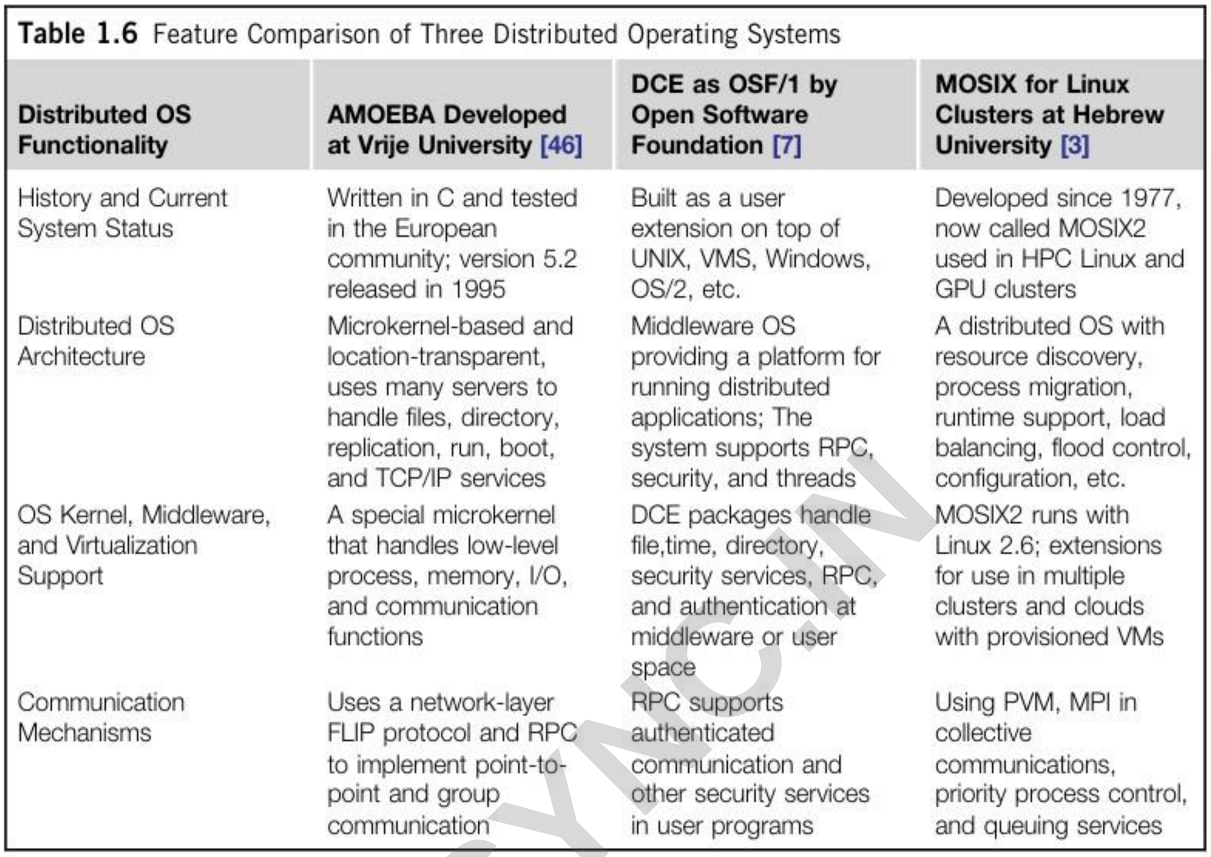


Table 1.6 Feature Comparison of Three Distributed Operating Systems

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

Approaches to Distributed Resource Management (Tanenbaum's Classification)

1. **Network OS** – Built over multiple heterogeneous OS platforms; offers lowest transparency, mainly used for **file sharing**.
2. **Middleware-based OS** – Provides limited resource sharing (e.g., **MOSIX/OS** for clusters).
3. **Truly Distributed OS** – Provides **higher transparency** and better resource management.

