MITE

Invent Solutions

Mangalore Institute of Technology & Engineering

Moodabidri, Karnataka

Department of Artificial Intelligence and Machine Learning

Modern Computing and Python Programming

Course Code: 26ESCC106

First Year B.E. (Physics Cycle) • Common to all branches

Module 1

Modern Computing Approaches

Study Notes

Topics Covered
1. Evolution of Computing
2. Parallel and Distributed Computing
3. Cluster and Grid Computing
4. Cloud Computing
5. Edge Computing

Text Book 1: Kai Hwang, Geoffrey C. Fox and Jack J. Dongarra, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, Morgan Kaufmann, 2012.

Text Book 2: Rajkumar Buyya, Christian Vecchiola and S. Thamarai Selvi, *Mastering Cloud Computing*, McGraw Hill Education, 2013.

Contents

Topic 1: Evolution of Computing **4**

1 Scalable Computing over the Internet 4
2 The Age of Internet Computing 5
3 The Platform Evolution: Five Generations of Computers 5
4 Evolutionary Trend toward Parallel, Distributed, and Cloud Computing 6
5 High-Performance Computing and High-Throughput Computing 8
6 Three New Computing Paradigms 10
7 Computing Paradigm Distinctions 11
8 Distributed System Families and Design Objectives 13
9 Scalable Computing Trends 14
10 Degrees of Parallelism 15
11 Innovative Applications 16
12 The Trend toward Utility Computing 17
13 The Hype Cycle of New Technologies 18
14 Question Bank 20

Topic 2: Parallel and Distributed Computing **21**

1 Eras of Computing 21
2 Parallel vs. Distributed Computing 22
3 Elements of Parallel Computing: Parallel Processing 23
4 Hardware Architectures for Parallel Processing . 25
5 Shared-Memory and Distributed-Memory MIMD 27
6 Levels of Parallelism 28
7 Elements of Distributed Computing: General Concepts 30
8 Components of a Distributed System: The Layered View 30
9 A Cloud Computing Distributed System 32
10 Question Bank 33

Topic 3: Cluster and Grid Computing **34**

1 Introduction 34
2 Cluster Computing 34
3 Grid Computing 38
4 Cluster Computing versus Grid Computing . . . 41
5 Review Questions 42
6 References and Further Reading 42

Topic 4: Cloud Computing **44**

1 Introduction to Cloud Computing 44
2 The Cloud Reference Model: Architecture 44
3 Mapping the Layers to Service Models (IaaS, PaaS, SaaS) and XaaS 46
4 Infrastructure- and Hardware-as-a-Service (IaaS/HaaS) 47
5 Platform-as-a-Service (PaaS) 49
6 Software-as-a-Service (SaaS) 51
7 Types of Clouds 52
8 Public Clouds 53
9 Private Clouds 53
10 Hybrid Clouds and Cloudbursting 55
11 Community Clouds 56
12 Comparison of the Types of Clouds 57
13 Economics of the Cloud 57
14 Quick Revision 58
15 Question Bank 59

Topic 5: Edge Computing **60**

1 Introduction 60
2 Need for Edge Computing 61
3 Edge Computing Architecture 61
4 Cloud versus Edge Computing 62
5 Applications 63
6 Advantages and Limitations 63
7 Review Questions 64
8 References and Further Reading 64

How to Use These Notes

- The notes follow the order of the Module 1 syllabus. Each topic starts on a new page and its sections are numbered from 1.
- In Topics 1, 2 and 4, each section begins with a typical university-style question and its marks (6 or 8); the content below it is written so that it can be reproduced as the answer.
- **6-mark answer:** definition + 5–6 explained points (+ a diagram where asked).
- **8-mark answer:** introduction + 6–8 explained points + a neat, labelled diagram + a concluding sentence.
- Figures are numbered topic-wise (Figure 2.5 is the fifth figure of Topic 2). Always label every box and arrow when you draw them.
- Every topic ends with a question bank or review questions for practice.

Note on Preparation

AI assistance was used to draft and to typeset these notes in LaTeX. The content follows the prescribed textbooks and the syllabus scope of the course, and has been reviewed by the course coordinator.

Module 1: Modern Computing Approaches

Topic 1: Evolution of Computing

Text Book 1: Kai Hwang, Geoffrey C. Fox and Jack J. Dongarra, *Distributed and Cloud Computing*, Morgan Kaufmann, 2012 — Sections 1.1, 1.1.1, 1.1.2

1. Scalable Computing over the Internet

Q. What is scalable computing over the Internet? Explain how computing has changed from centralized to parallel and distributed computing. **[6 Marks]**

Scalable computing is computing in which a system can handle a growing amount of work simply by adding more resources such as computers, processors, and storage. **Scalable computing over the Internet** means using many computers connected through the Internet, instead of one centralized computer, to solve large-scale problems.

Over the past 60 years, computing technology has gone through a series of platform and environment changes. Instead of using a single centralized computer to solve computational problems, a parallel and distributed computing system uses **multiple computers** to solve large-scale problems over the Internet.

Evolutionary changes have taken place in four areas:

1. **Machine architecture:** The internal design of computers has changed from single processors to multicore processors, clusters, and massively parallel systems.
2. **Operating system platform:** Operating systems have evolved from single-machine OSes to systems that manage many networked and virtualized machines.
3. **Network connectivity:** Computers that once worked alone are now linked by high-speed wired and wireless networks and by the Internet.
4. **Application workload:** Applications now serve millions of users simultaneously and handle huge volumes of data.

Consequences of this change:

5. Computing has become **data-intensive**, i.e., the main challenge is storing, moving, and processing very large amounts of data.
6. Computing has become **network-centric**, i.e., the work depends on many computers communicating over a network.
7. Large-scale Internet applications such as web search, online banking, e-mail, and social networking have significantly enhanced the **quality of life** and information services in society.



Figure 1.1: Centralized computing versus parallel/distributed computing

Example: A single Google search is answered by thousands of servers working together, not by one computer.

Key Points to Remember

- Centralized → parallel and distributed computing over the Internet.
- Four areas of change: architecture, OS platform, network connectivity, application workload.
- Result: computing is data-intensive and network-centric.

2. The Age of Internet Computing

Q. Explain the age of Internet computing. Why is the Linpack benchmark no longer optimal for measuring system performance? **[6 Marks]**

Billions of people use the Internet every day. This has changed the requirements placed on computer systems in the following ways:

1. **Huge concurrent demand:** Supercomputer sites and large data centers must provide high-performance computing services to huge numbers of Internet users **concurrently** (at the same time).
2. **Linpack is no longer optimal:** The **Linpack Benchmark** measures how fast a computer solves a large system of linear equations (in flops). It is suited to ranking HPC systems by raw speed, but it does not measure how well a system serves millions of simultaneous users. Hence it is no longer the best measure of performance for Internet computing.
3. **Shift to HTC:** The emergence of computing clouds demands **High-Throughput Computing (HTC)** systems built with parallel and distributed computing technologies.
4. **Upgrading data centers:** Data centers must be upgraded with
   - fast servers,
   - large storage systems, and
   - high-bandwidth networks.
5. **Purpose:** The aim is to advance network-based computing and web services using emerging technologies.

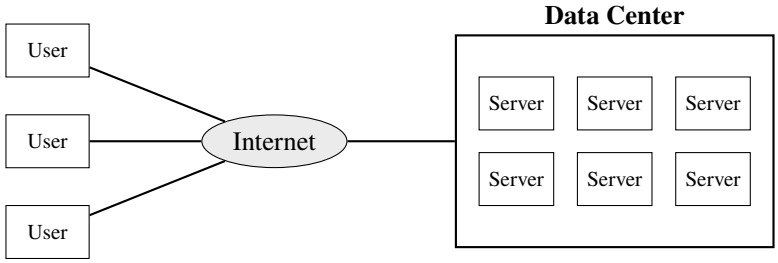


Figure 1.2: Many Internet users served concurrently by a data center

Key Points to Remember

- Billions of users ⇒ concurrent service from data centers.
- Linpack measures raw speed, not service to many users ⇒ not optimal.
- Clouds demand HTC systems; data centers need fast servers, storage, and high-bandwidth networks.

3. The Platform Evolution: Five Generations of Computers

Q. Explain the platform evolution of computer technology through five generations with suitable examples. **[6 Marks]**

Computer technology has gone through **five generations** of development. Each generation lasted **10 to 20 years**, and successive generations **overlapped by about 10 years**.

1. **1950–1970: Mainframes.** A handful of large, expensive mainframes such as the **IBM 360** and **CDC 6400** were built to satisfy the demands of large businesses and government organizations.
2. **1960–1980: Minicomputers.** Lower-cost minicomputers such as the **DEC PDP 11** and **VAX Series** became popular among small businesses and on college campuses.
3. **1970–1990: Personal computers.** Personal computers built with **VLSI microprocessors** came into widespread use by individuals and offices.
4. **1980–2000: Portable and pervasive devices.** Massive numbers of portable computers (laptops) and pervasive devices appeared in both **wired and wireless** applications.
5. **Since 1990: HPC and HTC systems.** The use of HPC and HTC systems hidden in **clusters, grids, and Internet clouds** has proliferated. These are used by consumers as well as by high-end web-scale computing and information services.

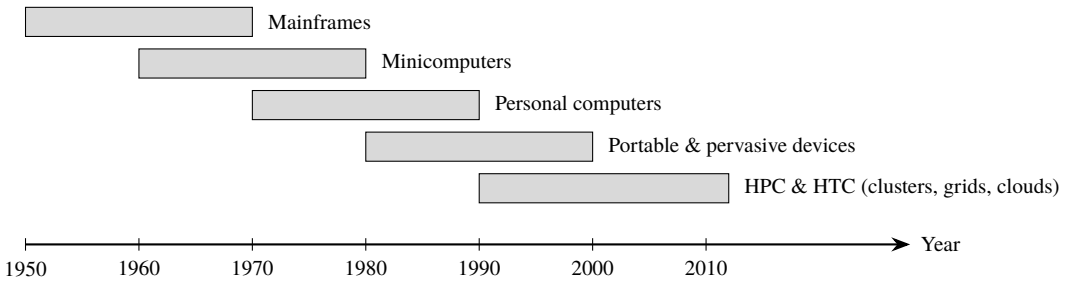


Figure 1.3: Overlapping generations of computer platforms

Period	Platform	Examples	Main Users
1950–1970	Mainframes	IBM 360, CDC 6400	Large businesses, government
1960–1980	Minicomputers	DEC PDP 11, VAX Series	Small businesses, colleges
1970–1990	Personal computers	PCs with VLSI microprocessors	Individuals, offices
1980–2000	Portable & pervasive devices	Laptops, wired/wireless devices	Mobile users
Since 1990	HPC and HTC systems	Clusters, grids, Internet clouds	Consumers, web-scale services

General trend: The general computing trend is to leverage **shared web resources** and **massive amounts of data** over the Internet.

4. Evolutionary Trend toward Parallel, Distributed, and Cloud Computing

Q. With a neat diagram, explain the evolutionary trend toward parallel, distributed, and cloud computing. **[8 Marks]**

Figure 1.4 (Text Book 1, Fig. 1.1) illustrates the evolution of HPC and HTC systems and how they lead to grids, clouds, web services, and the Internet of Things.

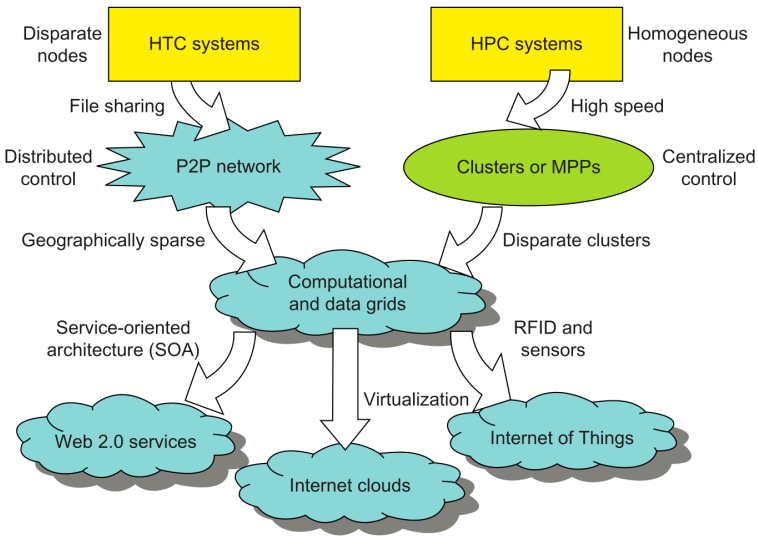


Figure 1.4: Evolutionary trend toward parallel, distributed, and cloud computing with clusters, MPPs, P2P networks, grids, clouds, web services, and the Internet of Things (Text Book 1, Fig. 1.1)

Explanation of the figure:

1. **Two starting points:** The figure starts with **HTC systems** (left) and **HPC systems** (right).
2. **HPC side:** Supercomputers, also called **Massively Parallel Processors (MPPs)**, are gradually replaced by **clusters** of cooperative computers, because organizations want to share computing resources. HPC systems use **homogeneous nodes**, **centralized control**, and aim for **high speed**.
3. **Cluster:** A cluster is a collection of homogeneous compute nodes that are physically connected in close range to one another.
4. **HTC side: Peer-to-peer (P2P) networks** are formed for distributed **file sharing** and **content delivery**. A P2P system is built over many client machines that are globally distributed. HTC systems use **disparate nodes** and **distributed control**.
5. **Focus of HTC platforms:** P2P, cloud computing, and web service platforms are focused more on HTC applications than on HPC applications.
6. **Grids:** Clustering technology (**disparate clusters**) and P2P technology (**geographically sparse** peers) together lead to **computational and data grids**.
7. **From grids to new paradigms:**
   - **Service-oriented architecture (SOA)** leads to **Web 2.0 services**.
   - **Virtualization** leads to **Internet clouds**.
   - **RFID and sensors** lead to the **Internet of Things**.

Labels on the arrows of Figure 1.4:

Feature	HPC path (Clusters / MPPs)	HTC path (P2P networks)
Type of nodes	Homogeneous (same kind of computers)	Disparate (different kinds of computers)
Control	Centralized control	Distributed control
Location	Physically close	Geographically sparse
Purpose	High speed	File sharing

The difference between centralized and distributed control and between client–server and P2P organization is shown below.

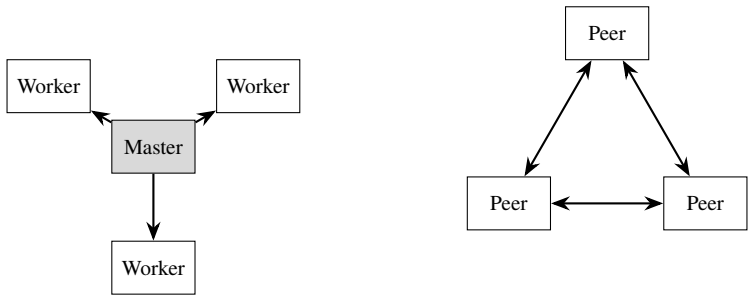


Figure 1.5: Centralized control (HPC side) versus distributed control (HTC side)

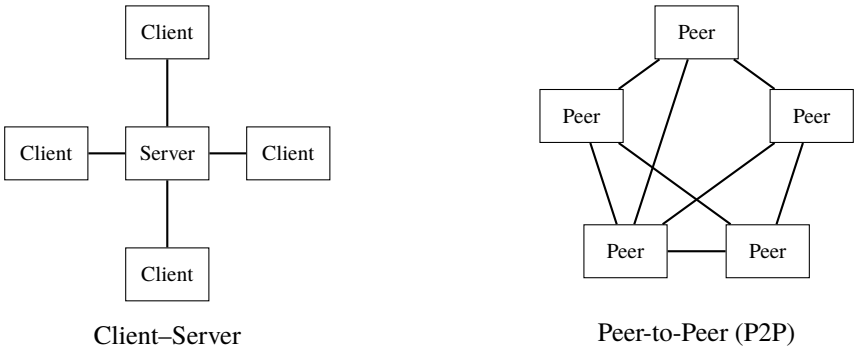


Figure 1.6: Client–server versus peer-to-peer organization

Conclusion: Both HPC and HTC paths converge into grids, and grids in turn give rise to Web 2.0 services, Internet clouds, and the IoT.

5. High-Performance Computing and High-Throughput Computing

5.1 High-Performance Computing (HPC)

Q. What is High-Performance Computing (HPC)? Explain its characteristics. **[6 Marks]**

High-Performance Computing (HPC) is the use of supercomputers and clusters to solve large and complex problems as **fast as possible**. HPC systems emphasize **raw speed performance**.

1. **Focus on speed:** For many years, HPC systems have emphasized raw speed performance, i.e., how fast a single large problem can be solved.
2. **Growth in speed:** The speed of HPC systems increased from **Gflops** (10^9 floating-point operations per second) in the early 1990s to **Pflops** (10^{15} flops) in 2010.
3. **Driving force:** This improvement was driven mainly by the demands of the **scientific, engineering, and manufacturing** communities.
4. **Measurement:** The **Top 500** most powerful computer systems in the world are ranked by their floating-point speed in **Linpack benchmark** results.
5. **Limited users:** The number of supercomputer users is limited to **less than 10%** of all computer users.
6. **Majority of users:** Today, most users use desktop computers or large servers for Internet searches and market-driven computing tasks, not supercomputers.

Examples: Weather forecasting, earthquake simulation, and genome analysis.

5.2 High-Throughput Computing (HTC)

Q. What is High-Throughput Computing (HTC)? Why is market-oriented computing shifting from HPC to HTC? **[6 Marks]**

High-Throughput Computing (HTC) is computing that emphasizes **high-flux computing**, i.e., completing the **largest number of tasks per unit of time**, rather than finishing one task as fast as possible.

1. **Strategic change:** The development of market-oriented high-end computing systems is undergoing a strategic change from an **HPC paradigm** to an **HTC paradigm**.
2. **High-flux computing:** The HTC paradigm pays more attention to high-flux computing, i.e., a very high flow of tasks through the system.
3. **Main applications:** Internet searches and web services used by **millions of users simultaneously**.
4. **Performance goal:** The performance goal shifts to **throughput**, i.e., the number of tasks completed per unit of time.
5. **Beyond speed:** HTC must improve batch processing speed and must also address the acute problems of
   - cost,
   - energy savings,
   - security, and
   - reliability

   at data and enterprise computing centers.

Example: Millions of students checking their examination results on a university website at the same time.

5.3 HPC versus HTC

Q. Differentiate between HPC and HTC systems with a neat diagram and suitable examples. **[8 Marks]**



HPC: one job split across many processors → finish fast

HTC: many independent jobs → more jobs per hour

Figure 1.7: HPC versus HTC

Sl.	Aspect	High-Performance Computing	High-Throughput Computing
1	Goal	Solve one big problem as fast as possible	Complete as many tasks as possible per unit time
2	Emphasis	Raw speed performance	High-flux computing
3	Measure	Speed in flops (Linpack benchmark, Top 500)	Throughput (tasks completed per unit time)
4	Typical systems	Supercomputers, MPPs, clusters	P2P networks, clouds, web service platforms
5	Users	Scientists and engineers (<10% of users)	General Internet users (millions)
6	Driven by	Scientific, engineering, and manufacturing communities	Market-oriented, business computing
7	Other concerns	Mainly speed	Cost, energy saving, security, reliability
8	Examples	Weather forecasting, simulations	Web search, e-commerce, social networks

Analogy: HPC is like a racing car that carries one passenger very fast; HTC is like a metro train that carries thousands of passengers per hour.

6. Three New Computing Paradigms

Q. Explain the three new computing paradigms. How has the meaning of the term "computer" changed over the years? **[8 Marks]**

As shown in Figure 1.4 (Text Book 1, Fig. 1.1), three new technologies have given rise to three new computing paradigms.

1. SOA → Web 2.0 services

With the introduction of **Service-Oriented Architecture (SOA)**, **Web 2.0 services** became available. In SOA, software is built as a collection of independent services, each doing one job, that communicate over a network using standard interfaces. A service provider publishes its service in a registry, and a service consumer finds and uses it.

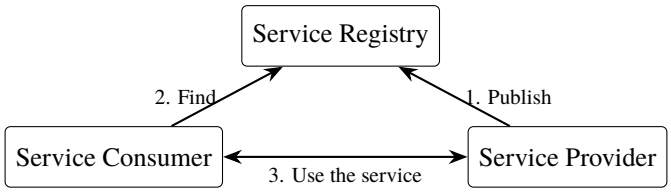


Figure 1.8: Service-Oriented Architecture (SOA)

2. Virtualization → Internet clouds

Advances in **virtualization** made it possible to see the growth of **Internet clouds** as a new computing paradigm. Virtualization creates several virtual machines (VMs) on one physical computer using a **hypervisor**; each VM behaves like a separate computer with its own operating system.

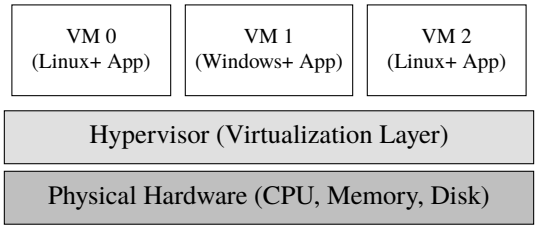


Figure 1.9: Virtualization — several virtual machines on one physical machine

3. RFID, GPS, and sensors → Internet of Things

The maturity of **Radio-Frequency Identification (RFID)**, the **Global Positioning System (GPS)**, and **sensor** technologies has triggered the development of the **Internet of Things (IoT)**.

