

1. INTRODUCTION TO DISTRIBUTED SYSTEM

1.1 Introduction to Distributed Systems

1.2 Examples of Distributed Systems

1.3 Main Characteristics

1.4 Advantages and Disadvantages of Distributed System

1.5 Design Goals

1.6 Main Problems

1.7 Models of Distributed System

1.8 Resource Sharing and Web Challenges

1.9 Types of Distributed System: Grid, Cluster, Cloud

1.1 Introduction to Distributed Systems

1.1.1 Definition: A distributed system is a network of independent computers that work together to achieve a common goal, appearing to the users as a single coherent system.

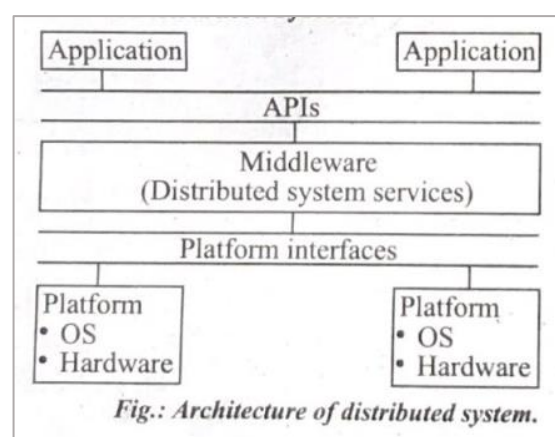
1.1.2 Key Features:

1. **Scalability:** Distributed systems can easily scale horizontally by adding more machines, allowing the system to handle more load and increase computational power.
2. **Fault Tolerance:** These systems are designed to handle failures gracefully. If one machine fails, others can take over its tasks, ensuring continuous operation.
3. **Concurrency:** Multiple processes can run simultaneously across different machines, improving efficiency and performance.
4. **Resource Sharing:** Resources such as files, printers, and storage can be shared across the network, improving utilization and efficiency.
5. **Reliability:** With redundancy and replication, distributed systems can provide higher reliability and availability compared to single-system counterparts.
6. **Heterogeneity:** Distributed systems can integrate various types of hardware, software, and network architectures, allowing diverse systems to work together seamlessly.

1.1.3 Architecture:

It comprises 3 main layers:

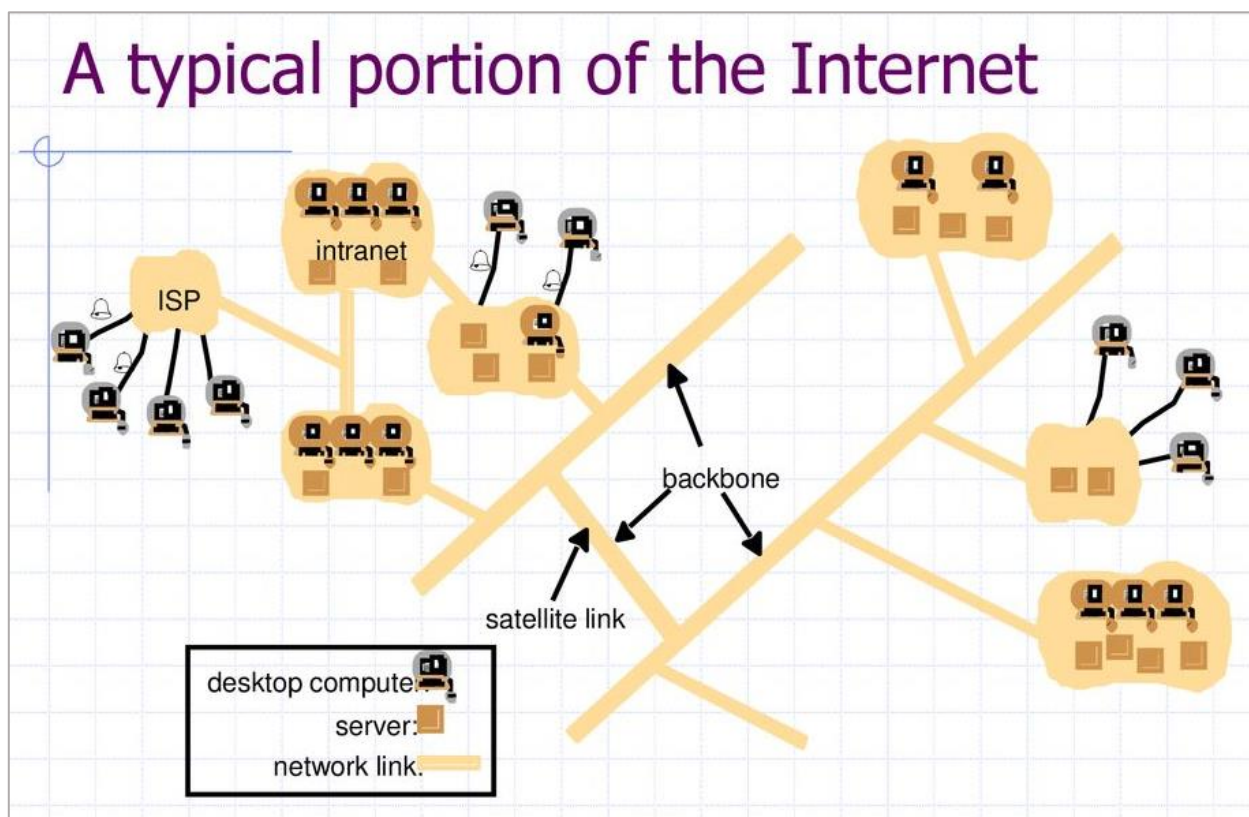
- a. **Hardware Layer:** Nodes (servers, desktops, laptops and mobile devices), Networking Equipment (routers, switches and hubs) and Storage Systems (Network-Attached Storage-NAS or Storage Area Networks-SAN)
- b. **Software Layer:** OS, Distributed Applications and Security Protocols (TCP/IP or HTTP)
- c. **Middleware Layer:** Communication Protocols (Message Passing Interfaces-MPI or



1.2 Examples of Distributed Systems

1.2.1 The Internet

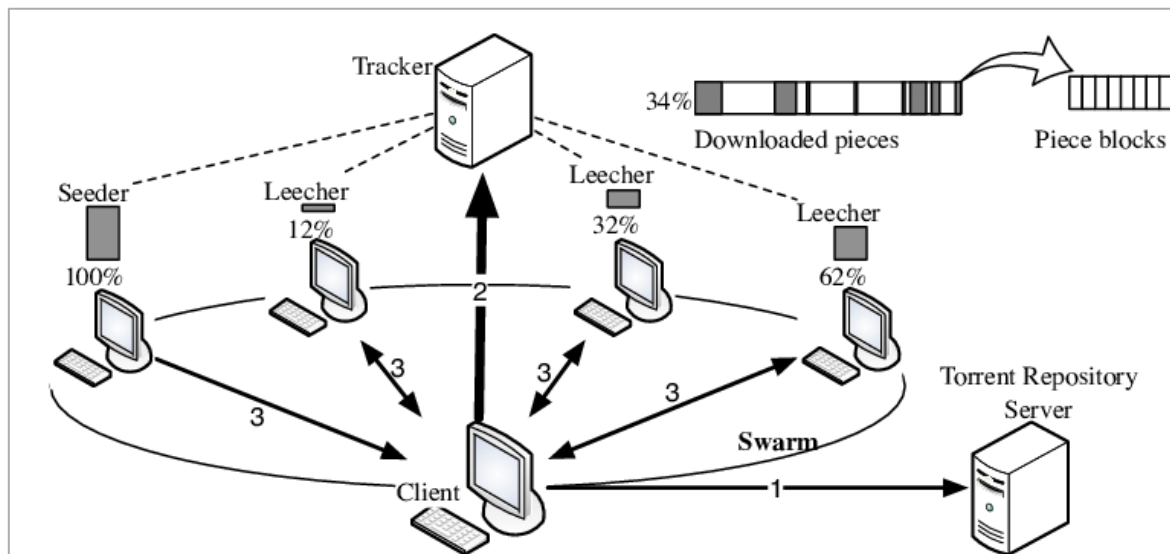
- **Global System:** The Internet is a **global network** of **interconnected computers** and **servers**.
- **Protocols:** Uses protocols like **HTTP, TCP/IP** for **communication** between devices.
- **Connection:** The devices are connected by copper wires, fiber-optic cables and/or wireless connection.
- **Web Services:** Users are able to use services like the WWW, email and file transfer.
- **Working:**
 - When a user requests a webpage, the request is sent to a web server using the HTTP protocol.
 - The server processes the request and sends back the webpage data, which is then displayed in the user's web browser.