Comparison of Distributed OS (Amoeba vs. DCE)

- **Amoeba** (Netherlands) and **DCE** (Open Software Foundation) are research prototypes.
- No commercial OS has succeeded in following these systems.

- **Future trends** focus on **web-based OS** for virtualization and lightweight **microkernel designs**.

MOSIX2 for Linux Clusters

- **MOSIX2** is a **distributed OS** with a **virtualization layer** for Linux.
- Provides **single-system image**, supports **sequential and parallel applications**.
- Enables **resource migration** across Linux nodes in clusters and grids.
- Used in **Linux clusters, GPU clusters, grid computing, and cloud environments**.

Transparency in Programming Environments

Computing infrastructure is divided into **four levels**:

1. **User Data** (separated from applications).
2. **Applications** (runs on multiple OSes).
3. **Operating Systems** (provide standard interfaces).
4. **Hardware** (standardized across OSes).

Future **cloud computing** will allow users to **switch OS and applications easily**.

Parallel and Distributed Programming Models

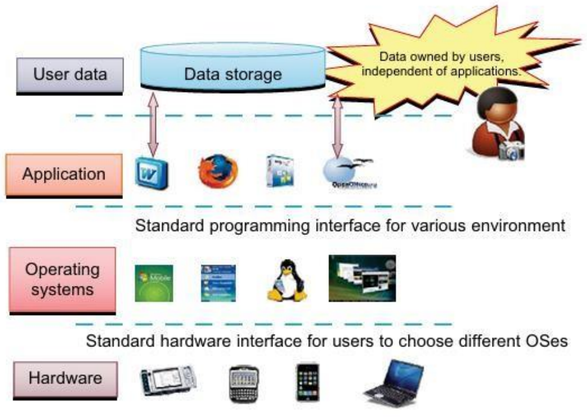

User data
Data storage
Data owned by users, independent of applications.
Application
Standard programming interface for various environment
Operating systems
Standard hardware interface for users to choose different OSes
Hardware

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.

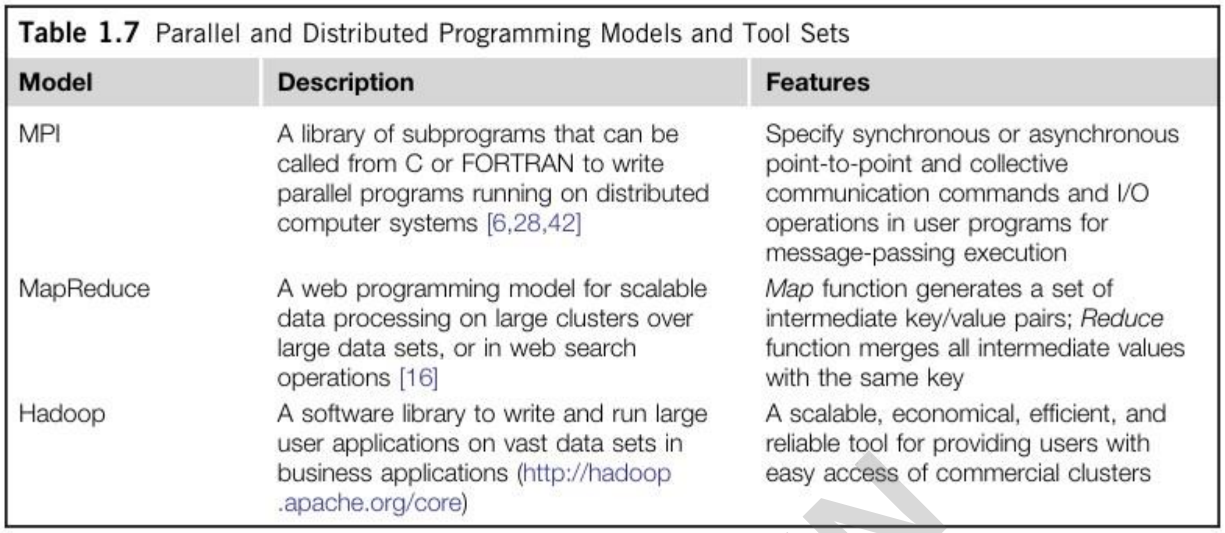


Table 1.7 Parallel and Distributed Programming Models and Tool Sets

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)

- A standard library for **parallel programming** in C and FORTRAN.
- Used in **clusters, grid systems, and P2P networks**.
- Alternative: **Parallel Virtual Machine (PVM)**.