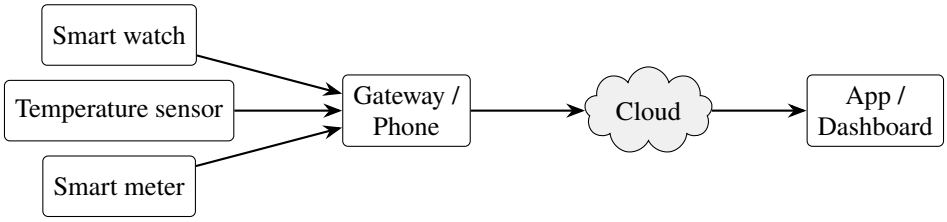


Figure 1.10: Internet of Things — everyday objects connected through the cloud

How the idea of a "computer" has changed:

1. **1969 — Leonard Kleinrock (UCLA):** When the Internet was introduced, he predicted that computer networks would grow into **computer utilities** that, like electric and telephone utilities, would serve individual homes and offices.
2. **1984 — John Gage (Sun Microsystems):** "**The network is the computer.**"
3. **2008 — David Patterson (UC Berkeley):** "**The data center is the computer.**" He noted the dramatic difference between developing software for millions to use as a service and distributing software to run on PCs.
4. **Recently — Rajkumar Buyya (University of Melbourne):** "**The cloud is the computer.**"

Blurring of paradigms: The differences among clusters, grids, P2P systems, and clouds may blur in the future. Some view clouds as grids or clusters with modest changes through virtualization; others feel the changes could be major, since clouds are expected to process huge data sets generated by the Internet, social networks, and the IoT.

7. Computing Paradigm Distinctions

Q. Distinguish between centralized, parallel, distributed, and cloud computing with neat diagrams. **[8 Marks]**

In general, **distributed computing is the opposite of centralized computing**. Parallel computing overlaps with distributed computing to a great extent, and cloud computing overlaps with distributed, centralized, and parallel computing.

1. Centralized computing

- All computer resources are centralized in **one physical system**.
- All resources (processors, memory, and storage) are **fully shared** and **tightly coupled** within **one integrated OS**.
- Many data centers and supercomputers are centralized systems, but they are used in parallel, distributed, and cloud computing applications.

2. Parallel computing

- All processors are either **tightly coupled** with **centralized shared memory** or **loosely coupled** with **distributed memory**.
- Some authors refer to this discipline as **parallel processing**.

- Interprocessor communication is accomplished through **shared memory** or via **message passing**.
- A system capable of parallel computing is a **parallel computer**; programs running on it are **parallel programs**; writing them is **parallel programming**.



Shared memory (tightly coupled) Distributed memory (loosely coupled)

Figure 1.11: Parallel computing with shared memory and with distributed memory

3. Distributed computing

- It is the field of computer science and engineering that studies **distributed systems**.
- A distributed system consists of **multiple autonomous computers**, each having its **own private memory**, communicating through a computer network.
- Information exchange is accomplished through **message passing**.
- A program running in a distributed system is a **distributed program**; writing it is **distributed programming**.

4. Cloud computing

- An Internet cloud of resources can be either a **centralized** or a **distributed** computing system.
- The cloud applies **parallel or distributed computing, or both**.
- Clouds can be built with **physical or virtualized resources** over large data centers that are centralized or distributed.
- Some authors consider cloud computing to be a form of **utility computing** or **service computing**.

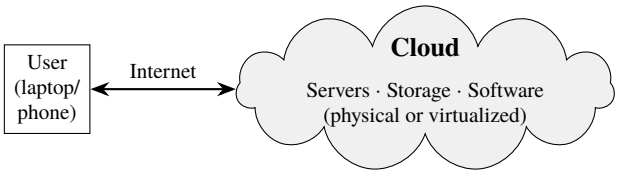


Figure 1.12: Cloud computing

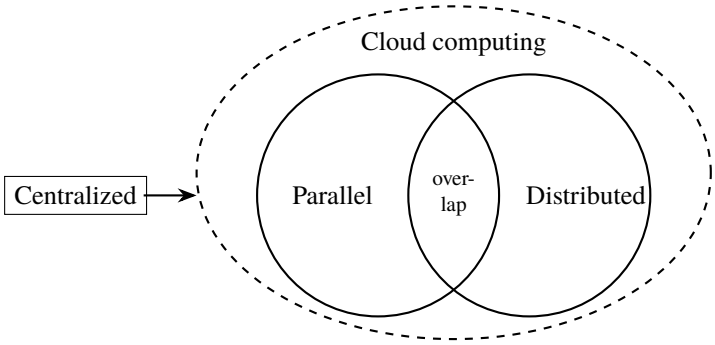


Figure 1.13: Overlap of the computing paradigms

Comparison at a glance:

Paradigm	Memory	Communication	Everyday Analogy
Centralized	One system, fully shared	Within one OS	One chef cooking in one kitchen
Parallel	Shared or distributed	Shared memory / message passing	Many chefs in one kitchen cooking one meal
Distributed	Private memory per computer	Message passing over a network	Branches of a restaurant chain coordinating by phone
Cloud	Centralized or distributed	Over the Internet	Ordering food online without seeing the kitchen

7.1 Related Terms

Q. Write short notes on: (a) Concurrent computing (b) Ubiquitous computing (c) Internet of Things (d) Internet computing. **[6 Marks]**

(a) **Concurrent computing / concurrent programming:** An alternative term preferred by some in the high-tech community. It typically refers to the **union of parallel computing and distributed computing**.

(b) **Ubiquitous computing:** Computing with **pervasive devices** at **any place and any time** using wired or wireless communication. Example: smartphones and smart watches used everywhere.

(c) **Internet of Things (IoT):** A networked connection of **everyday objects**, including computers, sensors, and humans. The IoT is supported by **Internet clouds** to achieve ubiquitous computing with any object at any place and time (see Figure 1.10).

(d) **Internet computing:** The broadest term; it covers **all computing paradigms over the Internet**.

8. Distributed System Families and Design Objectives

Q. Explain distributed system families. What are the design objectives of future HPC and HTC systems? **[8 Marks]**

Distributed system families:

1. **Grids:** Since the mid-1990s, technologies for building P2P networks and networks of clusters have been consolidated into many national projects that establish **wide area computing infrastructures**, known as **computational grids** or **data grids**.
2. **Internet clouds:** There is a surge of interest in using Internet cloud resources for **data-intensive applications**. Clouds result from moving desktop computing to **service-oriented computing** using server clusters and huge databases at data centers.
3. **Resource sharing:** Grids and clouds are disparity systems that place great emphasis on **resource sharing** in hardware, software, and data sets.
4. **Massively distributed systems:** These systems exploit a high **degree of parallelism or concurrency** among many machines.

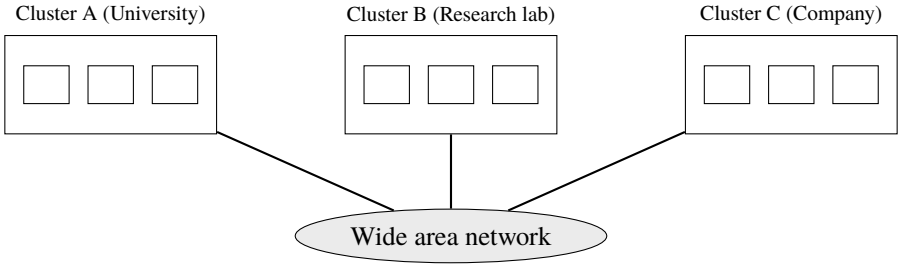


Figure 1.14: A grid joins clusters owned by different organizations over a wide area network

Scale of these systems (as given in the textbook):

- In October 2010, the highest-performing cluster machine, **Tianhe-1A** (China), had **86,016 CPU cores** and **3,211,264 GPU cores**.
- The largest computational grid connects up to **hundreds of server clusters**.
- A typical P2P network may involve **millions of client machines** working simultaneously.
- Experimental cloud computing clusters have been built with **thousands of processing nodes**.

Need for new design: Future HPC and HTC systems will demand **multicore or many-core processors** that handle large numbers of computing threads per core. They must satisfy huge demand in terms of **throughput, efficiency, scalability, and reliability**. System efficiency is decided by **speed, programming, and energy** factors (i.e., **throughput per watt** of energy consumed).

Design objectives:

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

9. Scalable Computing Trends

Q. Explain the scalable computing trends. State Moore's law and Gilder's law. **[6 Marks]**

Several predictable trends in technology drive computing applications. Designers and programmers want to predict the technological capabilities of future systems.

1. **Rules of thumb:** Jim Gray's paper "*Rules of Thumb in Data Engineering*" is an excellent example of how technology affects applications and vice versa.
2. **Moore's law:** Processor speed **doubles every 18 months**. Moore's law has been valid over the last 30 years, but it is difficult to say whether it will continue to be true in the future.
3. **Gilder's law:** Network bandwidth has **doubled each year** in the past.
4. **Commodity hardware:** The tremendous **price/performance ratio** of commodity hardware was driven by the desktop, notebook, and tablet computing markets.
5. **Use in large-scale computing:** This has also driven the adoption of commodity technologies in large-scale computing (clusters and data centers built from ordinary components).
6. **Distributed systems:** Distributed systems emphasize both **resource distribution** and **concurrency**, i.e., a high **Degree of Parallelism (DoP)**.

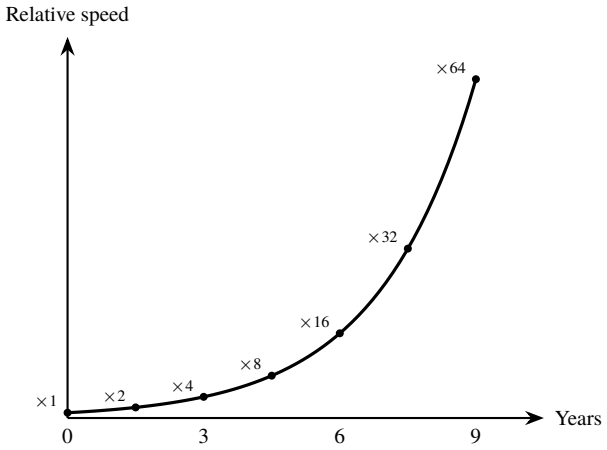


Figure 1.15: Moore's law — doubling every 1.5 years gives 64 times the speed in 9 years

10. Degrees of Parallelism

Q. Explain the different degrees of parallelism with neat diagrams. **[8 Marks]**

The **Degree of Parallelism (DoP)** is the number of operations or tasks that can be executed at the same time. Parallelism has evolved from fine-grain (bits, instructions) to coarse-grain (tasks, jobs).

1. Bit-Level Parallelism (BLP)

- Fifty years ago, when hardware was bulky and expensive, most computers were designed in a **bit-serial** fashion.
- BLP gradually converts bit-serial processing to **word-level** processing.
- Users graduated from 4-bit microprocessors to 8-, 16-, 32-, and 64-bit CPUs.

2. Instruction-Level Parallelism (ILP)

- The processor executes **multiple instructions simultaneously** rather than one at a time.
- ILP is practised through **pipelining**, **superscalar** computing, **VLIW** (very long instruction word) architectures, and **multithreading**.
- ILP requires **branch prediction**, **dynamic scheduling**, **speculation**, and **compiler support** to work efficiently.

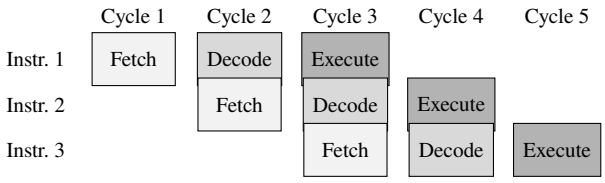


Figure 1.16: Pipelining (ILP) — in cycle 3, three instructions are processed at once

3. Data-Level Parallelism (DLP)

- One instruction operates on **many data items** at the same time.
- DLP was made popular through **SIMD** (single instruction, multiple data) and **vector machines** using vector or array instructions.
- DLP requires even more hardware support and compiler assistance to work properly.

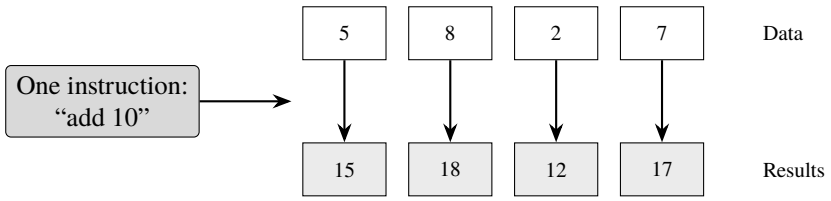


Figure 1.17: SIMD (DLP) — one instruction processes four data items at the same time

4. Task-Level Parallelism (TLP)

- Explored since the introduction of **multicore processors** and **chip multiprocessors (CMPs)**.
- Different tasks of a program run on different cores at the same time.
- TLP is **far from being very successful** due to the difficulty of programming and compiling code for efficient execution on multicore CMPs.

5. Job-Level Parallelism (JLP)

- As we move from parallel processing to **distributed processing**, computing granularity increases to job-level parallelism.
- Entire independent jobs run on different computers at the same time.

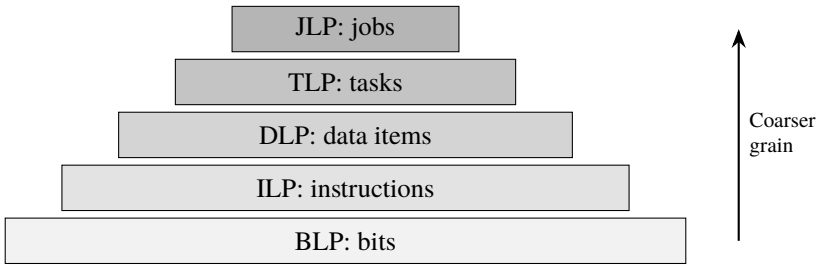


Figure 1.18: Granularity of parallelism — coarse-grain parallelism is built on top of fine-grain parallelism

Conclusion: A modern processor explores all these types of parallelism. BLP, ILP, and DLP are well supported by advances in hardware and compilers. It is fair to say that **coarse-grain parallelism is built on top of fine-grain parallelism**.

11. Innovative Applications

Q. Explain the innovative applications of HPC and HTC systems. What are their common requirements? **[8 Marks]**

Both HPC and HTC systems desire **transparency** in many application aspects. The following should be made transparent (hidden) from both users and system management:

Type of Transparency	The user does not need to know…
Data access	where or how the data is stored
Resource allocation	which machines are given to the job
Process location	on which computer the program is running
Concurrency in execution	that many other users are running jobs at the same time
Job replication	that copies of the job are running for safety
Failure recovery	that a machine failed and the system recovered

Applications of HPC and HTC systems (Table 1.1 of the textbook):

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

Common requirements:

1. Almost all applications demand **computing economics**, **web-scale data collection**, **system reliability**, and **scalable performance**.
2. **Distributed transaction processing** is often practised in the banking and finance industry. Transactions represent **90 percent** of the existing market for reliable banking systems.
3. Users must deal with **multiple database servers** in distributed transactions.
4. Maintaining the **consistency of replicated transaction records** is crucial in real-time banking services.
5. Other complications include **lack of software support**, **network saturation**, and **security threats**.

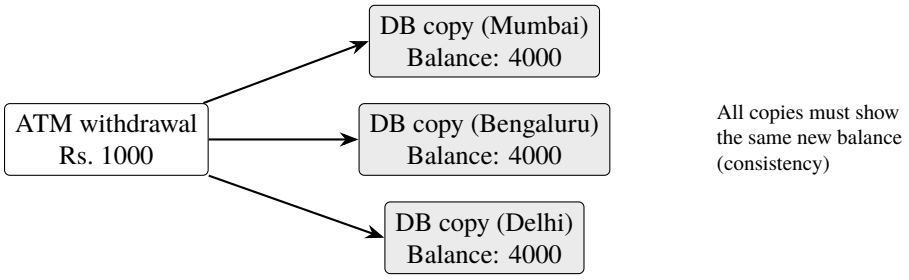


Figure 1.19: Consistency of replicated transaction records in banking

12. The Trend toward Utility Computing

Q. With a neat diagram, explain the vision of computer utilities in modern distributed computing systems. What is utility computing? **[8 Marks]**

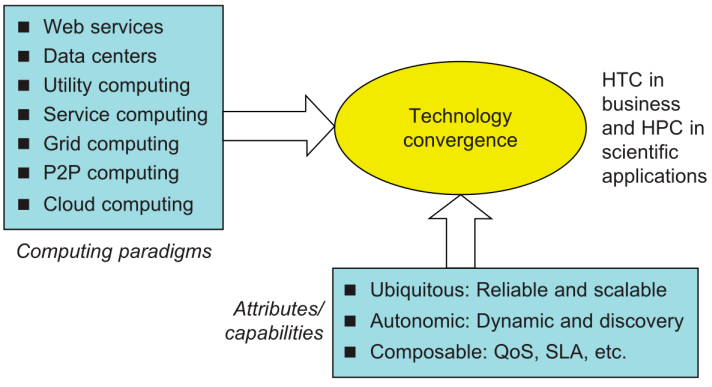


Figure 1.20: The vision of computer utilities in modern distributed computing systems (Text Book 1, Fig. 1.2)

Explanation of the figure:

1. **Computing paradigms (left box):** Web services, data centers, utility computing, service computing, grid computing, P2P computing, and cloud computing are the major paradigms used to study distributed systems.
2. **Attributes / capabilities (bottom box):** These paradigms share three common characteristics:
   - **Ubiquitous:** They are present everywhere in daily life. **Reliability** and **scalability** are the two major design objectives.
   - **Autonomic:** They aim at autonomic operations that can be **self-organized** to support **dynamic discovery**.
   - **Composable:** They are composable with **QoS** (Quality of Service) and **SLAs** (Service-Level Agreements).
3. **Technology convergence (centre):** The paradigms and their attributes converge together.
4. **Result (right):** This convergence supports **HTC in business** and **HPC in scientific applications**, and realizes the **computer utility vision**.

Utility computing focuses on a business model in which customers receive computing resources from a **paid service provider**, and pay according to usage, just like electricity or water.

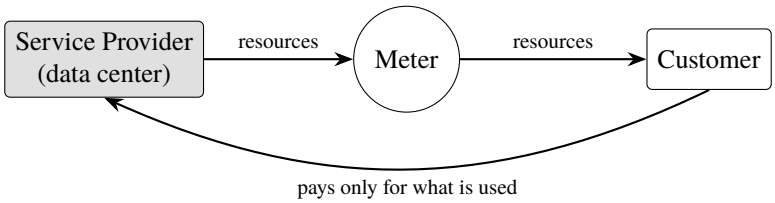


Figure 1.21: Utility computing — pay-per-use model

Points on utility computing:

1. All grid and cloud platforms are regarded as **utility service providers**.
2. Cloud computing offers a **broader concept** than utility computing.
3. Distributed cloud applications run on any available servers in some **edge networks**.
4. **Technological challenges** include all aspects of computer science and engineering. Users demand:
   - new network-efficient processors,
   - scalable memory and storage schemes,
   - distributed OSes,
   - middleware for machine virtualization,
   - new programming models,
   - effective resource management, and
   - application program development.
5. These hardware and software supports are necessary to build distributed systems that explore **massive parallelism at all processing levels**.

Analogy: We pay the electricity bill for the units consumed; we do not build a power plant at home.

13. The Hype Cycle of New Technologies

Q. What is a hype cycle? With a neat diagram, explain the five stages of the hype cycle of new technologies. **[8 Marks]**

A **hype cycle** is a graphical representation (by the research firm Gartner) of the expectations for an emerging technology at **five different stages**, from its introduction until its mainstream adoption.

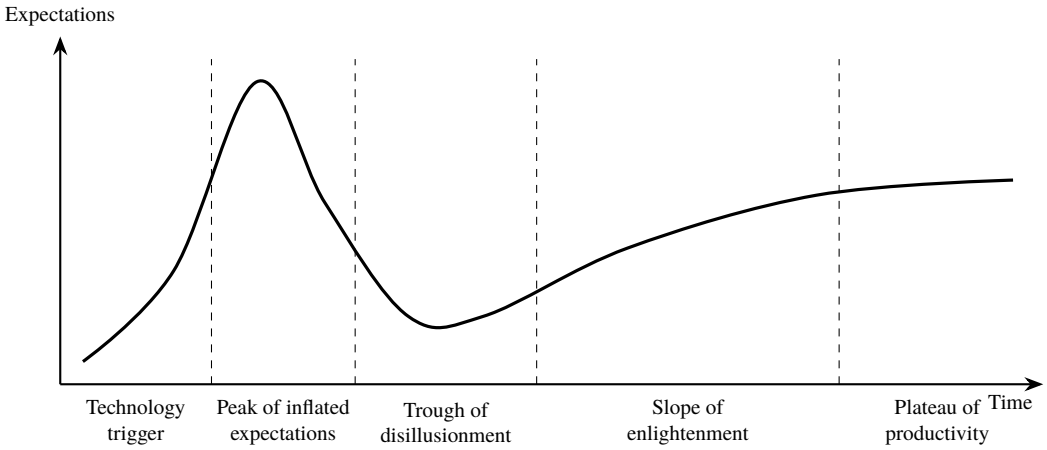


Figure 1.22: The five stages of the hype cycle

Five stages:

1. **Technology trigger:** A new technology is introduced and expectations begin to rise.
2. **Peak of inflated expectations:** Expectations rise sharply from the trigger period to a high peak, often beyond what the technology can deliver.
3. **Trough of disillusionment:** Through a short period of disillusionment, expectations drop to a valley.
4. **Slope of enlightenment:** Expectations then increase steadily over a long enlightenment period as the real uses are understood.
5. **Plateau of productivity:** The technology reaches mainstream adoption and is productively used.