1.2.2 Bit Torrent

- **Protocol:** BitTorrent is a peer-to-peer (P2P) **file-sharing protocol**.
- **Decentralization:** **Files are distributed across multiple peers** rather than stored on a central server.
- **Swarming:** **Peers download pieces of a file** from **multiple sources** simultaneously
- **Resilience:** The system is resilient to failures; if one peer goes offline, others can still provide the file parts.

- **Working:**
 - BitTorrent breaks down large files into smaller pieces.
 - When a user downloads a file, they simultaneously download pieces from multiple peers (other users who have the file) and upload pieces they already have to others.
 - This swarming technology maximizes download speeds and distributes the load across many users.
- **Components:** Peers, seeders (users with the complete file), trackers (servers that coordinate the file-sharing process), and torrent files (metadata about the file and tracker).
- **Communication:** Peers communicate directly with each other to exchange file pieces, often facilitated by trackers.

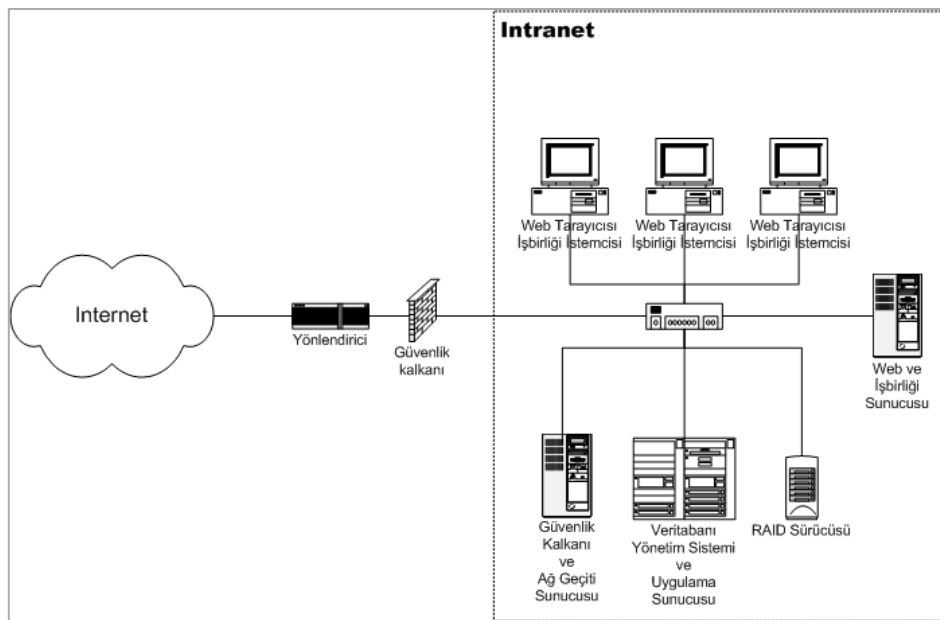


1.2.3 Mobile and Ubiquitous Computing

- **Mobile Applications:** Apps that leverage distributed computing for functionality, such as cloud storage and processing (e.g., Google Drive, Dropbox).
- **Context-Awareness:** Systems that use sensors and data from the environment to provide services (e.g., smart home devices, fitness trackers).
- **Working:**
 - Mobile and ubiquitous computing involves devices that can sense and respond to their environment.
 - Mobile apps connect to cloud services for data storage and processing, while ubiquitous computing devices (like smart home devices) use sensors and network connections to provide context-aware services.
- **Components:** Mobile devices, sensors, cloud servers.
- **Communication:** Devices communicate through wireless networks (Wi-Fi, Bluetooth, cellular networks) and often use cloud services for data processing and storage.