MapReduce

- **Scalable data processing** model for **large clusters** (Google).
- Uses **Map (key/value generation)** and **Reduce (merging values)** functions.
- Handles **terabytes of data** across thousands of machines.

Hadoop

- **Open-source** version of MapReduce, initially developed by Yahoo!.
- Enables **massive data processing** over distributed storage.
- Provides **high parallelism, reliability, and scalability**.

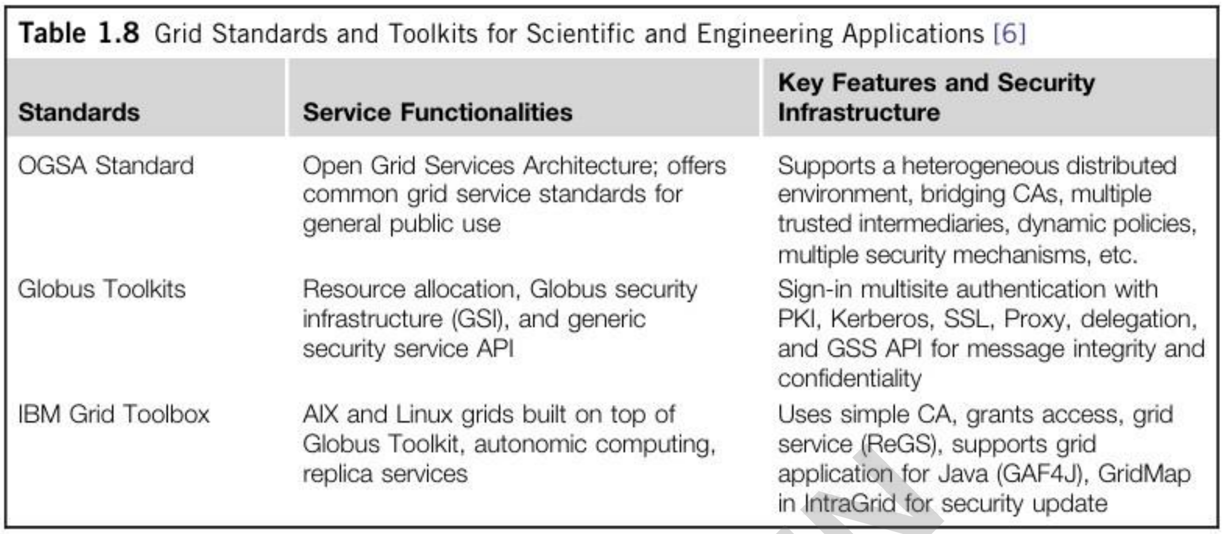


Table 1.8 Grid Standards and Toolkits for Scientific and Engineering Applications [6]

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)

- A standard for **grid computing**.
- Supports **distributed execution, security policies, and trust management**.
- **Genesis II** is an OGSA-based implementation.

Globus Toolkits

- Middleware for **resource allocation, security, and authentication** in grid computing.
- Developed by **Argonne National Lab and USC**.
- IBM extended Globus for **business applications**.

Performance, Security and Energy Efficiency

Performance Metrics and Scalability Analysis

Distributed system performance depends on several factors, including CPU speed (MIPS), network bandwidth (Mbps), system throughput (Tflops, TPS), job response time, and network latency.

High-performance interconnection networks require low latency and high bandwidth. Other key metrics include OS boot time, compile time, I/O data rate, system availability, dependability, and security resilience.