Once a technology begins to climb the slope of enlightenment, it may reach the productivity plateau within **two to five years**.

Hype cycle for emerging technologies, 2010:

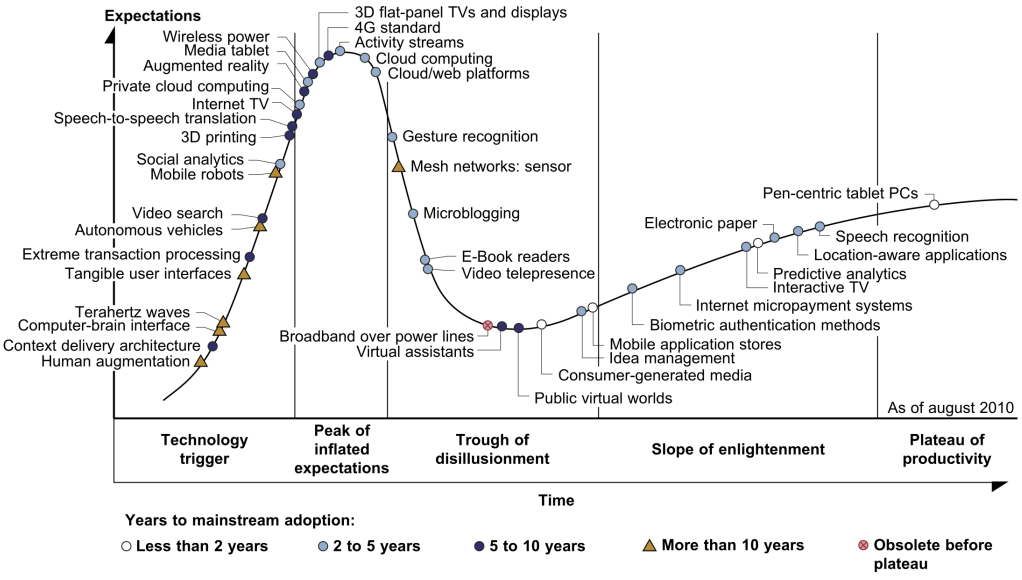


Figure 1.23: Hype cycle for emerging technologies, 2010 (Text Book 1, Fig. 1.3; source: Gartner, Inc., 2010)

Symbols used (years to mainstream adoption):

Symbol	Years to Mainstream	Example (as of August 2010)
Hollow circle	Less than 2 years	Consumer-generated media (in the trough of disillusionment)
Gray circle	2 to 5 years	Internet micropayment systems (enlightenment to maturity); cloud computing
Solid circle	5 to 10 years	3D printing (rising expectation stage)
Triangle	More than 10 years	Mesh network sensors (inflated expectation stage)
Crossed circle	Obsolete before plateau	Broadband over power lines

Observations from the 2010 hype cycle:

- **Cloud technology** had just crossed the peak of expectations in 2010 and was expected to take **two to five years** to reach the productivity stage.
- **Broadband over power lines** was expected to become obsolete before leaving the trough of disillusionment.
- Many technologies (dark circles) were at their peak in August 2010 and were expected to take five to ten years to reach the plateau.
- Promising technologies included clouds, biometric authentication, interactive TV, speech recognition, predictive analytics, and media tablets.

14. Question Bank

Sl.	Question	Marks	Section
1	What is scalable computing over the Internet? Explain how computing has changed from centralized to parallel and distributed computing.	6	1
2	Explain the age of Internet computing. Why is the Linpack benchmark no longer optimal?	6	2
3	Explain the platform evolution of computer technology through five generations with examples.	6	3
4	With a neat diagram, explain the evolutionary trend toward parallel, distributed, and cloud computing.	8	4
5	What is HPC? Explain its characteristics.	6	5.1
6	What is HTC? Why is market-oriented computing shifting from HPC to HTC?	6	5.2
7	Differentiate between HPC and HTC with a neat diagram and examples.	8	5.3
8	Explain the three new computing paradigms. How has the meaning of "computer" changed?	8	6
9	Distinguish between centralized, parallel, distributed, and cloud computing with neat diagrams.	8	7
10	Write short notes on concurrent computing, ubiquitous computing, IoT, and Internet computing.	6	7.1
11	Explain distributed system families and the design objectives of future HPC and HTC systems.	8	8
12	Explain scalable computing trends. State Moore's law and Gilder's law.	6	9
13	Explain the different degrees of parallelism with neat diagrams.	8	10
14	Explain the innovative applications of HPC and HTC systems and their common requirements.	8	11
15	With a neat diagram, explain the vision of computer utilities. What is utility computing?	8	12
16	What is a hype cycle? Explain its five stages with a neat diagram.	8	13
17	What is transparency in distributed systems? List the aspects that must be made transparent.	6	11
18	Explain the design objectives: efficiency, dependability, adaptation, and flexibility.	6	8

Module 1: Modern Computing Approaches

Topic 2: Parallel and Distributed Computing

Text Book 2: Rajkumar Buyya, Christian Vecchiola and S. Thamarai Selvi, *Mastering Cloud Computing*, McGraw Hill Education, 2013 — Sections 2.1, 2.2, 2.3.1, 2.3.2, 2.3.4, 2.4.1, 2.4.2

1. Eras of Computing

Q. With a neat diagram, explain the eras of computing. What are the four key elements of computing developed in each era? **[8 Marks]**

Introduction: Cloud computing supports better utilization of IT infrastructure, services, and applications through a **pay-per-use** service delivery model. The principles of parallel and distributed computing are the foundation for building cloud systems.

1. **Two models of computing:** The two fundamental and dominant models of computing are **sequential** and **parallel**. In sequential computing, instructions are executed one after another on one processor; in parallel computing, many instructions are executed at the same time on multiple processors.
2. **Sequential era:** The sequential computing era began in the **1940s**.
3. **Parallel era:** The parallel (and distributed) computing era followed it **within a decade**.
4. **Four key elements:** In both eras, four key elements of computing were developed:
   - **Architectures** — the hardware design of computers;
   - **Compilers** — system software, including compilers and operating systems;
   - **Applications** — programs that solve users' problems;
   - **Problem-solving environments** — tools that make it easy for engineers to build solutions.
5. **Order of development:** Each era started with a development in **hardware architectures**, which enabled the creation of **system software** (compilers and operating systems). System software supports the management of the hardware and the development of **applications**.
6. **Maturity:** The era reached **consolidation** when **problem-solving environments** were designed to facilitate and empower engineers. At this point the paradigm achieved maturity and became **mainstream**.
7. **Three-phase process:** Every aspect of each era went through three phases: **research and development (R&D)**, **commercialization**, and **commoditization**.

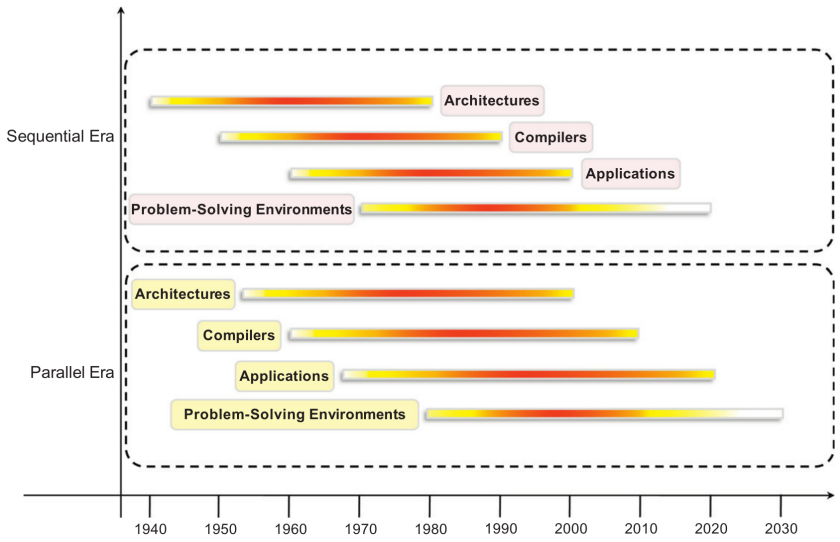


Figure 2.1: Eras of computing, 1940s–2030s (Text Book 2, Fig. 2.1)

Simplified diagram for the exam:

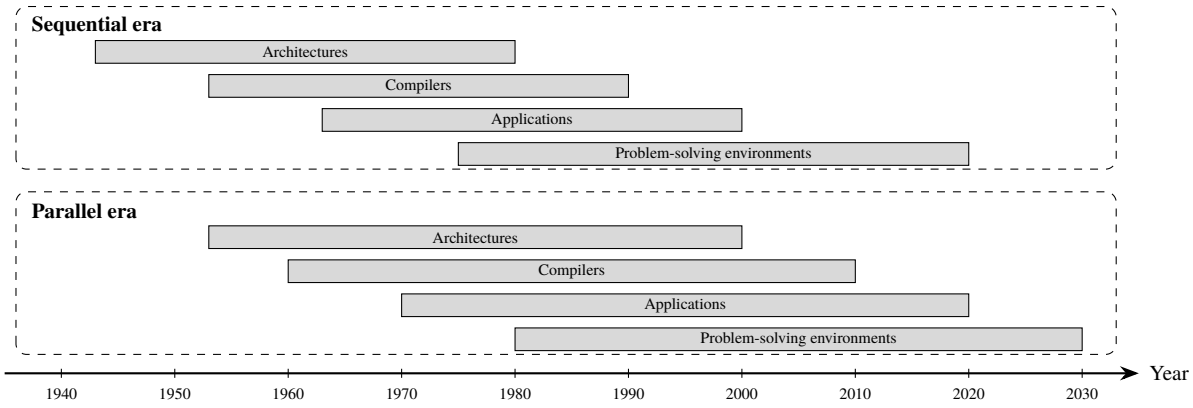


Figure 2.2: Eras of computing — approximate periods of the four elements, redrawn from Text Book 2, Fig. 2.1

Explanation of the figure: The upper box shows the sequential era and the lower box shows the parallel era. Each bar shows the period in which one element was developed. In both eras the order is the same — architectures, then compilers, then applications, then problem-solving environments — and the parallel era starts about a decade later and extends to the 2030s.

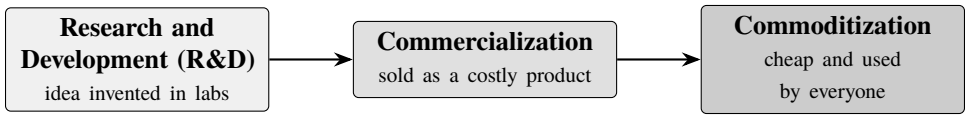


Figure 2.3: The three-phase process followed by every era

Example of the three phases: Multicore processors were first research projects, were then sold in expensive servers, and are now found in every mobile phone.

Key Points to Remember

- Two eras: sequential (1940s) and parallel (a decade later).
- Four elements: architectures, compilers, applications, problem-solving environments.
- Three phases: R&D → commercialization → commoditization.

2. Parallel vs. Distributed Computing

Q. Differentiate between parallel computing and distributed computing. **[6 Marks]**

The terms **parallel computing** and **distributed computing** are often used **interchangeably**, even though they mean slightly different things. The term *parallel* implies a **tightly coupled** system, whereas *distributed* refers to a **wider class** of systems, including those that are tightly coupled.

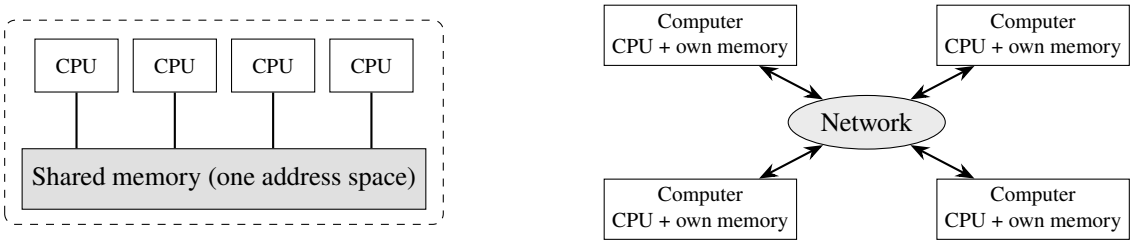
Parallel computing is a model in which the computation is divided among several processors **sharing the same memory**.

1. **Homogeneity:** A parallel computing system is characterized by the homogeneity of its components: each processor is of the **same type** and has the same capability.
2. **Shared memory:** The shared memory has a **single address space**, which is accessible to all the processors.
3. **Parallel programs:** They are broken down into several **units of execution** that are allocated to different processors and communicate by means of the shared memory.

4. **Broader meaning today:** Originally, only machines with multiple processors sharing the same physical memory were called parallel. Today, all architectures based on the concept of shared memory are parallel, whether the shared memory is physically present or created with libraries, special hardware, and fast networks. For example, a **cluster** whose nodes are connected through an **InfiniBand** network and configured with a **distributed shared memory** system can be considered a parallel system.

Distributed computing encompasses any architecture or system that allows the computation to be **broken down into units** and executed **concurrently** on different computing elements.

5. **Computing elements:** These may be processors on different nodes, processors on the same computer, or cores within the same processor.
6. **More general term:** Distributed computing includes a wider range of systems and applications than parallel computing.
7. **Location and heterogeneity:** The term usually implies that the computing elements are at **different locations** and may be **heterogeneous** in hardware and software.
8. **Examples:** Computing **grids** and **Internet computing systems**, which combine the biggest variety of architectures, systems, and applications in the world.



Parallel: one machine, tightly coupled

Distributed: many computers, messages

Figure 2.4: Parallel computing (shared memory) versus distributed computing (messages over a network)

Sl.	Aspect	Parallel Computing	Distributed Computing
1	Coupling	Tightly coupled	Wider class; mostly loosely coupled
2	Memory	Shared memory, single address space	Usually private memory per computer
3	Communication	Through the shared memory	By passing messages over a network
4	Components	Homogeneous	Often heterogeneous
5	Location	Usually one place	Often different locations
6	Scope of term	Narrower	More general
7	Examples	Multicore computer; InfiniBand cluster with DSM	Grids, Internet computing systems

Analogy: Parallel computing is like students solving a problem together on one shared whiteboard; distributed computing is like students in different cities working in their own notebooks and sharing answers by messages.

3. Elements of Parallel Computing: Parallel Processing

3.1 What Is Parallel Processing?

Q. What is parallel processing? Why is parallel computing needed? **[6 Marks]**

Need for parallel computing: Silicon-based processor chips are reaching their **physical limits**. Processing speed is constrained by the **speed of light**, and the density of transistors in a processor is constrained by **thermodynamic limitations**. A viable solution is to connect **multiple processors** working in coordination to solve

"**Grand Challenge**" problems. This led to parallel computing, which covers techniques, architectures, and systems for performing multiple activities in parallel. Here, parallel computing refers to parallelism **within a single computer** by coordinating the activity of multiple processors.

Parallel processing is the processing of **multiple tasks simultaneously** on **multiple processors**.

1. **Parallel program:** It consists of multiple active processes (tasks) simultaneously solving a given problem.
2. **Divide and conquer:** A given task is divided into multiple subtasks using the **divide-and-conquer** technique, and each subtask is processed on a different CPU.
3. **Parallel programming:** Programming a multiprocessor system using the divide-and-conquer technique is called parallel programming.
4. **Need for more power:** Many applications today require more computing power than a traditional sequential computer can offer.
5. **Cost-effective solution:** Parallel processing increases the number of CPUs in a computer and adds an efficient **communication system** between them. The workload is shared between processors, giving higher computing power and performance than a single-processor system.

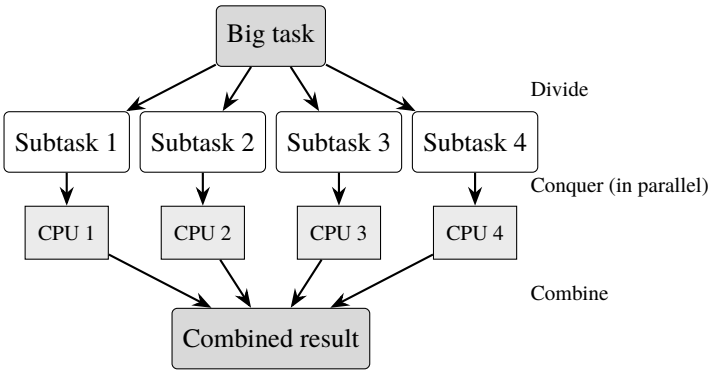


Figure 2.5: Parallel processing using the divide-and-conquer technique

Example: 1,200 answer scripts are divided among 12 teachers; each evaluates 100 scripts, and the work is finished 12 times sooner.

3.2 Factors Influencing Parallel Processing

Q. Explain the factors that influence the development of parallel processing. **[6 Marks]**

1. **Increasing computational requirements:** Computational needs keep increasing in both scientific and business computing. Technical problems that need high-speed computing power arise in life sciences, aerospace, geographical information systems, and mechanical design and analysis.
2. **Physical limits of sequential architectures:** Sequential architectures are constrained by the speed of light and the laws of thermodynamics. The speed of sequential CPUs is reaching a **saturation point** (no more **vertical growth**), so the alternative is to connect multiple CPUs (**horizontal growth**).
3. **Limits of hardware improvements:** Improvements such as pipelining and superscalar execution are **non-scalable** and require sophisticated compiler technology, which is difficult to develop.
4. **Limits of vector processing:** Vector processing works well for certain problems — scientific problems with many matrix operations, and graphical processing — but is not useful for other areas such as **databases**.
5. **Mature technology:** Parallel processing technology is mature and can be exploited **commercially**; significant R&D work on development tools and environments already exists.
6. **Networking:** Significant development in networking technology is paving the way for **heterogeneous computing**.

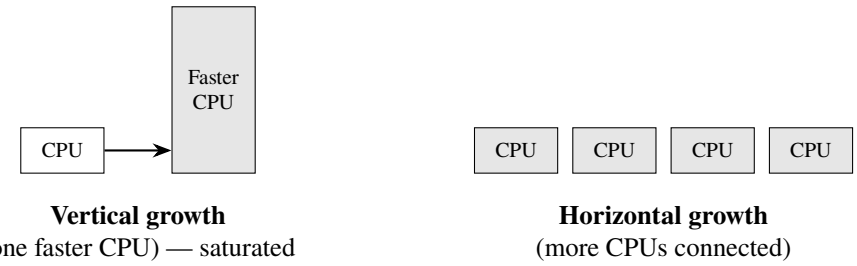


Figure 2.6: Vertical growth versus horizontal growth

4. Hardware Architectures for Parallel Processing

Q. Explain the hardware architectures for parallel processing (SISD, SIMD, MISD, and MIMD) with neat diagrams. **[8 Marks]**

The core elements of parallel processing are **CPUs**. Based on the number of **instruction streams** and **data streams** that can be processed simultaneously, computing systems are classified into four categories (commonly called **Flynn's classification**):

	Single data	**Multiple data**
Single instruction	**SISD** sequential computer	**SIMD** vector machines
Multiple instruction	**MISD** not commercial	**MIMD** multicore, clusters

Figure 2.7: Classification of computers by instruction and data streams

1. Single-instruction, single-data (SISD) systems

- An SISD system is a **uniprocessor** machine that executes a single instruction operating on a single data stream.
- Machine instructions are processed **sequentially**; hence these computers are called **sequential computers**.
- Most conventional computers are built using the SISD model.
- All instructions and data must be stored in **primary memory**.
- The speed of the processing element is limited by the rate at which the computer can transfer information internally.
- **Examples:** IBM PC, Macintosh, and workstations.

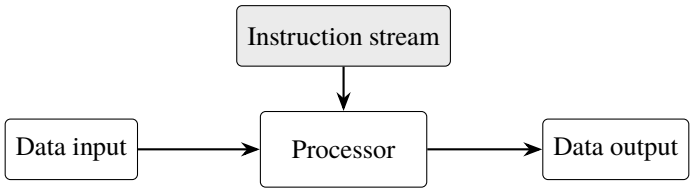


Figure 2.8: SISD architecture (redrawn from Text Book 2, Fig. 2.2)

2. Single-instruction, multiple-data (SIMD) systems

- An SIMD system is a **multiprocessor** machine that executes the **same instruction** on all the CPUs, each operating on a **different data stream**.
- SIMD machines are well suited to **scientific computing**, which involves many vector and matrix operations.

- **Example:** The statement $C_i = A_i \times B_i$ is passed to all the processing elements (PEs). The elements of vectors A and B are divided into N sets (for N PEs), and each PE processes one set.
- **Examples of machines:** Cray's vector processing machine and Thinking Machines' CM*.