1.2.4 Intranets

- **Internal Network:** Intranets are private networks used within organizations for secure communication and resource sharing.
- **Connection:** An intranet is connected to the internet via a router, enable the users inside the intranet to make use of web services.



1.3 Main Characteristics

- A. **Concurrency:** Multiple processes can run simultaneously across different machines, improving efficiency and performance.
- B. **No Global Clock:** Distributed systems **communicates** messages through **message passing**, so there is no need for a global clock system. In primitive systems, global clock must be synchronized but as there's no correct global timing system, it is difficult to synchronize.
- C. **Independent Failure Node:** Failure of a single node does not hamper the entire system. Other nodes can still run their processes.

1.4 Advantages and Disadvantages of Distributed System

A. Advantages:

- a. **Better price to performance ratio** (cost effective way to increase computing power)
- b. Offers high **Fault Tolerance** (System does not fail even when a single node crashes)
- c. **Efficient processing speed** due to load distribution.
- d. **Scalability** (Computing power can be increased by adding more devices)
- e. **Resource Sharing** (Resources such as files, printers, and storage can be shared across the network, improving utilization and efficiency)

B. Disadvantages:

- a. **Difficult to implement**
- b. Exchange of information between components require coordination creating **Processing Overheads**.
- c. **Security** may be compromised.

1.5 Design Goals/Challenges/Issues

1.5.1. Heterogeneity

Challenge: Distributed systems often consist of diverse hardware, operating systems, network protocols, and programming languages.

Solution: The Internet communication protocols mask the difference in networks and middleware can deal with the other differences.

1.5.2. Openness

Challenge: Being Open means that the system can be extended or re-implemented/integrated with other systems. Distributed Systems should be extensible, interoperable and integrable.

Solution: Use open standards and protocols to ensure compatibility with other systems.

1.5.3. Security

Challenge: Distributed systems are vulnerable to various security threats, including unauthorized access (Confidentiality), data corruption or alteration (Integrity), and denial of service (Availability)

Solution: Use encryption to protect shared resources. User authentication mechanisms to protect against unauthorized access. Regularly update and patch the system.

1.5.4. Scalability

Challenge: Designing systems that can scale horizontally (adding more machines) or vertically (upgrading existing machines) is challenging, especially while maintaining performance and reliability.

Solution: Design the system to be modular and decomposable, allowing for horizontal scaling by adding more nodes. Use load balancers to distribute workloads evenly across multiple servers.

1.5.5. Failure Handling

Challenge: Distributed systems are prone to partial failures where some components may fail while others continue to function. The system should detect, tolerate, and recover from failures without significant disruption to overall operations

Solution: Use redundancy and replication to ensure that data is available even if some components fail. Adopt monitoring and alerting system to notify the system in case of any component failure.

1.5.6. Concurrency

Challenge: Multiple processes or threads running simultaneously can lead to conflicts and inconsistencies if not properly managed. Coordinating access to shared resources and ensuring consistent data states in the presence of concurrent operations is complex.

Solution: Use synchronization mechanisms (e.g., locks, semaphores) to manage access to shared resources.

1.5.7. Transparency

Challenge: The system should hide its distributed nature from users and applications, and act as a single coherent system to its users.

Solution: Use middleware to maintain transparency

Layers:

a. Access Transparency

Users and applications interact with resources without knowing the access method.

Example: In a distributed file system, users access files using a standard interface (like a file explorer), regardless of whether the file is local or on a remote server.

b. Location Transparency

Users and applications access resources without knowing their physical or network location.

Example: When accessing a website, the URL doesn't reveal the physical location of the web server. Users get the same experience regardless of the server's location.

c. Migration/Mobility Transparency

Resources can move within the system without affecting users or applications.

Example: Virtual machines (VMs) in a cloud environment can be migrated to different physical hosts without disrupting the applications running on them.

d. Replication Transparency

Users and applications are unaware of the replication of resources.