Dimensions of Scalability

1. **Size Scalability** – Increasing the number of processors, memory, or I/O channels to improve performance.
2. **Software Scalability** – Upgrading OS, compilers, and application software to work efficiently in large systems.
3. **Application Scalability** – Adjusting problem size to match machine scalability.
4. **Technology Scalability** – Adapting to hardware and networking advancements while ensuring compatibility with existing systems.

Scalability versus OS Image Count

Scalability is affected by OS image count. SMP systems have a single OS image, limiting scalability, whereas NUMA, clusters, and cloud environments support multiple OS images, enabling higher scalability.

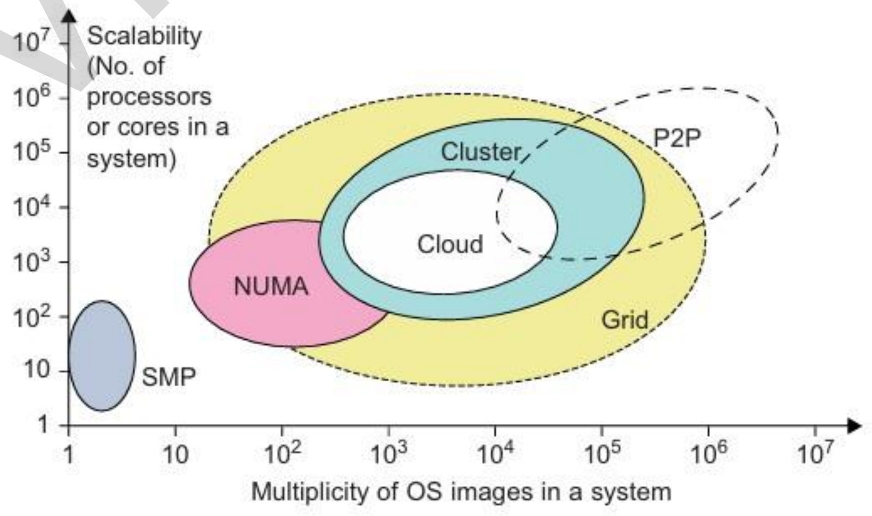


FIGURE 1.23

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

Performance Models: Amdahl's and Gustafson's Laws

Amdahl's Law states that system speedup is limited by the sequential portion of a program. Even with infinite processors, speedup is constrained by non-parallelizable code.

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

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:

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

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.

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 *system efficiency* defined as follows:

$$E = S/n = 1/[\alpha n + 1 - \alpha] \quad (1.2)$$

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.

Gustafson's Law addresses this limitation by scaling the workload along with system size, resulting in better efficiency in large distributed systems.

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:

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

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

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

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.

System Availability and Fault Tolerance

High Availability (HA) is crucial in clusters, grids, P2P networks, and clouds. It is defined as: System Availability=

$$\text{System Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

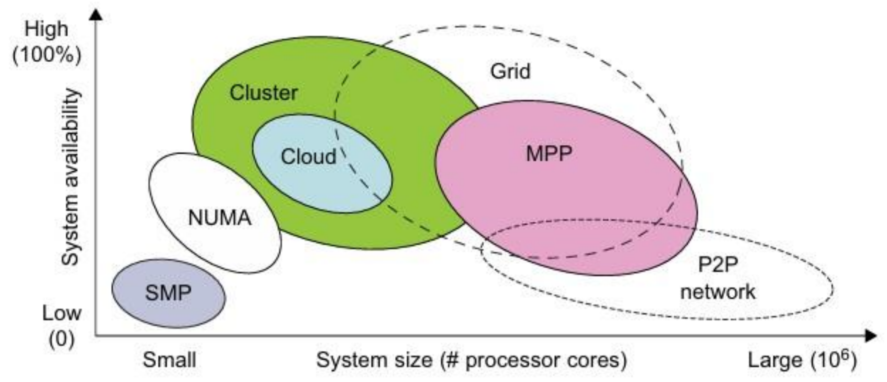


FIGURE 1.24

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

Fault Tolerance Strategies include redundancy, component reliability, and failover mechanisms.