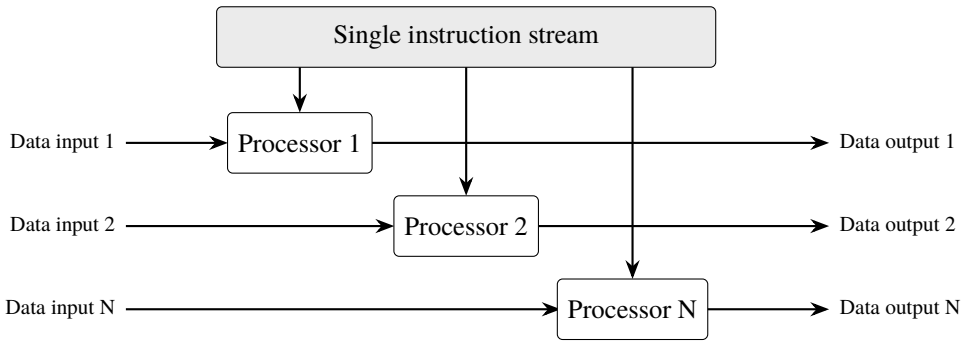


Figure 2.9: SIMD architecture (redrawn from Text Book 2, Fig. 2.3)

3. Multiple-instruction, single-data (MISD) systems

- An MISD system is a multiprocessor machine that executes **different instructions** on different PEs, all operating on the **same data set**.
- **Example:** $y = \sin(x) + \cos(x) + \tan(x)$ performs different operations on the same data x.
- MISD machines are not useful in most applications. A few were built, but none is available commercially; they are more an **intellectual exercise** than a practical configuration.

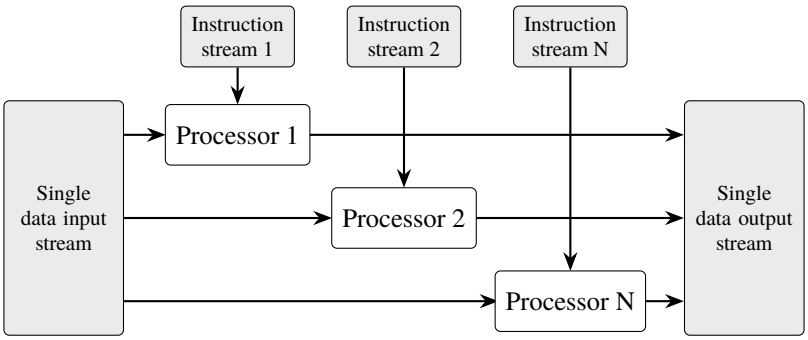


Figure 2.10: MISD architecture (redrawn from Text Book 2, Fig. 2.4)

4. Multiple-instruction, multiple-data (MIMD) systems

- An MIMD system is a multiprocessor machine that executes **multiple instructions on multiple data sets**.
- Each PE has **separate instruction and data streams**; hence MIMD machines are well suited to **any kind of application**.
- Unlike SIMD and MISD machines, PEs in MIMD machines work **asynchronously**.
- MIMD machines are broadly classified into **shared-memory MIMD** and **distributed-memory MIMD** (Section 5).

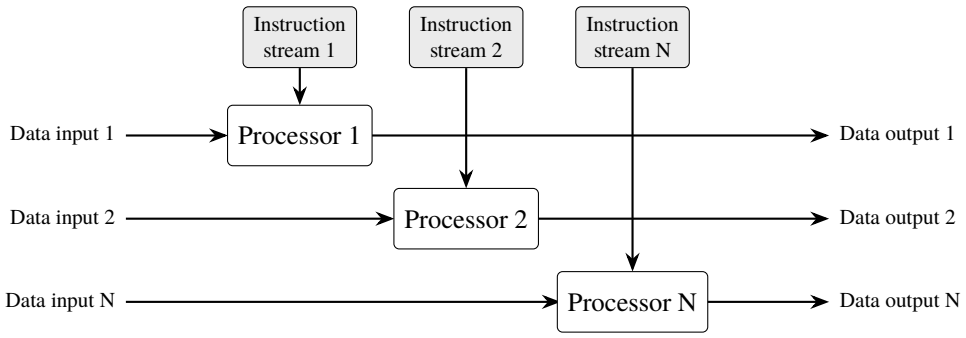


Figure 2.11: MIMD architecture (redrawn from Text Book 2, Fig. 2.5)

Comparison at a glance:

Type	Meaning	Examples	Everyday Analogy
SISD	One instruction, one data stream	IBM PC, Macintosh, workstations	One cook, one recipe, one dish
SIMD	Same instruction, many data streams	Cray vector machine, Thinking Machines' CM*	One PT command followed by all students
MISD	Many instructions, same data	None commercially available	One script checked by three examiners for different things
MIMD	Many instructions, many data streams	Silicon Graphics machines, Sun/IBM SMP	Many cooks, many recipes, many dishes

Conclusion: SISD is sequential, while SIMD, MISD, and MIMD are multiprocessor architectures. MIMD is the most general-purpose and is the basis of most modern multicore computers and clusters.

5. Shared-Memory and Distributed-Memory MIMD

Q. With a neat diagram, differentiate between shared-memory MIMD and distributed-memory MIMD machines.

[6 Marks]

MIMD machines are broadly categorized into shared-memory MIMD and distributed-memory MIMD, based on the way the PEs are coupled to the main memory.

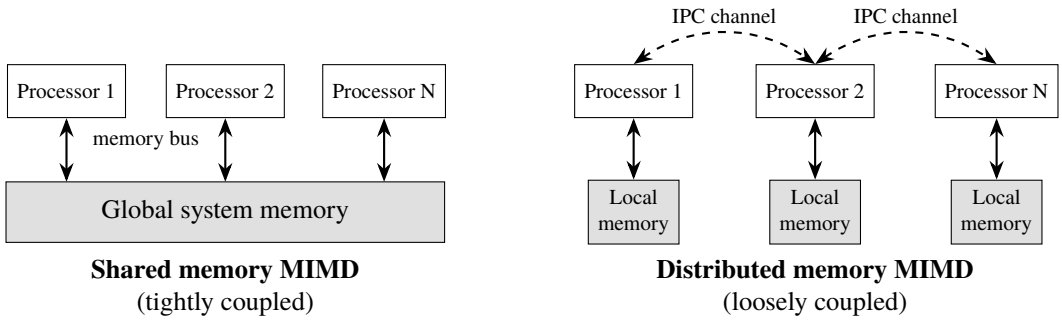


Figure 2.12: Shared-memory (left) and distributed-memory (right) MIMD architecture (redrawn from Text Book 2, Fig. 2.6)

Shared-memory MIMD machines

- All the PEs are connected to a **single global memory** and all have access to it.
- These systems are called **tightly coupled multiprocessor systems**.
- Communication between PEs takes place through the **shared memory**; a modification of data in the global memory by one PE is **visible to all** other PEs.

- **Examples:** Silicon Graphics machines and Sun/IBM SMP (Symmetric Multi-Processing).

Distributed-memory MIMD machines

- All PEs have a **local memory**.
- These systems are called **loosely coupled multiprocessor systems**.
- Communication between PEs takes place through the interconnection network, the **interprocess communication (IPC) channel**.
- The network connecting the PEs can be configured as a **tree, mesh, cube**, and so on.
- Each PE operates **asynchronously**; if communication or synchronization is necessary, PEs **exchange messages**.

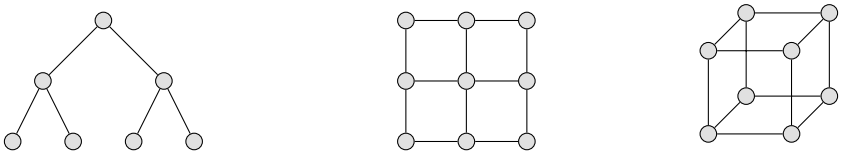


Figure 2.13: Interconnection network topologies

Sl.	Aspect	Shared-Memory MIMD	Distributed-Memory MIMD
1	Memory	One global memory	Local memory for each PE
2	Coupling	Tightly coupled	Loosely coupled
3	Communication	Through the shared memory	Messages over the IPC channel
4	Programming	Easier to program	Harder to program
5	Failures	Less tolerant; a failure affects the entire system	A failed PE can be easily isolated
6	Extending	Harder to extend	Easier to extend
7	Scalability	Less likely to scale due to **memory contention**	Scales well; each PE has its own memory
8	Popularity	Less popular	**Most popular today**

Memory contention is the situation in which many processors try to access the same memory at the same time and must wait for one another; it limits the scalability of shared-memory MIMD.

6. Levels of Parallelism

Q. Explain the levels of parallelism with a neat diagram. **[8 Marks]**

Levels of parallelism are decided by the **lumps of code (grain size)** that can be a potential candidate for parallelism.

1. **Common goal:** All approaches aim to **boost processor efficiency by hiding latency**.
2. **Hiding latency:** To conceal latency, there must be another thread ready to run whenever a lengthy operation occurs.
3. **Idea:** Execute two or more single-threaded applications concurrently, such as compiling, text formatting, database searching, and device simulation.
4. **Four levels:** Parallelism within an application can be detected at several levels:

- **Large grain** (or task level),
- **Medium grain** (or control level),
- **Fine grain** (data level),
- **Very fine grain** (multiple-instruction issue).

Grain Size	Code Item	Parallelized By
Large	Separate and heavyweight process	Programmer
Medium	Function or procedure	Programmer
Fine	Loop or instruction block	Parallelizing compiler
Very fine	Instruction	Processor

Table: Levels of parallelism (Text Book 2, Table 2.1)

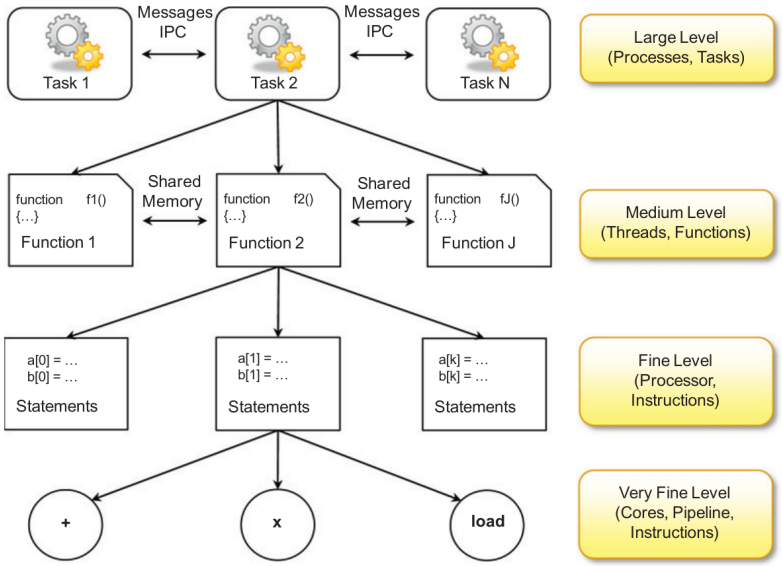


Figure 2.14: Levels of parallelism in an application (Text Book 2, Fig. 2.7)

Explanation of the figure:

1. **Large level (processes, tasks):** Separate tasks (Task 1, Task 2, …, Task N) run in parallel and communicate by **messages (IPC)**.
2. **Medium level (threads, functions):** Each task is divided into functions (f1, f2, …, fJ) that run as threads and communicate through **shared memory**.
3. **Fine level (processor, instructions):** Each function contains statements and loops, such as `a[0]`, `a[1]`, …, `a[k]`, which are processed in parallel.
4. **Very fine level (cores, pipeline, instructions):** Individual operations such as `+`, `x`, and `load` are executed in parallel inside the cores and pipeline.

The textbook considers parallelism and distribution at the **top two levels**, which distribute the computation among multiple threads or processes.

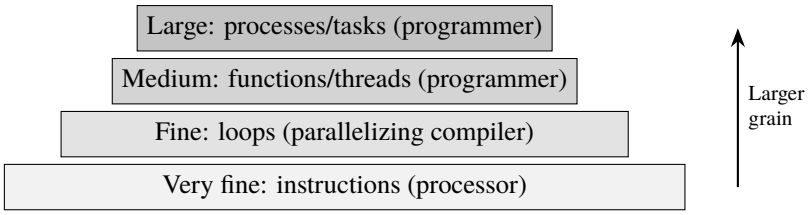


Figure 2.15: Grain sizes of parallelism and who parallelizes them

Link with Topic 1 (Evolution of Computing, Section 10 — Degrees of Parallelism): very fine grain ≈ instruction-level parallelism; fine grain ≈ data-level parallelism; medium grain ≈ task-level parallelism; large grain ≈ job-level parallelism.

7. Elements of Distributed Computing: General Concepts

Q. What is distributed computing? Define a distributed system and explain its general concepts. **[6 Marks]**

Parallel computing introduces parallelism within a single machine. Distributed computing extends these concepts so that multiple activities can be performed by systems composed of **multiple heterogeneous machines**.

Distributed computing studies the **models, architectures, and algorithms** used for building and managing distributed systems.

1. **Definition by Tanenbaum et al.:** "A distributed system is a collection of **independent computers** that appears to its users as a **single coherent system**."
2. **Scope of the definition:** It is general enough to include many types of distributed systems that focus on unified usage and **aggregation of distributed resources**.
3. **Architectural focus:** The architectural models are used to harness independent computers and present them as a whole coherent system.
4. **Communication:** Since distributed systems are made of more than one computer that collaborate, data and information must be exchanged between them, generally through the **network**.
5. **Definition by Coulouris et al.:** "A distributed system is one in which components located at **networked computers** communicate and coordinate their actions **only by passing messages**."
6. **Message passing:** The components of a distributed system communicate through some form of message passing, a term that encompasses several communication models.

Example: When we pay using UPI, the phone, the payment app's servers, a central switch, and two banks' computers work together, but the user sees only one simple service. They cooperate only by sending messages to each other.

8. Components of a Distributed System: The Layered View

Q. With a neat diagram, explain the components of a distributed system (layered view). **[8 Marks]**

A distributed system is the result of the interaction of several components that traverse the entire **computing stack** from hardware to software. It emerges from the collaboration of several elements that, by working together, give users the **illusion of a single coherent system**.

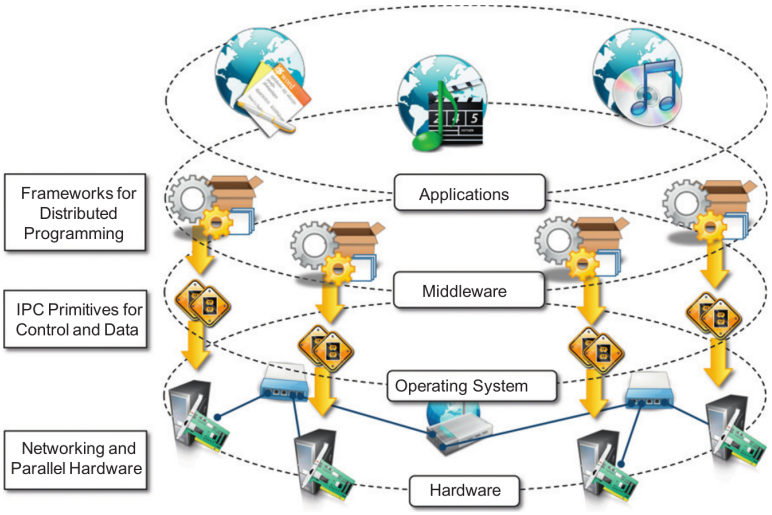


Figure 2.16: A layered view of a distributed system (Text Book 2, Fig. 2.10)

Simplified diagram for the exam:

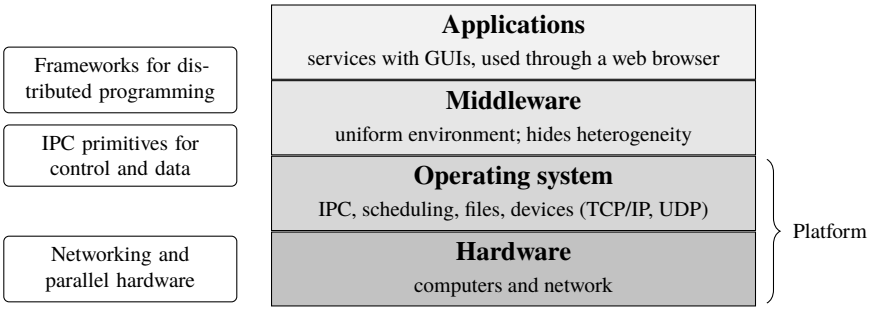


Figure 2.17: Layers of a distributed system (redrawn from Figure 2.16)

Explanation of the layers:

1. **Hardware layer:** At the bottom, computer and network hardware constitute the **physical infrastructure** (networking and parallel hardware).
2. **Operating system layer:** The hardware is directly managed by the OS, which provides the basic services for:
   - interprocess communication (IPC),
   - process scheduling and management, and
   - resource management in terms of the file system and local devices.

   Together, the hardware and OS layers form the **platform** on which specialized software is deployed to turn a set of networked computers into a distributed system.
3. **Role of standards:** Well-known standards at the OS, hardware, and network levels allow **heterogeneous** components to be harnessed into a coherent and uniform system. For example, network connectivity is controlled by standards, and IPC services are implemented on top of standard protocols such as **TCP/IP** and **UDP**.
4. **Middleware layer:** It uses the OS services to build a **uniform environment** for the development and deployment of distributed applications. It supports the programming paradigms of distributed systems and provides its own protocols, data formats, and programming languages or frameworks. It offers a uniform interface that is independent of the underlying OS and **hides the heterogeneities** of the bottom layers.
5. **Applications layer:** At the top are the applications and services designed and developed using the middleware. They often expose their features through **graphical user interfaces (GUIs)** accessible locally or through the Internet via a **web browser**.

Conclusion: Each layer uses the services of the layer below it, and together the four layers make many independent computers appear as one system.

9. A Cloud Computing Distributed System

Q. With a neat diagram, explain how the layered view of a distributed system applies to a cloud computing system. **[6 Marks]**

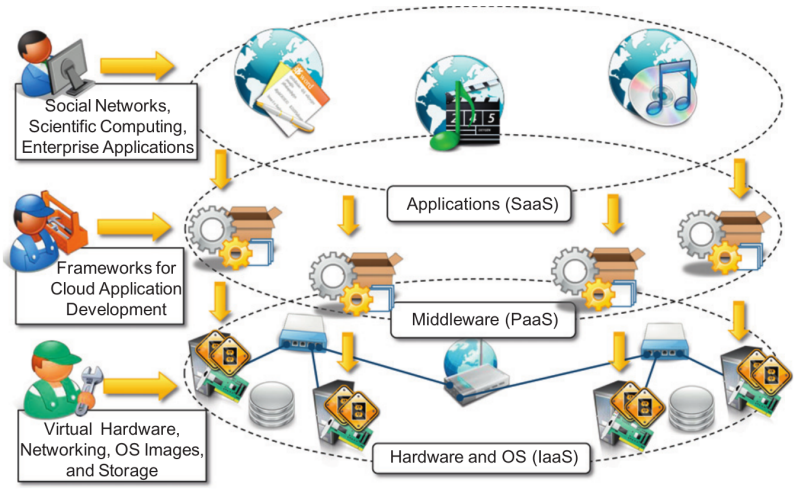


Figure 2.18: A cloud computing distributed system (Text Book 2, Fig. 2.11)

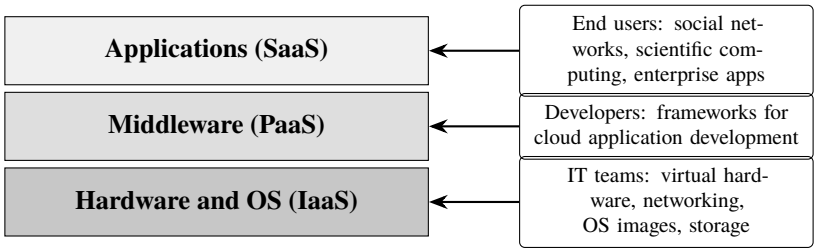


Figure 2.19: Layers of a cloud computing system and their users (redrawn from Figure 2.18)

1. **Use of web technologies:** In a cloud computing system, web technologies are strongly preferred, both to connect distributed applications with the end user and to provide platform services for building distributed systems.
2. **Hardware and OS (IaaS):** The hardware and OS layers make up the **bare-bone infrastructure** of one or more data centers, where racks of servers are connected by high-speed networks. This layer offers virtual hardware, networking, OS images, and storage.
3. **Middleware (PaaS):** The core logic is implemented in the middleware, which manages the **virtualization layer** deployed on the physical infrastructure to maximize its utilization and to provide a customizable runtime environment for applications. It offers frameworks for cloud application development.
4. **Applications (SaaS):** The top layer offers applications such as social networks, scientific computing, and enterprise applications.
5. **Web 2.0 interfaces:** The middleware provides different facilities according to the type of services sold to customers, offered through **Web 2.0-compliant interfaces**, ranging from virtual infrastructure building to application development and runtime environments.
6. **Example:** Infrastructure-as-a-Service providers such as **Amazon Web Services (AWS)** provide facilities for creating virtual machines, organizing them into a cluster, and deploying applications and systems on top.

Analogy: IaaS is like renting an empty flat, PaaS is like renting a furnished flat, and SaaS is like staying in a hotel room where everything is ready.

Key Points to Remember

- Layered view: hardware → OS → middleware → applications.
- Hardware + OS = platform; middleware hides heterogeneity.
- In the cloud: hardware and OS = IaaS, middleware = PaaS, applications = SaaS.

10. Question Bank

Sl.	Question	Marks	Section
1	With a neat diagram, explain the eras of computing. What are the four key elements of computing developed in each era?	8	1
2	Differentiate between parallel computing and distributed computing.	6	2
3	What is parallel processing? Why is parallel computing needed?	6	3.1
4	Explain the factors that influence the development of parallel processing.	6	3.2
5	Explain the hardware architectures for parallel processing (SISD, SIMD, MISD, MIMD) with neat diagrams.	8	4
6	With a neat diagram, differentiate between shared-memory and distributed-memory MIMD machines.	6	5
7	Explain the levels of parallelism with a neat diagram.	8	6
8	What is distributed computing? Define a distributed system and explain its general concepts.	6	7
9	With a neat diagram, explain the components of a distributed system (layered view).	8	8
10	With a neat diagram, explain how the layered view applies to a cloud computing system.	6	9
11	Explain the three-phase process (R&D, commercialization, commoditization) of the eras of computing.	6	1
12	Compare SISD, SIMD, MISD, and MIMD architectures with examples.	6	4

Module 1: Modern Computing Approaches

Topic 3: Cluster and Grid Computing

Introduction, need for clusters, components and characteristics, types and examples; grid computing, the power-grid analogy, virtual organisations, characteristics, types, examples, advantages and limitations

Course	Modern Computing and Python Programming (26ESCC106)
Module	Module 1 — Modern Computing Approaches
Topic coverage	Cluster and Grid Computing — Introduction and Characteristics
Position in module	After Distributed Computing; before Cloud Computing
Mapped CO	CO1 — Describe modern computing approaches and their characteristics
Bloom's level	L1 — Remember, L2 — Understand

1. Introduction

The previous topic established that a distributed system is a collection of independent computers that presents itself to its users as a single coherent system [2]. That definition permits many different arrangements, and the differences between them matter. Two of these arrangements – clusters and grids – are the subject of the topic in the module. They are important in their own way, and they are also the foundation on which cloud computing, the next topic in this module, is built.

Both answer the same question: how do you obtain more computing capacity than one machine can provide? Their answers differ in one respect above all others, and it is worth stating plainly at the outset because every other difference follows from it.

A cluster is many computers owned and administered by one organisation, in one place, connected by a dedicated network. A grid is many computers owned and administered by different organisations, in different places, connected by the public network. Ownership is the distinction; everything else is a consequence of it.

2. Cluster Computing

2.1 Definition

A computer cluster is a set of complete, standalone computers connected by a dedicated high-speed local network and working together as a single integrated computing resource [1, 3]. Each machine in the cluster — called a node — has its own processor, memory, storage and operating system, and could function independently. What makes the collection a cluster is that software coordinates the nodes so that a user submits work to the cluster as a whole rather than to any particular machine.

Figure 3.1 illustrates the basic idea. The input data is given to the root (master) node, which divides the work and distributes the parts to the slave (worker) nodes. The nodes work on their parts in parallel and exchange intermediate results with one another over the network where the problem requires it. The root node then collects the partial results and produces the final output.

2.2 Why Clusters Exist

Until the early 1990s, an organisation needing very high computational capacity bought a supercomputer: a custom-designed machine with specially built processors and memory systems, produced in small numbers and priced accordingly. Meanwhile, mass-produced personal computer components were improving rapidly and falling in price, precisely because they were mass-produced.

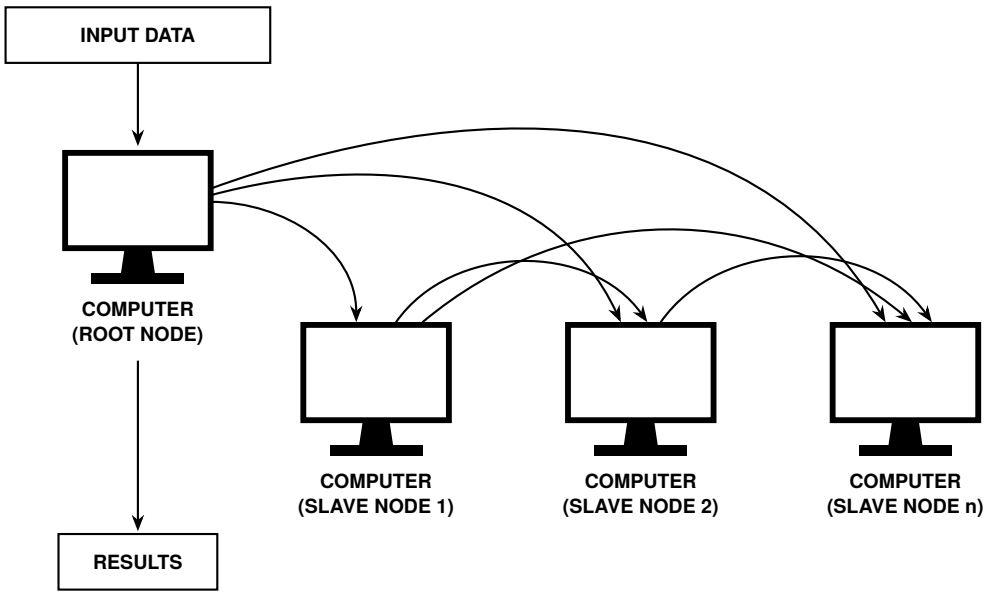


Figure 3.1: *Basic working model of cluster computing*

The cluster is the consequence of noticing that a hundred ordinary machines cost far less than one extraordinary machine of comparable aggregate capacity. The idea was demonstrated in 1994 at NASA's Goddard Space Flight Center, where Thomas Sterling and Donald Becker built a working parallel machine from sixteen commodity personal computers and standard Ethernet running free software. The arrangement became known as a Beowulf cluster [4], and it established that serious parallel computing did not require specialised hardware.

The trade-off is real and should be understood. Commodity components are cheap but their interconnect is slower relative to processor speed than in a purpose-built supercomputer. A cluster therefore performs well on problems that divide into parts needing little communication between them, and less well on problems requiring constant exchange of intermediate results.

2.3 Components of a Cluster

Component	Function
Compute nodes	Standalone computers that perform the actual work; each has its own processor, memory and operating system
Head or master node	The single point of entry through which users log in and submit work; it does not usually perform computation itself
High-speed interconnect	A dedicated private network joining the nodes — typically 10-, 25- or 100-Gigabit Ethernet, or InfiniBand where very low latency is required
Cluster middleware	Software that makes the collection behave as one machine, providing the single system image and handling node failure
Resource manager and job scheduler	Accepts submitted jobs, queues them, allocates nodes and manages execution — for example Slurm or PBS
Parallel programming environment	Libraries that let one program run across many nodes, most commonly MPI for communication between nodes and OpenMP within a node
Shared storage	A file system visible to every node, so that a job can read its input and write its output regardless of where it runs

2.4 Cluster Architecture

Figure 3.2 shows the generic architecture of a cluster, as presented by Yeo, Buyya *et al.* [3]. It is read from the bottom up. The nodes — PCs or workstations — are joined by the cluster interconnection network or switch.

Each node has network interface hardware that attaches it to this network, communication software that provides fast and reliable data exchange between nodes, and its own operating system. Above the individual nodes sits the cluster middleware, which provides the single system image and the availability infrastructure: it makes the separate machines appear as one computer and keeps the service running when a node fails. On top of the middleware, sequential applications run directly, while parallel applications run through a parallel programming environment — compilers, message-passing libraries such as MPI, debuggers and performance tools — that lets one program use many nodes at once.

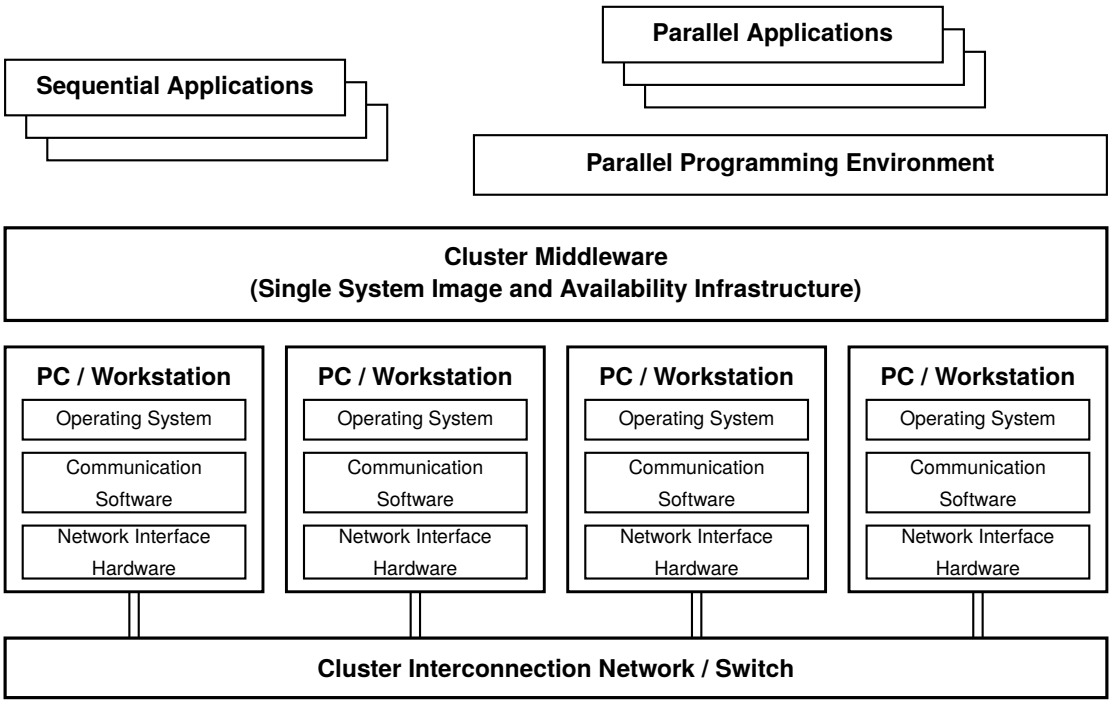


Figure 3.2: *Cluster computer architecture (redrawn from [3]; the figure originates in [7])*

Figure 3.3 shows how such a cluster is typically deployed in practice. Users never connect to the compute nodes directly; they log in to the head node over the campus network or the Internet and submit their jobs to the scheduler running there. The head node, the shared storage and every compute node are joined by the private high-speed interconnect, which carries the messages exchanged between the parts of a running job as well as the traffic between each node and the shared file system. The head node and the shared storage are where much of the middleware and scheduling software of Figure 3.2 actually runs. Everything inside the dashed boundary belongs to one organisation and sits in one facility.

2.5 Single System Image

The property that distinguishes a cluster from a merely networked room of computers is the single system image: the cluster presents itself to the user as one machine. The user does not choose which nodes will run their program, does not log in to individual machines, and does not manually distribute data. They describe the resources their job needs and submit it; the scheduler decides where it runs. This abstraction is what makes a cluster usable, and providing it is the main job of the cluster middleware [3, 1].

2.6 Characteristics of Cluster Computing

The following are the characteristics of cluster computing:

- **Single administrative domain** — all nodes are owned, funded and administered by one organisation, under one security policy
- **Physical co-location** — nodes sit in one room or one data centre, typically in the same rack or set of racks
- **Homogeneity** — nodes are usually identical or near-identical in hardware, operating system and configuration, which simplifies scheduling and programming

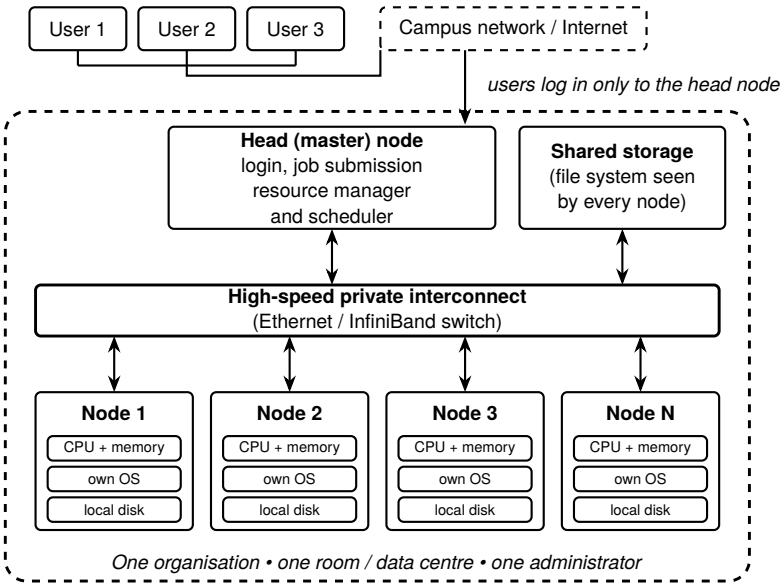


Figure 3.3: *Physical organisation of a cluster*

- **Tight coupling** — a dedicated, private, high-bandwidth and low-latency interconnect, not shared with general network traffic
- **Dedicated resources** — nodes exist to serve the cluster and are not used for anything else
- **Centralised scheduling** — one resource manager has complete knowledge of and authority over every node
- **Single system image** — the collection appears to the user as one computer
- **High availability** — the failure of one node need not stop the cluster, because work can be restarted elsewhere
- **Incremental scalability** — capacity grows by adding nodes, within the limits of space, power and interconnect

2.7 Types of Clusters

Clusters are built for three broad purposes [1].

Type	Purpose	Typical use
High-performance (HPC) cluster	Maximise computational throughput on a single large problem by dividing it across nodes	Scientific simulation, weather modelling, molecular dynamics, model training
Load-balancing cluster	Distribute many independent incoming requests across nodes so that no node is overwhelmed	Web servers, application servers, online services
High-availability (failover) cluster	Maintain service continuity by having standby nodes take over when an active node fails	Database servers, banking and other systems that must not stop

2.8 Examples of Clusters

- Beowulf clusters — the original 1994 NASA machine and the many university and departmental clusters modelled on it, built from commodity PCs running Linux
- The PARAM series developed by C-DAC under the National Supercomputing Mission, for example PARAM Siddhi-AI at C-DAC Pune and PARAM Pravega at IISc Bengaluru — high-performance clusters of many compute nodes joined by a high-speed interconnect [11]

- Most of the world's fastest computers listed in the TOP500 ranking, which are built as clusters rather than as single custom machines [10]
- Web server farms behind a load balancer — search engines, e-commerce and railway or airline booking sites spread incoming requests across many identical servers (load-balancing clusters)
- High-availability database clusters, such as Oracle Real Application Clusters, used for core banking and reservation systems where a standby node must take over if an active node fails

2.9 Advantages and Limitations

Advantages

- Substantially better price-to-performance ratio than a purpose-built supercomputer
- Capacity can be increased in small increments rather than by replacing the whole machine
- Redundancy is inherent, so availability is high if the software is configured for it
- Built from standard components with standard software, so expertise and spare parts are widely available

Limitations

- Bounded by one organisation's budget, floor space, power supply and cooling capacity
- Programs must be written explicitly for parallel execution; existing sequential software does not benefit
- The interconnect eventually limits how far a tightly coupled problem can be scaled
- Capacity is fixed once purchased — it must be sized for peak demand and is idle at other times

3. Grid Computing

3.1 Definition

Grid computing is the coordinated sharing of computing resources that belong to different organisations, in different locations, in order to solve a common problem. The resources are federated rather than owned in common: each participating institution keeps ownership of and authority over its own machines, and contributes their use under agreed terms.

The term was introduced in the late 1990s, principally through the work of Ian Foster and Carl Kesselman [5], who characterised the field by the problem of coordinated resource sharing across institutional boundaries [6].

3.2 The Power Grid Analogy

The name is borrowed from the electrical power grid, and the analogy explains the ambition [5]. When you switch on a light, you do not know or care which generating station supplied the electricity, whether it came from one source or several, or where those sources are. You draw power from a socket, it is metered, and you are billed. The generation, transmission and coordination are invisible.

Grid computing proposed the same relationship with computation: a researcher would draw processing power from a socket without knowing which institution's machines were serving them. It is worth noting that this ambition was only partially realised. Electricity is a uniform commodity and computing is not — machines differ in architecture, software and policy in ways that generators do not — and the seamlessness the analogy promised proved difficult to deliver. The vision was largely fulfilled later, by cloud computing, and by a different route.

3.3 The Virtual Organisation

The central concept in grid computing is the virtual organisation: a set of individuals or institutions, drawn from different real organisations, who agree to share specified resources under specified rules for a specified purpose. A virtual organisation is defined by its sharing agreement, not by a legal entity or a physical site [6].

A concrete example makes this clear. Physicists at twenty universities in twelve countries collaborate on one experiment. Each university owns a cluster. No university is willing to hand over control of its cluster to any other, and each has its own security policy, its own users and its own local commitments. Yet the experiment needs more capacity than any one of them has. The virtual organisation is the arrangement under which those twenty clusters serve the experiment while each remains under its owner's control. Grid middleware exists to make that arrangement work in practice.

3.4 Grid Architecture

Figure 3.4 shows how the sites of a grid are arranged and how a user's job reaches them. Each site — a university cluster, a research laboratory's supercomputer, a data archive, a scientific instrument, or a pool of contributed desktop PCs — is owned by a different institution. Every site keeps its own local scheduler and its own policy, which decide what may run there, for whom and at what priority; nothing in the grid overrides them. A layer of grid middleware at each site acts as a gateway, translating a request arriving from the grid into a local action that the owner has already agreed to permit. The sites are joined only by the wide-area network, drawn in the middle of the figure, and the dashed boundary around them is the virtual organisation: an agreement, not a building or a company.

The upper part of the figure shows what happens when work is submitted. The user does not approach the sites one by one. They authenticate once with a digital certificate and hand the job to a grid portal or resource broker, which is the only component they deal with directly. The broker asks the information service which resources are currently available and suitable — what exists, what is free, and what matches the job's requirements — because membership of a grid changes without notice. The security service then establishes that this user, authenticated at their home institution, is to be trusted at each site the broker has chosen, so that one login is accepted everywhere (single sign-on). Only then does the broker dispatch the job, and the data it needs, across the network. At the receiving site the middleware hands the job to the local scheduler, which queues it like any local work, and the results travel back the same way. Everything the user would otherwise have to do — find resources, negotiate access at each institution, move data, track progress — is what the middleware and these services exist to do.

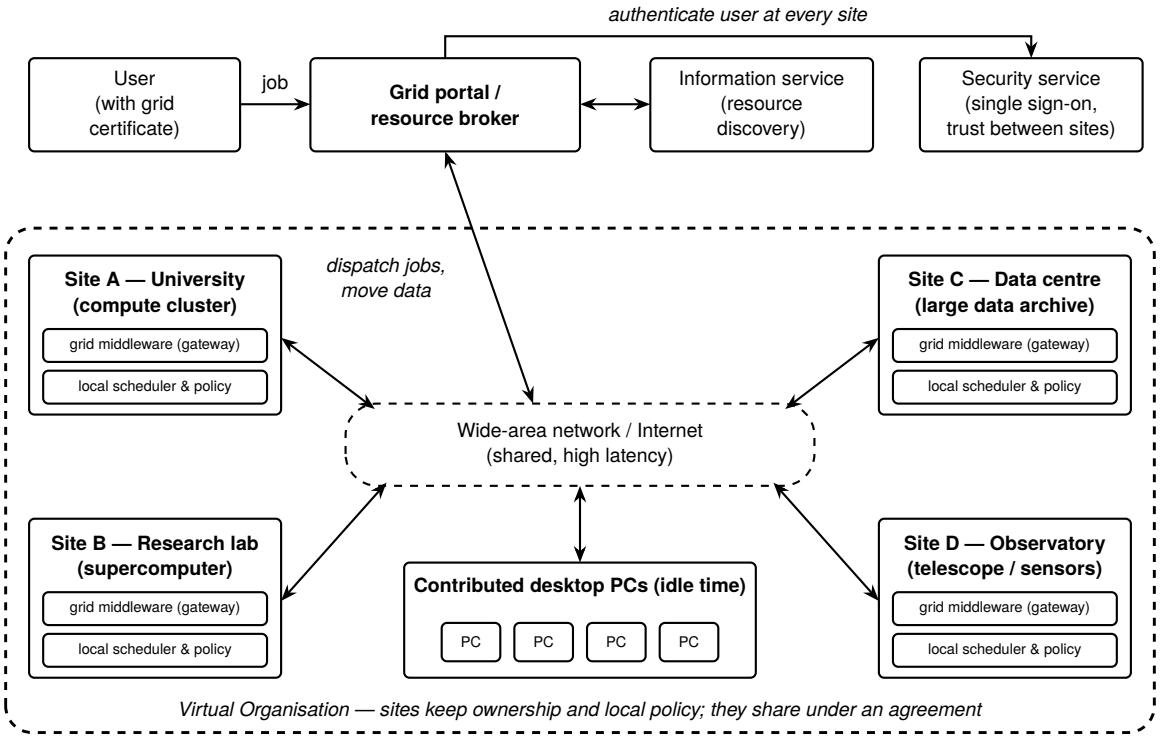


Figure 3.4: *Architecture of a grid — independent sites joined into a virtual organisation through grid middleware*

Internally, this software is organised in layers [6]. The Fabric layer is the physical resources themselves, each under its local manager. The Connectivity layer provides communication and security between sites. The Re-

source layer negotiates access to, and controls, one resource at a time, such as starting and monitoring a job on one cluster. The Collective layer coordinates many resources together — discovery, brokering, scheduling across sites and monitoring — and it is where the information service and the resource broker of Figure 3.4 belong. Applications sit on top and use these services. The Globus Toolkit provided software for the Connectivity, Resource and Collective layers.