As system size increases, availability decreases due to higher failure probability. Grids and clusters have better fault isolation than SMP and MPP systems, while P2P networks have the lowest availability.

Security Challenges in Distributed Systems

Common Threats:

- **Information leaks** (loss of confidentiality)
- **Data integrity breaches** (Trojan horses, user alterations)
- **Denial of Service (DoS) attacks** (disrupting system operation)
- **Unauthorized access** (exploiting open computing resources)

Security Responsibilities in Cloud Models:

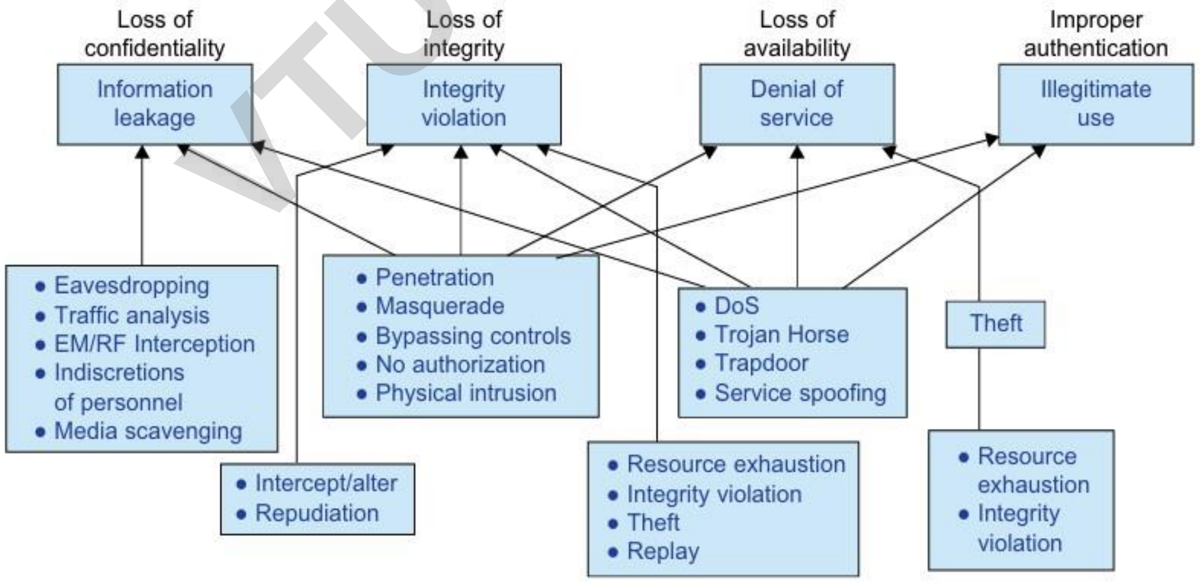


FIGURE 1.25

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

- **SaaS (Software as a Service)** – Provider handles all security.
- **PaaS (Platform as a Service)** – Provider ensures data integrity and availability, but users manage confidentiality.
- **IaaS (Infrastructure as a Service)** – Users handle most security functions, while providers ensure availability.

Copyright Protection in P2P Networks:

Collusive Piracy: Paid clients (colluders) share copyrighted content with unpaid clients (pirates), affecting commercial content delivery.

Content Poisoning Scheme: Proactively detects and prevents piracy using identity-based signatures and timestamped tokens, protecting legitimate clients while stopping colluders and pirates.

Reputation Systems: Essential in detecting and addressing piracy in P2P networks and digital content sharing.

System Defense Technologies:

First Generation: Tools focused on preventing intrusions through access control policies, cryptography, and tokens.

Second Generation: Tools for detecting intrusions (e.g., firewalls, IDS, reputation systems) and triggering remedial actions.

Third Generation: Intelligent systems that respond to intrusions and adapt to security threats.

Data Protection Infrastructure:

Security Infrastructure: Involves trust negotiation, reputation aggregation, and intrusion detection against viruses and DDoS attacks.