3.5 Characteristics of Grid Computing

- **Multiple administrative domains** — no single owner or authority; each site retains autonomy over its own resources and enforces its own policies
- **Geographical distribution** — resources are spread across cities, countries or continents
- **Heterogeneity** — nodes differ in hardware architecture, operating system, software stack and configuration
- **Loose coupling** — sites are connected by wide-area networks or the public Internet, with latency orders of magnitude higher than a cluster interconnect
- **Dynamic and unpredictable membership** — resources join, leave and fail without notice, so the grid must tolerate their disappearance
- **Non-dedicated resources** — contributed machines usually have local work of their own, and grid jobs run on spare capacity at lower priority
- **Policy-based sharing** — what may be used, by whom, and for what is governed by negotiated agreements rather than by ownership
- **Decentralised scheduling** — no single scheduler has authority over all resources; allocation is negotiated between sites
- **Reliance on standard protocols and middleware** — the Globus Toolkit and the Open Grid Services Architecture were the principal efforts to standardise this
- **Best-effort service** — there is generally no guarantee of when a job will run or on what, because no participant has undertaken to provide capacity on demand

3.6 Types of Grids

Grids are classified by what they share [1, 5].

Type	Description
Computational grid	Aggregates processing capacity across sites to execute computationally intensive work
Data grid	Provides controlled, coordinated access to very large datasets held at multiple institutions
Collaboration or service grid	Shares instruments, services and applications — telescopes, sensors, specialised software — rather than raw computation
Desktop or volunteer grid	Uses idle time on ordinary personal computers contributed by the public, coordinated over the Internet

3.7 Examples

- The Worldwide LHC Computing Grid, which distributes and processes data from the Large Hadron Collider across roughly 170 computing centres in more than 40 countries [8]
- The European Grid Infrastructure, federating computing resources across European research institutions [9]
- GARUDA, the Indian national grid initiative coordinated by C-DAC, connecting research and academic institutions across the country [11]

- Volunteer computing projects for protein folding and radio-signal analysis, run on donated capacity from personal computers

3.8 Advantages and Limitations

Advantages

- Exploits existing resources that would otherwise sit idle, at little additional hardware cost
- Aggregate capacity far exceeding what any single participant could afford to build
- Enables collaboration across institutional and national boundaries without transferring ownership
- No participant is required to purchase the whole of the capacity it occasionally needs

Limitations

- Heterogeneity makes software difficult to write, deploy and validate across sites
- Security and trust are hard: a user authenticated at one institution must be granted appropriate rights at another that does not directly know them
- Wide-area latency makes the grid unsuitable for tightly coupled problems requiring frequent communication
- Middleware is complex and demands specialist administration at every participating site
- No guarantee of availability or performance, because resources are contributed rather than committed
- Accounting is difficult — measuring and settling who consumed whose capacity never worked cleanly, which limited participation beyond publicly funded research

4. Cluster Computing versus Grid Computing

The table below draws together the differences established in the two previous sections [1, 6].

Aspect	Cluster computing	Grid computing
Ownership	One organisation	Many independent organisations
Administration	Single administrative domain	Multiple autonomous domains
Location	Co-located in one facility	Geographically distributed
Hardware and software	Homogeneous	Heterogeneous
Interconnect	Dedicated high-speed LAN, low latency	Wide-area network or Internet, high latency
Coupling	Tight	Loose
Resource dedication	Dedicated to the cluster	Shared with the owner's own local workload
Scheduling	Centralised, with full authority	Decentralised and negotiated
Membership	Static and known	Dynamic and variable
Security	One policy, one trust domain	Cross-domain trust and identity federation required
Service guarantee	Predictable and controlled	Best effort
User view	A single machine	A pool of resources reached through middleware
Suitable workload	Tightly coupled parallel problems	Independent, coarse-grained tasks

5. Review Questions

CO1 — Describe modern computing approaches and their characteristics
Bloom's Level L1 — Remember, L2 — Understand

1. Define cluster computing and explain what distinguishes a cluster from a set of networked computers in the same room.
2. List the principal components of a cluster and state the function of each.
3. Explain the term single system image and describe why it matters to a cluster user.
4. List any six characteristics of cluster computing.
5. With a neat diagram, explain the architecture of a computer cluster, showing both its physical organisation and its software layers.
6. With a neat diagram, explain how a user's job is executed on a grid, stating the role of the resource broker, the information service, the security service and the grid middleware at each site.
7. Differentiate between high-performance, load-balancing and high-availability clusters, giving one application of each.
8. State any four limitations of cluster computing.
9. Explain the power grid analogy. Identify one respect in which the analogy does not hold.
10. What is a virtual organisation? Explain with an example why it is the central concept in grid computing.
11. List any six characteristics of grid computing.
12. Differentiate between a computational grid and a data grid.
13. Compare cluster and grid computing with respect to ownership, coupling, homogeneity, scheduling and service guarantee.
14. Explain why security is substantially harder to provide in a grid than in a cluster.
15. Why is grid computing unsuitable for tightly coupled parallel problems?
16. Give one real-world example each of a high-performance, a load-balancing and a high-availability cluster.

6. References and Further Reading

Items 1 to 7 are the sources used in preparing these notes and are suggested for further reading; section and chapter numbers are indicative. Items 8 to 11 are the project and ranking sites from which the examples and figures quoted in the text were taken.

[1] K. Hwang, G. C. Fox and J. J. Dongarra, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, Morgan Kaufmann, 2012 — Chapter 2 on computer clusters for scalable parallel computing, and Chapter 7 on grid computing systems and resource management.

[2] R. Buyya, C. Vecchiola, S. T. Selvi, S. Poojara and S. N. Srirama, *Mastering Cloud Computing*, 2nd Edition, Morgan Kaufmann, 2026 — Chapter 2 on the principles of parallel and distributed computing.

[3] C. S. Yeo, R. Buyya, H. Pourreza, R. Eskicioglu, P. Graham and F. Sommers, "Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers", in A. Y. Zomaya (ed.), *Handbook of Nature-Inspired and Innovative Computing*, Springer, 2006 — the source of Figure 3.2.

[4] T. Sterling, D. J. Becker, D. Savarese, J. E. Dorband, U. A. Ranawake and C. V. Packer, "BEOWULF: A Parallel Workstation for Scientific Computation", *Proceedings of the 24th International Conference on Parallel Processing*, 1995.

[5] I. Foster and C. Kesselman (eds.), *The Grid: Blueprint for a New Computing Infrastructure*, 2nd Edition, Morgan Kaufmann, 2004.

[6] I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001 — the source of the virtual organisation and of the layered grid architecture.

[7] R. Buyya (ed.), *High Performance Cluster Computing: Architectures and Systems*, Volume 1, Prentice Hall PTR, 1999 — where the architecture of Figure 3.2 first appeared.

[8] Worldwide LHC Computing Grid, CERN. https://wlcg.web.cern.ch/

[9] EGI Federation (European Grid Infrastructure). https://www.egi.eu/

[10] TOP500 list of the world's most powerful computer systems. https://www.top500.org/

[11] Centre for Development of Advanced Computing (C-DAC) — PARAM systems under the National Supercomputing Mission, and the GARUDA national grid initiative. https://www.cdac.in/

Module 1: Modern Computing Approaches

Topic 4: Cloud Computing

Text Book 2: Rajkumar Buyya, Christian Vecchiola and S. Thamarai Selvi, *Mastering Cloud Computing*, McGraw Hill Education, 2013 — Sections 4.2.1–4.2.4, 4.3.1–4.3.4

1. Introduction to Cloud Computing

Q. What is cloud computing? Explain the meaning of the term with suitable examples. **[6 Marks]**

Cloud computing covers any IT service that is **consumed as a utility** and **delivered over a network**, usually the Internet. Such services include infrastructure, development platforms, applications, and services.

1. **Utility model:** Computing is used like electricity or water — the user consumes it on demand and pays according to use, without owning the equipment.
2. **Delivered over a network:** The resources are located in the provider's datacenters and are reached through the Internet.
3. **Layered stack:** All forms of cloud computing fit into one layered stack — **IaaS** → **PaaS** → **SaaS** — from hardware up to end-user software (Section 2).
4. **Deployment:** Clouds are deployed as **public, private, hybrid**, or **community** clouds (Section 7).
5. **Adoption:** Cloud computing is adopted mainly for **economic reasons**: it reduces capital costs and follows a **pay-as-you-go** model (Section 13).

Examples: Amazon EC2 (virtual machines), Google AppEngine (a platform for building applications), and Gmail or Google Docs (ready-to-use applications).

Analogy: Cloud computing is like renting a hall for a function instead of building one — you pay only for the time you use it, and the owner maintains it.

2. The Cloud Reference Model: Architecture

Q. With a neat diagram, explain the cloud computing reference model (architecture). **[8 Marks]**

All realisations of cloud computing can be organised into one **layered stack**, from the hardware at the bottom to end-user software at the top. Figure 4.1 (Text Book 2, Fig. 4.1) shows this architecture.

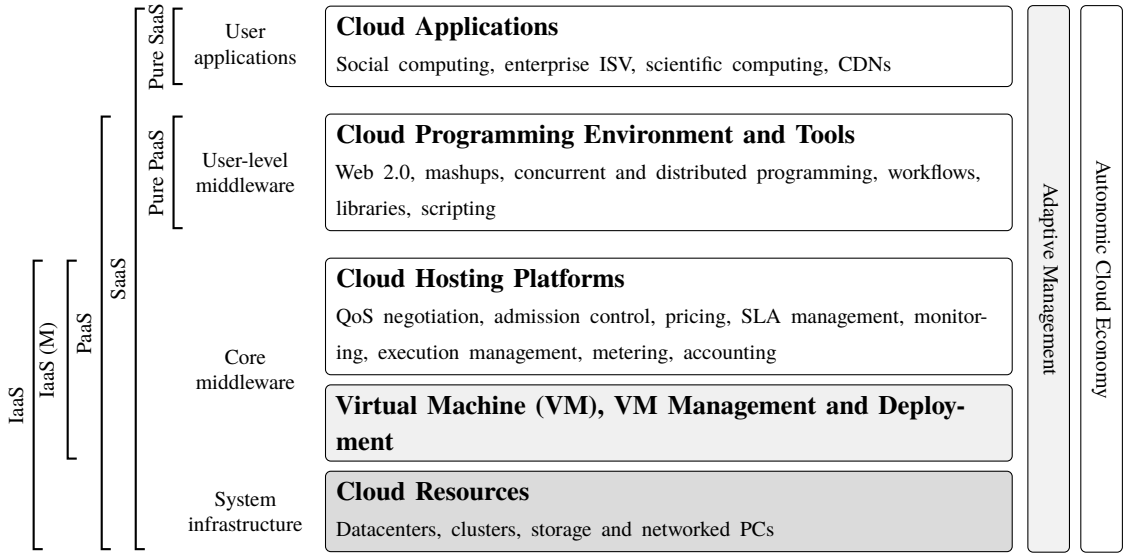


Figure 4.1: The cloud computing architecture (Text Book 2, Fig. 4.1, redrawn)

Reading the brackets on the left: SaaS spans the whole stack; PaaS covers everything from user-level middleware down; IaaS covers the hosting platforms and the resources. The "Pure" and "(M)" variants cover only their own layer.

Explanation of the layers (bottom to top):

1. **Cloud resources (system infrastructure):**
   - They supply the "computing horsepower" for the services — typically a **datacenter** with hundreds or thousands of stacked nodes.
   - They can be **heterogeneous**: clusters and even networked PCs; database systems and storage services can also be part of it.
2. **Core middleware:** It manages the physical infrastructure. Its goals are an appropriate **runtime environment** for applications and the **best use of resources**.
   - At the bottom, **virtualization** guarantees runtime customisation, application isolation, sandboxing, and quality of service.
   - **Hardware virtualization** is the most common: **hypervisors** manage the resource pool and expose it as a collection of **virtual machines (VMs)**, so CPU, memory, and devices can be finely partitioned.
   - It is usually paired with **storage and network virtualization**, so that the whole infrastructure is virtualized and controlled.
   - **Programming-level virtualization** gives a portable runtime (Java, .NET, Python), so the user does not build the system from bare metal.
   - **Infrastructure management functions:** QoS negotiation, admission control, execution management and monitoring, accounting, and billing (the **cloud hosting platforms** layer).
3. **User-level middleware (cloud programming environment and tools):** Web 2.0 interfaces, mashups, frameworks for concurrent and distributed programming, workflows, libraries, and scripting, used to build cloud applications.
4. **User applications (cloud applications):** Social computing, enterprise ISV applications, scientific computing, and content delivery networks (CDNs).
5. **Adaptive management:** Any cloud-style service should behave **autonomically**, especially for availability and performance, **scaling elastically** on demand. SaaS should do this automatically; PaaS and IaaS usually expose it as part of their API.
6. **Autonomic cloud economy:** Along the whole stack, the cloud follows an economic, pay-per-use model.

Conclusion: Every cloud service — infrastructure, platform, or application — can be placed at one of these layers, which is why the model is called the **reference model**.

3. Mapping the Layers to Service Models (IaaS, PaaS, SaaS) and XaaS

Q. Explain how the layers of the reference model map to IaaS, PaaS, and SaaS. Classify cloud computing services with examples. What is XaaS? **[8 Marks]**

1. Infrastructure-as-a-Service (IaaS)

- IaaS = **cloud hosting platforms + cloud resources**.
- Two kinds of providers: those offering **both** the management layer and the physical infrastructure, and those offering **only the management layer — IaaS (M)** — which is integrated with other IaaS providers' infrastructure.

2. Platform-as-a-Service (PaaS)

- PaaS = **cloud programming environments and tools** (user-level middleware): web interfaces, command-line tools, and frameworks for concurrent and distributed programming.
- Users build cloud-specific applications against the API of the user-level middleware.
- PaaS usually bundles the infrastructure too; **Pure PaaS** offers only the user-level middleware, to be combined with a virtual or physical infrastructure.

3. Software-as-a-Service (SaaS)

- SaaS = the **top layer**: mostly web applications that rely on the cloud to serve end users.
- The horsepower of IaaS and PaaS lets **independent software vendors (ISVs)** deliver applications over the Internet; gaming portals and social networking sites also belong here.

Cloud computing services classification (Table 4.1 of the textbook):

Category	Characteristics	Product Type	Vendors and Products
SaaS	Customers get applications accessible anytime, from anywhere	Web applications and services (Web 2.0)	SalesForce.com (CRM), Clarizen.com (project management), Google Apps
PaaS	Customers get a platform for developing applications hosted in the cloud	Programming APIs and frameworks; deployment systems	Google AppEngine, Microsoft Azure, Manjrasoft Aneka, Data Synapse
IaaS/HaaS	Customers get virtualized hardware and storage on which to build their infrastructure	VM management infrastructure; storage management; network management	Amazon EC2 and S3, GoGrid, Nirvanix

4. XaaS — Everything as a Service

- Services from different providers can be **combined** into one integrated solution covering the whole computing stack.
- IaaS supplies the VMs on which PaaS is deployed; with no PaaS layer, the virtual infrastructure is customised directly with the needed software stack.
- **Example — virtual Web farm:** web servers, database servers, and load balancers on which prepackaged software runs web applications.
- This cuts start-ups' **capital investment** in IT: they can commercialise ideas quickly and grow the infrastructure along with revenue.

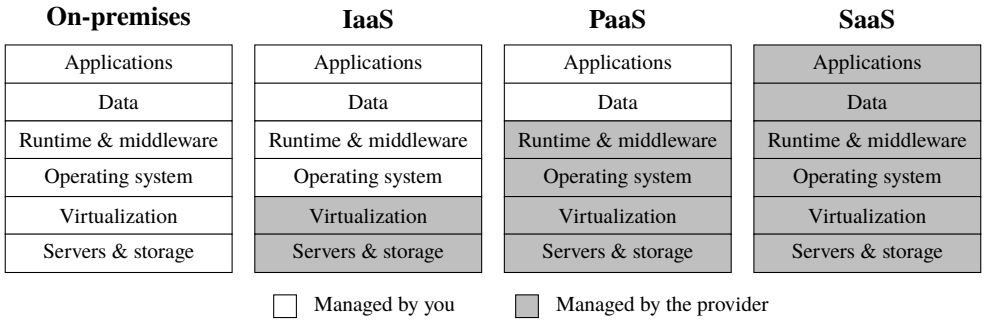


Figure 4.2: Who manages what in IaaS, PaaS, and SaaS (an everyday analogy)

Analogy: IaaS is like renting an **empty flat** (you bring the furniture), PaaS is like renting a **furnished flat** (you only bring your belongings), and SaaS is like staying in a **hotel room** (everything is ready).

4. Infrastructure- and Hardware-as-a-Service (IaaS/HaaS)

4.1 Technology and Benefits of IaaS

Q. What is IaaS/HaaS? Explain its core technology and benefits. **[6 Marks]**

IaaS/HaaS delivers **customisable infrastructure on demand** — from single servers to entire infrastructures with network devices, load balancers, and database and web servers. It is the most popular and most developed cloud market segment.

1. **Core technology — hardware virtualization:** One or more configured, interconnected VMs form the distributed system on which applications are deployed.
2. **Unit of deployment and pricing:** VMs are the atomic unit, priced by their virtual hardware — memory, number of processors, and disk storage.
3. **Virtualization benefits:** workload partitioning, application isolation, sandboxing, and hardware tuning.
4. **Provider's view:** better use of the IT infrastructure and a more secure environment for running third-party applications.
5. **Customer's view:** lower administration, maintenance, and capital (hardware) costs, with full customisation.
6. **VM images:** VMs come either with only an OS installed or as **prepackaged images** with a ready software stack (web servers, database servers, LAMP stacks).
7. **Extra services:** SLA resource-based allocation, workload management, infrastructure design through web interfaces, and integration of third-party IaaS.

LAMP = **L**inux + **A**pache (web server) + **M**ySQL (database) + **P**HP (server-side scripting) — the most common packaged stack for quickly deploying web applications.

4.2 IaaS Reference Implementation

Q. With a neat diagram, explain the reference implementation of Infrastructure-as-a-Service. Differentiate complete IaaS and IaaS (M). **[8 Marks]**

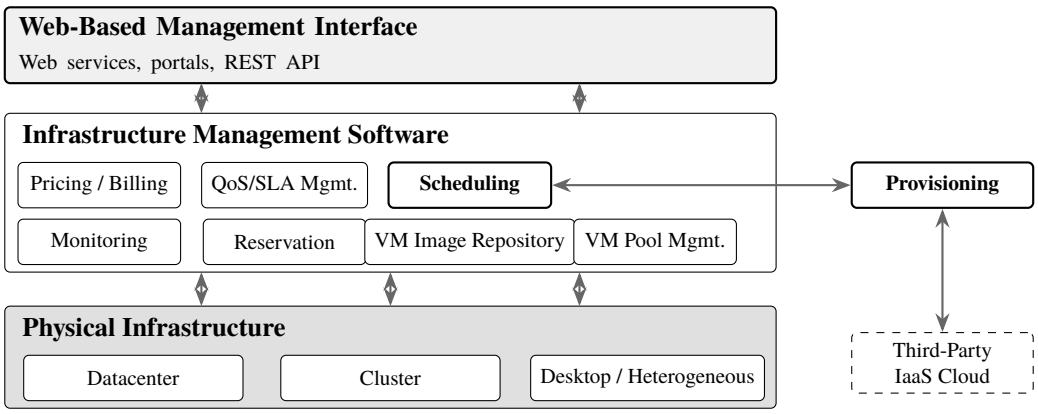


Figure 4.3: Infrastructure-as-a-Service reference implementation (Text Book 2, Fig. 4.2, redrawn)

Requests flow from the web interface to the management software, which runs VMs on the physical layer or, through provisioning, on a third-party IaaS cloud. The implementation has **three layers**:

1. User interface (top) — gives access to the services of the management layer, built on Web 2.0 technologies: web services, RESTful APIs, and mash-ups.

- Web 2.0 applications give full management consoles inside a browser.
- Web services and REST APIs let programs use the service **without human intervention**.

2. Infrastructure management software (middle) — the core of IaaS; **VM management** is its key function. The **scheduler** allocates VM instances and works with:

- **Pricing/billing:** cost of each VM instance; data used to charge the user.
- **Monitoring:** tracks each VM's execution; data for reporting and performance analysis.
- **Reservation:** records all VM instances executed or to be executed in the future.
- **QoS/SLA management:** a repository of SLAs; with monitoring, it ensures each VM runs at the agreed QoS.
- **VM image repository:** a catalogue of VM images (some providers allow users to upload their own).
- **VM pool manager:** tracks all live instances.
- **Provisioning:** with the scheduler, obtains VM instances from a **third-party IaaS provider**, outside the local infrastructure.

3. Physical infrastructure (bottom) — depends on the use of the cloud:

- Service provider → a massive **datacenter** with hundreds or thousands of nodes.
- In-house cloud (an SME or a university department) → a **cluster**.
- Heterogeneous environment → PCs, workstations, and clusters aggregated (an evolution of desktop grids that harvest idle machines).
- It may also include virtual resources rented from external IaaS providers.

Complete IaaS vs. IaaS (M):

Sl.	Aspect	Complete IaaS	IaaS (M)
1	Layers offered	All three layers as a service	Only the user interface and management layers
2	Infrastructure	Provider's own physical infrastructure	Needs credentials to a third-party IaaS or a private infrastructure
3	Examples	Amazon, GoGrid, Joyent, Rightscale, Terremark, Rackspace, ElasticHosts, Flexiscale	Enomaly, Elastra, Eucalyptus, OpenNebula, and offerings from VMware, IBM, and Microsoft

Conclusion: The model is a reference only — real products add or omit features. If **storage** is the main service, the three layers remain, but the management layer gives access to large infrastructures and implements **storage virtualization** instead of managing VMs.

5. Platform-as-a-Service (PaaS)

5.1 The PaaS Reference Model

Q. With a neat diagram, explain the Platform-as-a-Service reference model. Explain the development models and delivery options of PaaS.

[8 Marks]

Platform-as-a-Service (PaaS) provides a **development and deployment platform** for running applications in the cloud. It is the middleware on which applications are built.

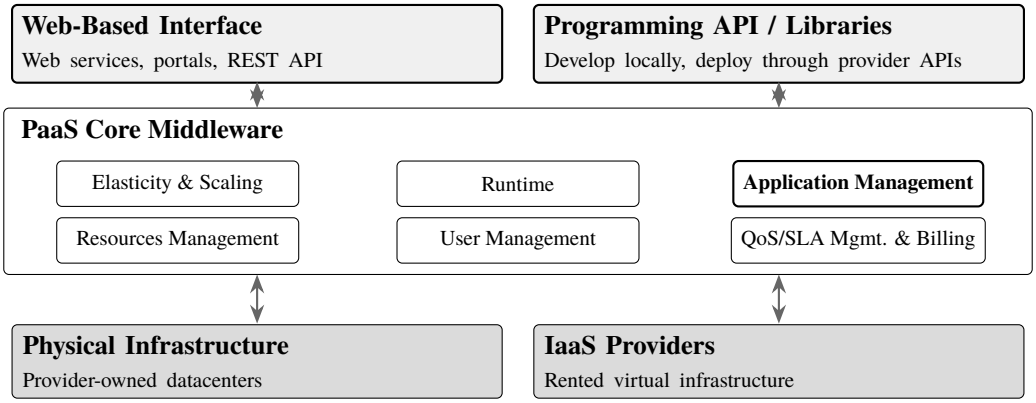


Figure 4.4: The Platform-as-a-Service reference model (Text Book 2, Fig. 4.3, redrawn)

Users reach the **core middleware** through a web interface or programming APIs; the middleware runs applications on owned infrastructure or on rented IaaS.

How PaaS works:

1. **Application management** is the core function of the middleware: it provides a runtime environment for applications, with no service exposed for managing the infrastructure underneath.
2. It **automates** deployment, configuration of application components, provisioning of supporting technology (load balancers, databases), and system change based on user policies.
3. Developers think in terms of **applications** — not hardware, operating systems, or low-level services.
4. The core middleware manages resources and **scales applications** on demand or automatically, as per commitments to users.
5. It is exposed to users as a **web-based interface** or as **programming APIs and libraries**.

Development models:

- **Web-based:** IDEs hosted in the cloud, often with 4GL/visual programming, or rapid prototyping by assembling mash-ups and user-defined components.
- **Programming-language based:** a full object model of the application; more flexible but with longer development cycles. It uses Java, .NET, Python, or Ruby (with some restrictions for scalability and security); applications are developed in traditional IDEs and deployed through the provider's APIs. A local runtime that simulates the cloud may be given for testing (limited, and not optimised for scaling).

Delivery options:

- **Middleware plus infrastructure:** the provider owns large datacenters where the applications run.
- **Pure PaaS:** only the middleware, installed on the user's premises; the middleware is the core value.
- Some vendors deliver both, and ship the middleware alone for private installations.

5.2 Classification of PaaS Offerings

Q. Classify PaaS offerings with examples (PaaS-I, PaaS-II, PaaS-III). **[6 Marks]**

Category	Description	Vendors and Products (Product Type)
PaaS-I	Runtime environment with a web-hosted application development platform; rapid application prototyping	Force.com (M + I), Longjump (M + I)
PaaS-II	Runtime environment for scaling web applications; the runtime may be enhanced with components that add scaling	Google AppEngine (M + I), AppScale (M), Heroku (M + I), Engine Yard (M + I), Joyent Smart Platform (M + I), GigaSpaces XAP (M)
PaaS-III	Middleware and programming model for developing distributed applications in the cloud	Microsoft Azure (M + I), DataSynapse (M), Cloud IQ (M), Manjrasoft Aneka (M), Apprenda SaaS-Grid (M), GigaSpaces DataGrid (M)

M = Middleware, I = Infrastructure (Text Book 2, Table 4.2)

1. **PaaS-I** follows the cloud style fully: design, development, composition, and deployment all happen in a **browser-hosted IDE**.
2. **PaaS-II** targets **scalable web applications** (mostly websites). AppEngine offers a modified, secured Java/Python runtime with extra scaling APIs; AppScale is its open-source, interface-compatible version; Heroku and Engine Yard scale Ruby / Ruby on Rails websites — applications are built normally and uploaded.
3. **PaaS-III** supports **any kind of application**, not only web applications. Azure provides a .NET-based framework for service-oriented applications hosted in Microsoft datacenters; the others provide middleware only.

5.3 Essential Characteristics of PaaS and Vendor Lock-in

Q. Explain the essential characteristics of PaaS. What is vendor lock-in? Explain the financial benefits of PaaS. **[8 Marks]**

Essential characteristics of PaaS (Sam Charrington, Appistry):

1. **Runtime framework:** the "software stack" of PaaS; it executes end-user code according to user and provider policies.
2. **Abstraction:** IaaS gives raw access to virtual or physical infrastructure; PaaS focuses on the **applications**, offering a way to deploy and manage them rather than a set of VMs.
3. **Automation:** deployment and scaling (provisioning extra resources) happen **automatically** as per the SLA; IaaS normally only provides ways to provision more resources.
4. **Cloud services:** services and APIs that simplify building elastic, highly available applications; these are the key differentiators between PaaS vendors (development components, monitoring, management, reporting).

Also essential: integrating **third-party cloud services** through service-oriented architecture, using standard interfaces and protocols.

Vendor lock-in — the major concern with PaaS:

- IaaS gives fully customisable bare virtual servers; PaaS **binds the application** to the provider's APIs and runtime, making migration hard if the relationship ends.

- **Proprietary runtime** (Force.com) → migration is very difficult.
- **Industry-standard runtime but private storage/infrastructure** (Google AppEngine, Microsoft Azure) → alternatives implementing the same interfaces may exist.
- **Open standards** (Appistry Cloud IQ, Heroku, Engine Yard) → migration is easier.

Financial view:

- IaaS shifts capital cost into **operational cost** through outsourcing; PaaS also cuts the cost of development, deployment, and management.
- The **technology-upgrade risk** moves to the PaaS provider, so consumers can focus on their core business.
- Bundled with IaaS, PaaS lets even small start-ups offer integrated hosted solutions at minimal cost.

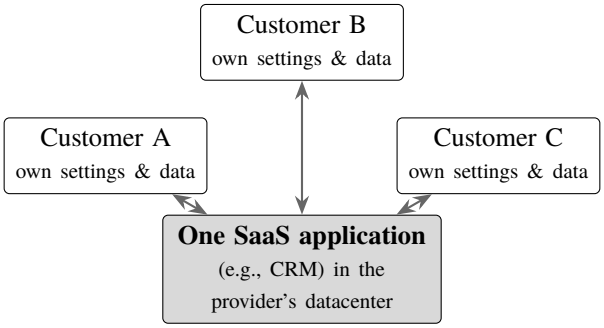
6. Software-as-a-Service (SaaS)

Q. What is SaaS? Explain its key ideas, origin, and the role of Application Service Providers. What is SaaS 2.0? Give examples. **[8 Marks]**

Software-as-a-Service (SaaS) delivers applications over the Internet as **web-based services**, so users need no local installation, licences, or heavy up-front cost — they log in, enter billing details, and use the application.

Key ideas:

1. It **offloads** hardware and software management to a third party; applications are reached through a **web browser** and can be customised.
2. Each customer's specific settings and features are kept in the provider's infrastructure and served on demand.
3. **One-to-many delivery:** one application is shared by many users; it suits common needs such as **CRM** (customer relationship management) and **ERP** (enterprise resource planning), with customisation for specific needs.
4. **Multitenancy:** distinguishes SaaS from packaged software. The provider centralises infrastructure management, upgrades transparently, and shares costs across a large user base; each customer pays a small usage fee.



One code base, shared infrastructure; each customer is isolated and pays a small usage fee

Figure 4.5: Multitenancy — one application serves many customers, each with its own settings

Origin — SIIA definition (2001):

- The SaaS idea predates cloud computing (late 1990s); the acronym was coined by the **Software & Information Industry Association (SIIA)** in 2001.
- In SIIA's view, the application is deployed from a **central datacenter** over a network (Internet, intranet, LAN, or VPN) and used for a **recurring fee**; users rent, subscribe to, or are granted access to it.

Application Service Providers (ASPs) — the forerunners, which already had these SaaS traits:

- The product sold is **application access**.
- The application is **centrally managed**.
- The service is **one-to-many**.
- The service is an **integrated solution delivered on contract**, as promised.

ASPs first hosted packaged applications for many customers; this became costly as customisation grew. SaaS is more flexible: users integrate new services, inject their own components, and design workflows — made possible by Web 2.0 turning the browser into a full interface.

SaaS and cloud computing: SaaS sits at the top of the stack (Figure 4.1) and fits the XaaS vision. Early benefits for lead users: lower software cost and total cost of ownership (TCO), better service levels, rapid implementation, standalone configurable applications, basic application/data integration, and subscription and pay-as-you-go pricing.

SaaS 2.0:

- Not a new technology — a new way of using SaaS, focused on rapid achievement of **business objectives** on SLA-driven, robust infrastructure and platforms.
- Applications are built by **interconnecting components** and value-added services from different providers.
- The market becomes a network of service providers clustered around a few "big hubs"; each component carries its own SLA and price, and customers choose which to integrate.

Examples of SaaS:

Class	Examples	Notes
CRM / ERP	SalesForce.com, NetSuite, RightNow	SalesForce.com is built on Force.com; AppExchange lets customers publish, search, and integrate add-ons. NetSuite combines financials, CRM, inventory, and e-commerce; RightNow centres on customer experience.
Social / professional networking	Facebook, LinkedIn	Extendable with third-party plug-in applications that get access to the user's contacts and profile.
Office automation	Google Documents, Zoho Office	Web-based documents, spreadsheets, and presentations, shareable from anywhere.
SaaS enablers	Box.net	Web space and profile extended with third-party apps (office tools, CRM integration, social sites, photo editing).

Conclusion: SaaS evolved from ASPs into one-to-many, multitenant web applications, and SaaS 2.0 composes services from many providers to meet business goals.

7. Types of Clouds

Q. What are the different types of clouds? Explain the classification based on administrative domain. **[6 Marks]**

Clouds are parallel and distributed systems that present physical and virtual computers as **one unified computing resource**. The most useful classification is by **administrative domain** — the boundary within which the services are implemented.

Type	Definition
Public cloud	Open to the wider public
Private cloud	Built within an institution's premises; used by its members or a subset of them
Hybrid (heterogeneous) cloud	A private cloud augmented with resources or services from a public cloud
Community cloud	Spans multiple administrative domains (public, private, hybrid); serves the needs of a specific industry

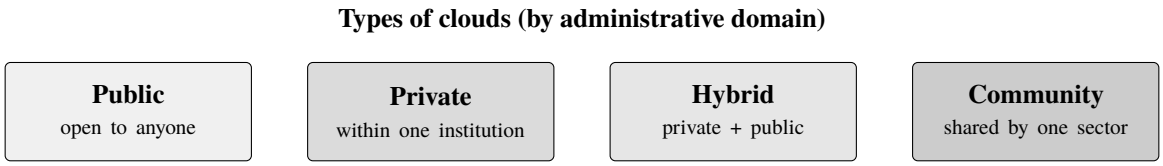


Figure 4.6: Types of clouds by administrative domain

Everyday analogy: A public cloud is like a **public bus** (anyone can board and pay per trip); a private cloud is like a **college bus** (only for members); a hybrid cloud is like a college that **hires extra buses** on exam days; a community cloud is like buses **shared by several colleges** of one university.

8. Public Clouds

Q. What is a public cloud? Explain its characteristics with examples. **[6 Marks]**

A **public cloud** is the first and canonical form of cloud computing: its services are available to **anyone, anywhere, any time** over the Internet.

1. **Structure:** a distributed system of one or more connected datacenters; any customer signs up with credentials and billing details.
2. **Uses:** to minimise IT infrastructure cost and handle peak loads. Small enterprises can start without a large up-front investment; customers grow or shrink their IT with demand — to replace or to extend their own infrastructure.
3. **Multitenancy is fundamental:** each customer gets a separate, isolated virtual environment, which enables monitoring and guaranteed QoS.
4. **QoS management:** much of the software stack monitors resources, bills as per the contract, and keeps a full usage history — full accountability.
5. **Any service level:** Amazon EC2 (IaaS), Google AppEngine (PaaS), SalesForce.com (SaaS).
6. **Scalability:** built to scale on demand and absorb peak loads.
7. **Geographic dispersion:** datacenters can be spread worldwide. Example: AWS has datacenters in the US, Europe, Singapore, and Australia, with **regions** (us-west-1, us-east-1, eu-west-1) priced differently and divided into **availability zones** mapped to specific datacenters.

9. Private Clouds

Q. What is a private cloud? Why do organisations prefer private clouds? With a neat diagram, explain the hardware and software stack of a private cloud. **[8 Marks]**

A **private cloud** keeps the cloud model (on-demand provisioning) **inside an organisation's boundary**, mainly to retain control.

Why not a public cloud?

1. **Loss of control:** the provider controls the infrastructure, the core logic, and sensitive data; government and military agencies will not take this risk.
2. **Data location:** the geography of a datacenter decides which laws apply (e.g., the USA PATRIOT Act gives US agencies broad access to data stored in US territory).
3. **Existing investment:** enterprises with large IT infrastructure want to use and optimise it.

Characteristics:

- Virtual distributed systems on private infrastructure, giving internal users **dynamic provisioning**.
- Instead of pay-as-you-go, usage may be billed proportionally to departments.
- Core business stays in-house; sensitive data does not leave; existing IT is better utilised.
- A cheaper place to test applications before deploying them on public clouds.

Key advantages (Forrester report):

1. **Customer information protection:** in-house security is easier to maintain and trust.
2. **Infrastructure ensuring SLAs:** clustering, failover, replication, monitoring, and disaster recovery sized to application needs.
3. **Compliance with standard procedures and operations:** third-party compliance rules can be met, which may be impossible on public infrastructure.

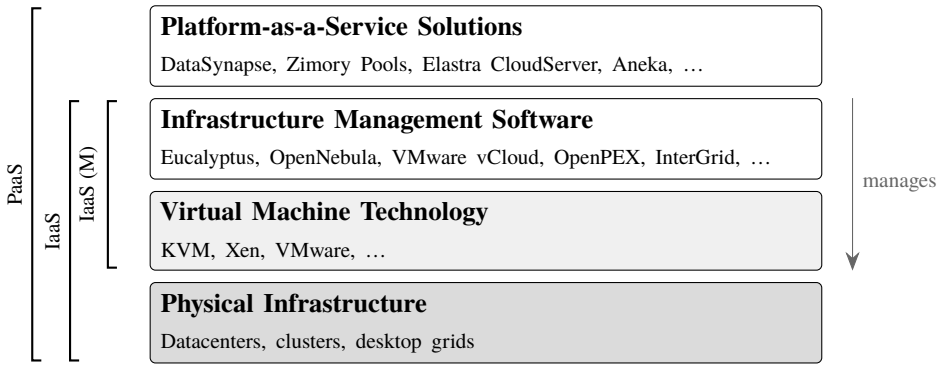


Figure 4.7: Private clouds hardware and software stack (Text Book 2, Fig. 4.4, redrawn)

Implementation stack (bottom to top): PaaS spans the whole private stack; IaaS covers the bottom three layers; IaaS (M) covers only the management software and VM technology.

1. **Physical layer:** existing IT — datacenter, cluster, enterprise desktop grid, or a mix (often heterogeneous).
2. **VM technology:** Xen, KVM, VMware.
3. **Infrastructure management (IaaS):** VMware vCloud (proprietary); Eucalyptus (open, AWS-compatible interfaces, supports Xen/KVM/VMware); OpenNebula (open source, supports KVM/Xen/VMware, integrates third-party IaaS, can use Haizea as the scheduler for VM reservation).
4. **Value-added services on top:** OpenPEX (web-based VM reservation; Xen back end), InterGrid (reservation and multi-domain clouds on top of OpenNebula and Amazon EC2).
5. **PaaS for private clouds:** DataSynapse (application virtualization on VMware), Elastra Cloud Server (configure and deploy distributed application infrastructures), Zimory (automates resource pools on Xen/KVM/VMware; mixes private and public resources; application migration), Aneka (service-oriented platform on datacenters, clusters, and desktop grids; bag of tasks, MapReduce, etc.).

Limitation: compared with public clouds, private clouds can scale elastically on demand **only to a limited extent**.

10. Hybrid Clouds and Cloudbursting

Q. What is a hybrid cloud? With a neat diagram, explain dynamic provisioning and cloudbursting. **[8 Marks]**

A **hybrid cloud** combines a **private cloud** with **public resources**, gaining the best of both: in-house control plus on-demand scale.

Why hybrid clouds?

1. Public clouds have huge capacity but security and administrative pitfalls; most enterprises keep a private cloud because of their existing IT.
2. The major drawback of private clouds is that they cannot scale on demand or handle peak loads efficiently → they **borrow public capacity** when needed.
3. Enterprises keep sensitive data on premises and grow or shrink by provisioning external resources and releasing them; security concerns apply only to the public part, which is used for less-constrained workloads.

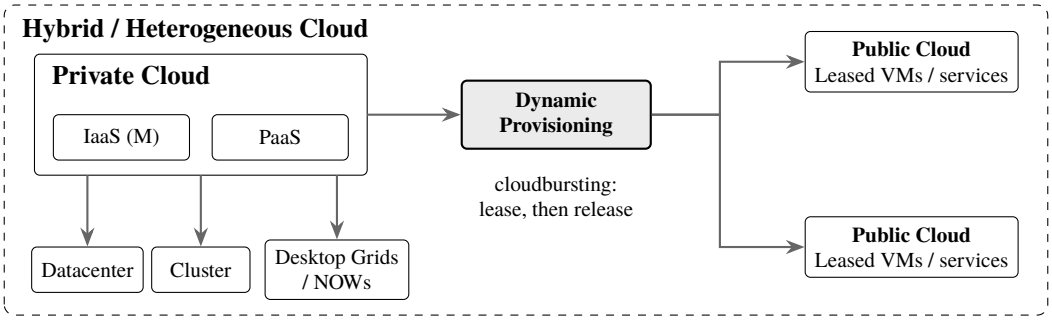


Figure 4.8: Hybrid/heterogeneous cloud overview (Text Book 2, Fig. 4.5, redrawn)

Explanation of the figure: The private cloud runs on its own resources (datacenter, cluster, desktop grids/networks of workstations) and, through **dynamic provisioning**, temporarily extends into one or more public clouds.

Dynamic provisioning and cloudbursting:

1. **Dynamic provisioning** is the key component of a hybrid cloud. Leasing resources temporarily and releasing them is called **cloudbursting**. (The term can also mean a cloud failing under a demand spike; the textbook uses the first meaning.)
2. It applies mostly to IT infrastructure (IaaS): VMs are acquired on demand and then released. Scheduling becomes more complex because it must also **optimise the budget** for rented public resources.
3. **OpenNebula** integrates Amazon EC2 VMs and manages them like local ones; cost-aware scheduling needs an add-on scheduler such as **Haizea**.
4. **InterGrid** is a distributed scheduling engine across peered networks (local cluster, gateway to a public cloud, or both); it allocates VMs considering the user's budget and peering arrangements.
5. **PaaS** is where dynamic provisioning is most common: **Aneka**'s provisioning service works with the scheduler, renting virtual nodes within each application's budget to meet its QoS. Elastra CloudServer and Zimory Pools also support hybrid deployment.

Example: A university exam-results portal runs on its own servers, but on result day it bursts into a public cloud for a few hours and then releases the extra servers.

Conclusion: Hybrid clouds keep sensitive work in-house and use public clouds only for peak demand, paying for that extra capacity only while it is used.

11. Community Clouds

Q. What is a community cloud? With a neat diagram, explain community clouds, their candidate sectors, and their benefits. **[8 Marks]**

A **community cloud** integrates services from different clouds to serve the needs of **one industry, community, or business sector** with shared concerns.
NIST definition (paraphrased): infrastructure shared by several organisations supporting a community with shared concerns (mission, security requirements, policy, compliance); managed by the organisations or a third party; on or off premise.