Cloud Security: Cloud service models (IaaS, PaaS, SaaS) divide security responsibilities:

- **IaaS**: Users manage confidentiality; providers manage data integrity and availability.
- **PaaS and SaaS**: Both providers and users share responsibility for data integrity and confidentiality.

Piracy Prevention: Measures against online piracy and copyright violations in digital content.

Energy Efficiency in Distributed Computing:

1. **Energy Consumption Challenges**: Systems face rising energy costs, especially in large-scale data centers and HPC systems (e.g., Earth Simulator, Petaflop).

2. **Unused Servers**: Many servers in data centers are left powered on without use, leading to significant energy waste (e.g., 4.7 million idle servers globally).
   - **Potential Savings**: Estimated savings of $3.8 billion in energy costs and $24.7 billion in operational costs from turning off unused servers.

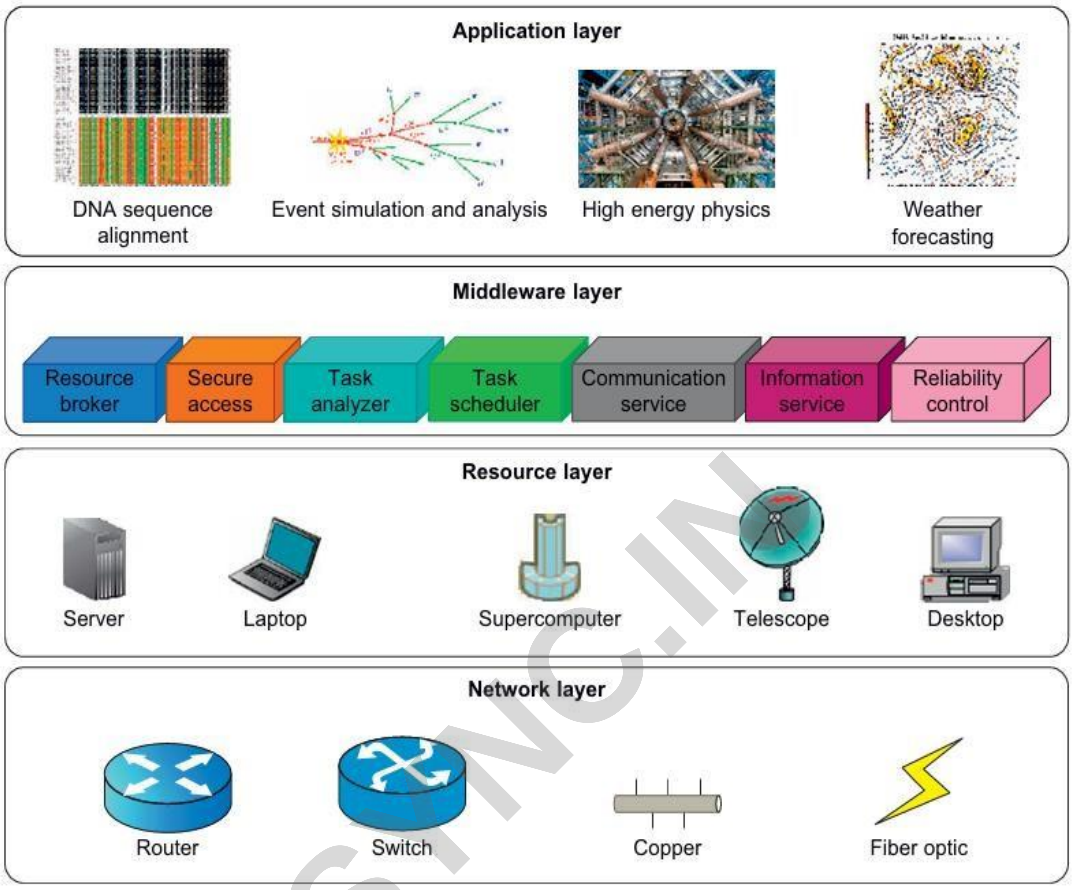


FIGURE 1.26

Four operational layers of distributed computing systems.