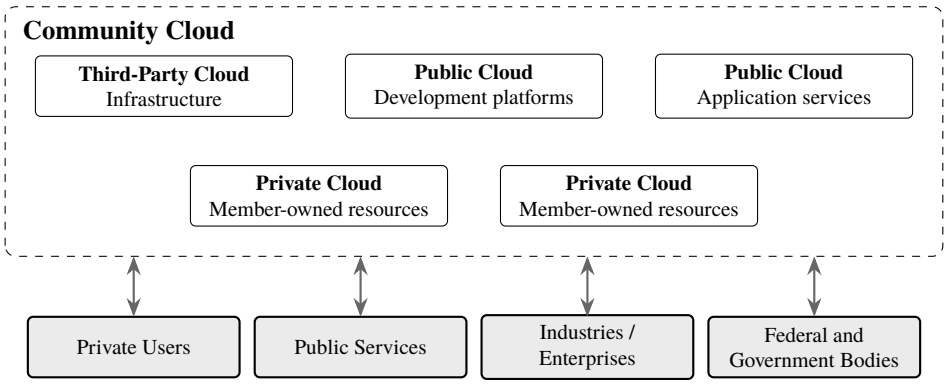


Figure 4.9: A community cloud (Text Book 2, Fig. 4.6, redrawn)

Explanation: Members of one community (private users, public services, industries/enterprises, federal and government bodies) share public, private, and third-party clouds that together form one community cloud.

- It differs from **public** clouds (many users with different needs) and from **private** clouds (services delivered within the owning institution).
- It usually spans **multiple administrative domains**: government bodies, enterprises, research organisations, and public providers all contribute resources.

Candidate sectors:

1. **Media industry:** collaborative digital content production — large data movement, compute-heavy rendering, and complex workflows across partners.
2. **Healthcare:** share knowledge globally without exposing sensitive data; patient data in a private cloud, shared infrastructure for non-critical services and hospital automation.
3. **Energy and other core industries:** bundle management, deployment, and orchestration across many providers and vendors; an open, fair market.
4. **Public sector:** legal/political limits on public clouds; multi-agency processes (business-to-administration, citizen-to-administration), e.g., invoice approval, infrastructure planning, public hearings.
5. **Scientific research:** science clouds, where the shared interest is scientific computing.

Alternative view — community cloud as a digital ecosystem: formed from the under-used resources of members' machines; each member can be a consumer, producer, or coordinator. Its benefits are:

1. **Openness:** no dependency on vendors; fair competition.
2. **Community:** grows by expanding its user base, so it is more scalable.
3. **Graceful failures:** no single provider, so no single point of failure.
4. **Convenience and control:** owned by the community; decisions are taken democratically.

5. **Environmental sustainability:** smaller carbon footprint by using idle resources; it grows and shrinks with community demand.

12. Comparison of the Types of Clouds

Q. Differentiate between public, private, hybrid, and community clouds with examples. **[8 Marks]**

Aspect	Public	Private	Hybrid	Community
Users	Anyone, over the Internet	Members of one institution	One institution, plus public resources when needed	Organisations of one sector with shared concerns
Administrative domain	Provider's	Institution's own	Private + public	Multiple domains
Control	Provider controls	Full in-house control	In-house control of the private part	Shared; decided by the community
Scalability	Very high, on demand	Limited to own resources	High (cloudbursting)	Grows with the community
Cost model	Pay-as-you-go	Own investment; billed to departments	Own investment + pay for bursts	Shared among members
Key concern	Loss of control, data location	Limited elasticity	Complex, budget-aware scheduling	Coordination among members
Examples	Amazon EC2, Google AppEngine, SalesForce.com	Eucalyptus, OpenNebula, VMware vCloud in-house	OpenNebula + Amazon EC2, InterGrid, Aneka	Media, healthcare, public sector, science clouds

13. Economics of the Cloud

Q. Explain the economics of cloud computing. What is capital cost? **[6 Marks]**

The main drivers of cloud computing are **economy of scale** and simple software delivery and operation; the biggest benefit is financial — the **pay-as-you-go** model.

Cloud computing allows:

1. Reducing the **capital costs** of IT infrastructure.
2. Eliminating the **depreciation / lifetime costs** of IT capital assets.
3. Replacing **software licensing with subscriptions**.
4. Cutting the **maintenance and administrative costs** of IT resources.

Capital cost is a one-time, usually up-front cost of buying an asset used to produce goods or render services; it contributes to profit over the long term.

- IT infrastructure and software are capital assets for every enterprise, IT-focused or not, because all businesses use IT for payroll, CRM, ERP, product tracking, and inventory.
- Good practice is to keep capital costs low; cloud computing turns them into **operational costs** that are paid as the resources are used.

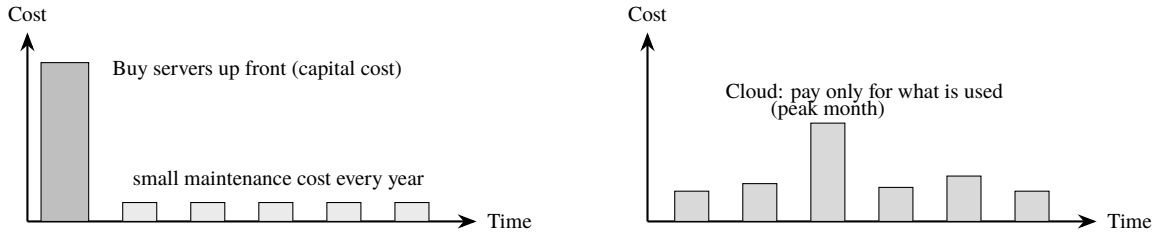


Figure 4.10: Buying infrastructure (capital cost) versus renting from the cloud (pay-as-you-go)

14. Quick Revision

Key Points to Remember

- **Layers bottom-up:** cloud resources → core middleware (VM management, hosting platforms) → user-level middleware → user applications; adaptive management spans all.
- **IaaS** = hosting + resources; **IaaS (M)** = management layer only. **PaaS** = user-level middleware (+ infrastructure); **Pure PaaS** = middleware only. **SaaS** = applications.
- **IaaS reference model:** web interface, infrastructure management software (scheduler at the centre), physical infrastructure.
- **PaaS essentials:** runtime framework, abstraction, automation, cloud services. Main risk: **vendor lock-in**.
- **SaaS:** one-to-many, multitenant; evolved from ASPs; SaaS 2.0 = composing services for business goals.
- **Cloud types by administrative domain:** public, private, hybrid (cloudbursting), community (shared concerns, multiple domains).
- **Economics:** pay-as-you-go; capital costs become operational costs.

15. Question Bank

Sl.	Question	Marks	Section
1	What is cloud computing? Explain the meaning of the term with examples.	6	1
2	With a neat diagram, explain the cloud computing reference model (Fig. 4.1).	8	2
3	Explain how the layers map to IaaS, PaaS, and SaaS. Classify cloud services with examples. What is XaaS?	8	3
4	What is IaaS/HaaS? Explain its core technology and benefits.	6	4.1
5	With a neat diagram, explain the IaaS reference implementation and the role of each management component (Fig. 4.2).	8	4.2
6	Differentiate complete IaaS and IaaS (M) with examples.	6	4.2
7	With a neat diagram, explain the PaaS reference model, its development models, and delivery options (Fig. 4.3).	8	5.1
8	Classify PaaS offerings (PaaS-I, PaaS-II, PaaS-III) with examples (Table 4.2).	6	5.2
9	Explain the essential characteristics of PaaS. What is vendor lock-in?	8	5.3
10	What is SaaS? Compare ASPs, SaaS, and SaaS 2.0. Give examples of SaaS.	8	6
11	What is multitenancy? Why is it important for SaaS and public clouds?	6	6, 8
12	What are the types of clouds? Explain the classification based on administrative domain.	6	7
13	What is a public cloud? Explain its characteristics with examples.	6	8
14	Explain the private cloud software stack (Fig. 4.4) and the advantages of private clouds.	8	9
15	What is cloudbursting? Explain dynamic provisioning in hybrid clouds (Fig. 4.5).	8	10
16	List the candidate sectors and benefits of community clouds (Fig. 4.6).	8	11
17	Differentiate public, private, hybrid, and community clouds with examples.	8	12
18	Explain the economics of the cloud. What is capital cost?	6	13

Module 1: Modern Computing Approaches

Topic 5: Edge Computing

Introduction, need for edge computing, edge computing architecture, cloud vs. edge computing, applications

Course	Modern Computing and Python Programming (26ESCC106)
Module	Module 1 — Modern Computing Approaches
Topic coverage	Edge Computing — Introduction, Need, Architecture, Cloud vs. Edge, Applications
Position in module	After Cloud Computing; last topic of the module
Mapped CO	CO1 — Describe modern computing approaches and their characteristics
Bloom's level	L1 — Remember, L2 — Understand

1. Introduction

1.1 Definition

Edge computing means performing computation close to where the data is produced, instead of sending all of it to a distant data centre [4, 1]. The "edge" is the outer boundary of the network: the place where cameras, sensors, machines, vehicles and phones meet the network, as opposed to the centre, where the large cloud data centres sit.

Two terms are worth fixing before going further, because the whole topic turns on them.

- **Latency** is the time taken for data to travel from where it is produced to where it is processed, and for the answer to come back. It is measured in milliseconds (ms), and one millisecond is a thousandth of a second.
- **Bandwidth** is how much data a network link can carry per second, measured in megabits per second (Mbps). A link with higher bandwidth carries more data in the same time.

In the cloud model studied in the previous topic, data travels from the device to a data centre that may be hundreds or thousands of kilometres away, is processed there, and the result travels back. Edge computing places a computer — a gateway, a small server, or the device itself — beside the data source, so that most of the work finishes locally and only a small part of the data ever travels to the cloud. Figure 5.1 shows this basic arrangement.

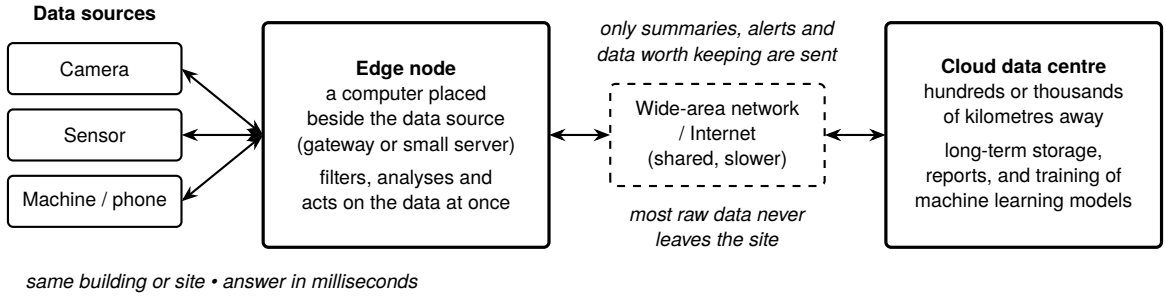


Figure 5.1: *Basic working model of edge computing*

Edge computing does not replace cloud computing. The two work together: the edge handles what must be fast, private or independent of the network, and the cloud handles what needs large storage, heavy computation or a view across many sites.

1.2 Where the Edge Is

"The edge" is not one fixed place. Depending on the application the computing may sit at any of the following points, and a real system often uses more than one of them [1].

- **On the device itself** — a camera that recognises a number plate inside the camera, or a watch that counts steps without help from a phone
- **On a gateway at the site** — one small computer in a factory, farm or hospital that serves all the devices around it
- **At the network operator's premises** — a small data centre at a mobile base station, serving every user connected to it. This arrangement is called multi-access edge computing [7]

The term **fog computing** is often met alongside edge computing. It describes the same idea with the emphasis on an intermediate layer of gateways and regional nodes between the devices and the cloud, rather than on the devices themselves [6]. For this course the two may be treated as the same approach seen from slightly different distances.

2. Need for Edge Computing

Cloud computing solved the problem of obtaining capacity without owning it. It did not solve the problem of distance. Sending data to a distant data centre takes time, consumes network capacity, and fails when the network fails. The reasons for computing at the edge follow from these facts [4, 5].

- **Latency** — some decisions cannot wait for a round trip to a data centre. A self-driving vehicle that must brake, a robot arm that must stop before it injures someone, or a game that must respond to a movement, needs an answer within a few milliseconds. A round trip to a cloud region typically costs tens of milliseconds, and more when the network is busy.
- **Bandwidth** — sending everything is often simply too much data. One high-definition camera produces roughly 4 Mbps of video, so a campus with 100 cameras would need about 400 Mbps of upload capacity, continuously, only to send video that is almost entirely uneventful. An edge node can watch all 100 streams locally and send on only the few seconds that matter.
- **Intermittent connectivity** — ships, mines, farms, oil rigs and rural clinics do not have a dependable link to the Internet. Work that depends on reaching the cloud stops when the link does; work done at the edge continues and synchronises later.
- **Privacy and regulation** — medical records, classroom footage and factory process data may be sensitive, or may be required by law to stay within the premises or the country. Processing locally and sending only anonymous summaries keeps the raw data where it belongs.
- **Cost** — network transfer and cloud storage are charged by volume, so discarding uninteresting data at the edge reduces both bills.
- **Scale** — the number of connected devices grows faster than the capacity of the links to the cloud, so a design that forwards everything to the centre becomes harder to sustain as devices are added [1, 3].

3. Edge Computing Architecture

An edge computing system is usually described as a set of tiers, or layers, arranged by distance from the data [1, 6]. Figure 5.2 shows the four tiers and what each one does. It is read from the bottom, where the data is created, to the top, where the cloud sits.

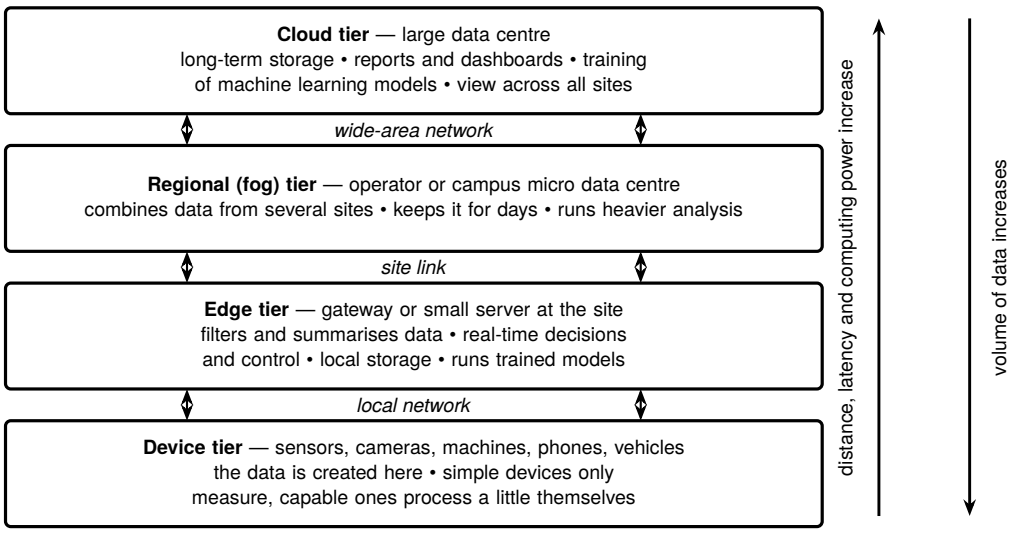


Figure 5.2: *Architecture of an edge computing system: four tiers, from the devices that create the data to the cloud*

3.1 The Tiers

- **Device tier** — the sensors, cameras, machines, vehicles and phones that produce the data. A simple sensor only measures and reports; a more capable device performs a little processing itself, such as discarding a reading that has not changed.
- **Edge tier** — one or more computers at the site: a gateway, an industrial PC or a small server. This tier receives data from the devices over a local network, filters and summarises it, takes the decisions that must be immediate, keeps a short history locally, and forwards what is worth forwarding.
- **Regional or fog tier** — a micro data centre belonging to the organisation or to the network operator and serving several sites. It combines their data, keeps it for days or weeks, and runs analysis that is too heavy for a gateway. Small systems often leave this tier out and connect the edge tier directly to the cloud.
- **Cloud tier** — the large data centre studied in the previous topic. It provides effectively unlimited storage and computing power, a view across every site, and the place where machine learning models are trained before being sent down to the edge to be used.

3.2 How Work Is Divided Between the Tiers

Two quantities change as one moves up the diagram, and the design follows from them. The volume of data is largest at the bottom, because every device produces data continuously, and it shrinks at each tier as data is filtered and summarised. Computing power, storage and distance all grow towards the top. The working rule is therefore simple: perform a task at the lowest tier that is able to perform it, and send upward only what the higher tier genuinely needs [4].

A traffic camera illustrates the division. The camera captures video (device tier). The edge node watches that video, recognises a vehicle crossing a red light and raises an alert within a fraction of a second, discarding the rest (edge tier). Each evening it uploads the recorded violations and a count of vehicles per hour to the cloud, which stores them, prepares reports for the whole city, and periodically trains an improved recognition model that is then sent back to the edge nodes (cloud tier).

4. Cloud versus Edge Computing

The two are partners rather than rivals, but they differ in almost every practical respect [4, 2].

Aspect	Cloud computing	Edge computing
Location of processing	Large data centres, far from the user	Close to where the data is produced
Typical response time	Tens of milliseconds or more	A few milliseconds
Computing power	Very large and elastic	Limited by what fits at the site
Storage	Effectively unlimited	Small; usually a short local history
Data sent over the network	All of it	Only summaries and important events
Works without the Internet	No	Yes, for local tasks
Privacy of raw data	Leaves the premises	Can stay on the premises
Cost pattern	Pay for capacity, transfer and storage used	Pay once for the equipment, then little transfer cost
Scope of view	Every site together	One site or one device
Suitable work	Training models, long-term storage, reports across sites	Immediate decisions, filtering, local control
Management	One provider manages the hardware	Many small units in many places must be managed

5. Applications

- **Industrial automation** — machines on a production line are monitored for vibration, temperature and current, and an edge node stops a machine or raises a maintenance alert before a fault becomes a breakdown
- **Autonomous and assisted driving** — a vehicle interprets its cameras and sensors on board, because braking cannot wait for a reply from a data centre [5]
- **Video surveillance and traffic management** — cameras and edge nodes detect events locally and send only the relevant clips instead of streaming everything
- **Healthcare** — bedside monitors raise alarms locally, and patient data is summarised before anything leaves the hospital
- **Agriculture** — soil, weather and crop-health sensors are processed on a farm gateway, which keeps working where network coverage is poor
- **Smart cities and utilities** — street lighting, water distribution and electricity meters are controlled locally, with readings summarised for the central system
- **Retail** — shelf cameras, billing counters and stock sensors are processed in the store, so that it continues to operate even if the link to the head office is lost
- **Content delivery and gaming** — copies of popular videos, pages and game servers are kept near users so that they load quickly and the central servers carry less load

6. Advantages and Limitations

Advantages

- Responses in milliseconds, because the data does not travel far
- Much less network traffic, and lower transfer and storage cost
- Continues to work when the connection to the cloud is lost

- Sensitive data can be kept within the premises
- Capacity grows naturally as devices and sites are added

Limitations

- Computing power, memory and storage at a site are limited compared with a data centre
- Many small units spread over many locations are harder to update, monitor and repair than one data centre
- Physical security is weaker: equipment sits in factories, streets and vehicles rather than in a guarded building
- Devices and gateways from different makers must be made to work together, and standards are still settling
- Only a local view is available, so conclusions that need data from every site still require the cloud

7. Review Questions

CO1 — Describe modern computing approaches and their characteristics
Bloom's Level L1 — Remember, L2 — Understand

1. Define edge computing. Where exactly is "the edge" in a computing system?
2. Define latency and bandwidth, and state the unit in which each is measured.
3. With a neat diagram, explain the basic working model of edge computing.
4. State any four reasons why edge computing is needed.
5. Explain, with one example each, how edge computing helps when the network connection is unreliable and when the data is sensitive.
6. With a neat diagram, explain the architecture of an edge computing system, naming each tier and stating what it does.
7. What is a gateway in an edge computing system, and what work does it perform?
8. Differentiate between cloud computing and edge computing with respect to location of processing, response time, storage and network usage.
9. Explain why edge computing is described as a partner of cloud computing rather than a replacement for it.
10. What is fog computing, and how is it related to edge computing?
11. List any five applications of edge computing, stating in each case why the processing must be done locally.
12. State any four limitations of edge computing.
13. Describe how a traffic camera system would divide its work between the device, edge and cloud tiers.

8. References and Further Reading

Items 1 to 3 are the prescribed books for this module; chapter numbers are indicative. Items 4 to 6 are the papers from which the definitions and the tiered architecture are taken, and item 7 is the standards site for edge computing in mobile networks.

[1] P. Lea, *IoT and Edge Computing for Architects*, 2nd Edition, Packt Publishing, 2020 — Reference Book 1 for this course; chapters on edge computing and edge architecture.

[2] R. Buyya, C. Vecchiola, S. T. Selvi, S. Poojara and S. N. Srirama, *Mastering Cloud Computing*, 2nd Edition, Morgan Kaufmann, 2026 — Text Book 2 for this course; used here for the cloud side of the comparison.

[3] K. Hwang, G. C. Fox and J. J. Dongarra, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, Morgan Kaufmann, 2012 — Text Book 1 for this course; used here for sensors, devices and the growth in the number of connected things.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges", *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[5] M. Satyanarayanan, "The Emergence of Edge Computing", *IEEE Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[6] F. Bonomi, R. Milito, J. Zhu and S. Addepalli, "Fog Computing and Its Role in the Internet of Things", *Proceedings of the First MCC Workshop on Mobile Cloud Computing*, ACM, 2012, pp. 13–16.

[7] ETSI, Multi-access Edge Computing (MEC). https://www.etsi.org/technologies/multi-access-edge-computing