3. **Energy in Active Servers**: Techniques needed to reduce energy consumption without affecting performance.

Energy Management in Distributed Systems (Four Layers): **Application**

Layer:

- Focus on energy-aware applications that balance energy consumption with performance.
- Key factors: Instruction count and storage transactions affect energy use.

Middleware Layer:

- Manages energy-efficient scheduling and task management.

- Incorporates energy-aware techniques to optimize power usage during task scheduling.

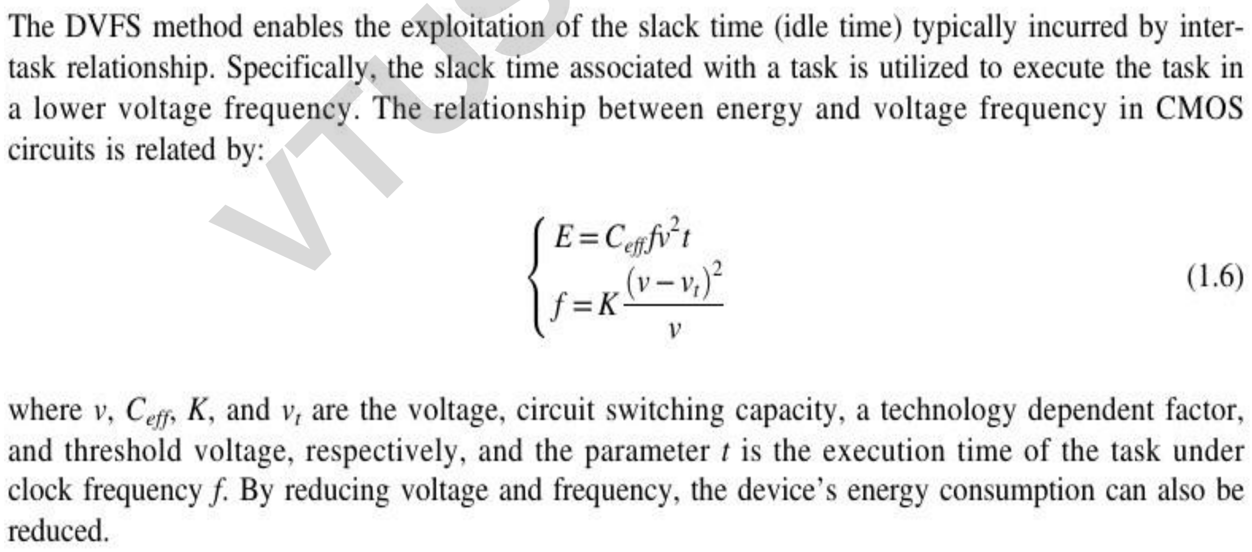
Resource Layer:

- Manages hardware (e.g., CPU) and operating systems to optimize energy usage.
- **Dynamic Power Management (DPM)**: Switches between idle and lower-power states.
- **Dynamic Voltage-Frequency Scaling (DVFS)**: Controls power consumption by adjusting voltage and frequency.

Network Layer:

- Focuses on energy-efficient network routing and protocols.
- New energy-efficient routing algorithms and models are needed for optimized performance and reduced energy consumption.

Energy Efficiency Techniques:



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:

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

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.

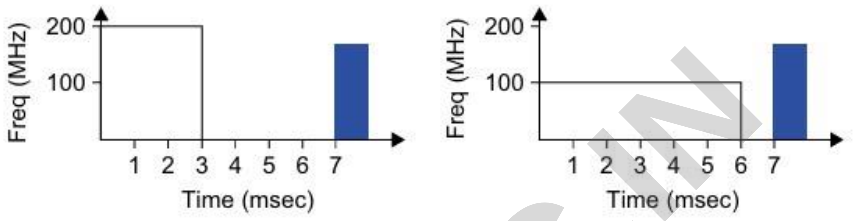


FIGURE 1.27

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