Example: A distributed database may replicate data across multiple servers to ensure high availability. Users see a single database instance and are unaware of the underlying replication.

e. Concurrency Transparency

Multiple users or processes can interact with shared resources without interference.

Example: In an online banking system, multiple users can access their accounts and perform transactions simultaneously without affecting each other's operations.

f. Failure Transparency

The failure of certain components in the system are concealed from the users and application allowing them to complete their task despite the failure.

Example: In an electronic mail system, the failure for message sending is hidden from the user and is masked by attempting to retransmit the message until they are successfully delivered.

g. Performance Transparency

The system optimizes performance automatically without user intervention.

Example: Caching mechanism done to reduce latency and improve load times without user intervention.

1.6 Main Problems

1.7 Models of Distributed System

1.7.1. Architectural Model

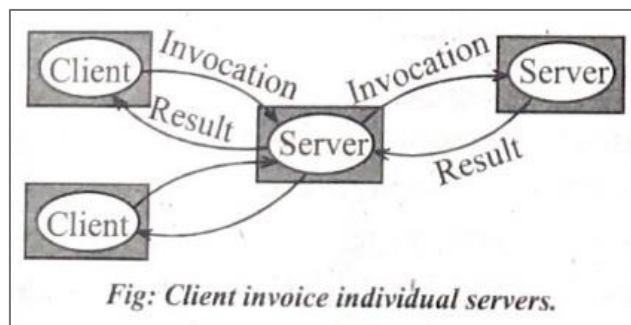
It is concerned with the placement of the components and their relationship/interconnection.

It describes how components interact, communicate, and coordinate to achieve the system's goals

i. Client-Server Model

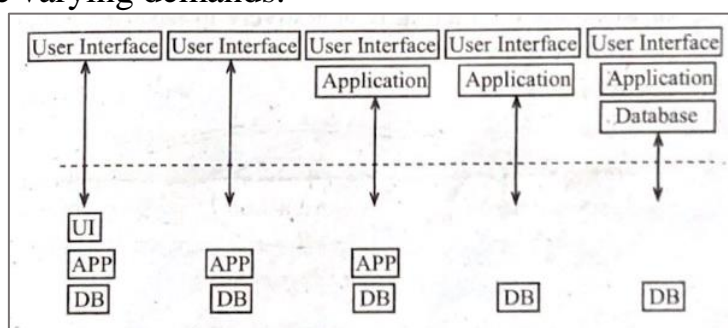
The client-server model is a common architecture in distributed systems where clients request services and resources from centralized servers. Clients act as the front-end interface for users, while servers provide back-end processing and data management. This model allows for centralized control, resource management, and simplified maintenance but can introduce bottlenecks and single points of

failure if not properly managed. Servers can themselves be clients for other servers.



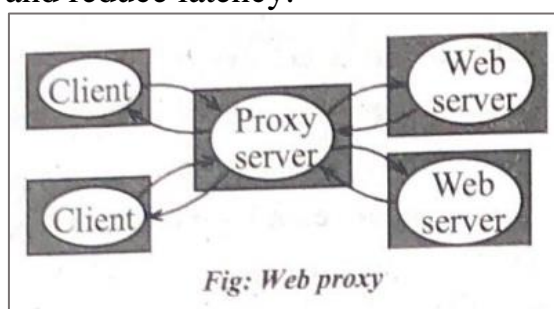
ii. Multi-tiered Architectures

Multi-tiered architectures extend the client-server model by introducing **additional layers**, such as **presentation, application logic, and data storage layers**. Each tier is responsible for specific tasks, promoting separation of concerns and **enhancing scalability** and maintainability. Commonly seen in **web applications**, this architecture allows for **load balancing** and independent scaling of different layers to handle varying demands.



Proxy Servers & Cache:

Caching mechanisms enables frequently accessed data to be stored in small but fast storage entities called cache. Caches are present in each clients or located in proxy servers. **Proxy servers provide a shared cache of web resources** for clients to enhance load time and reduce latency.

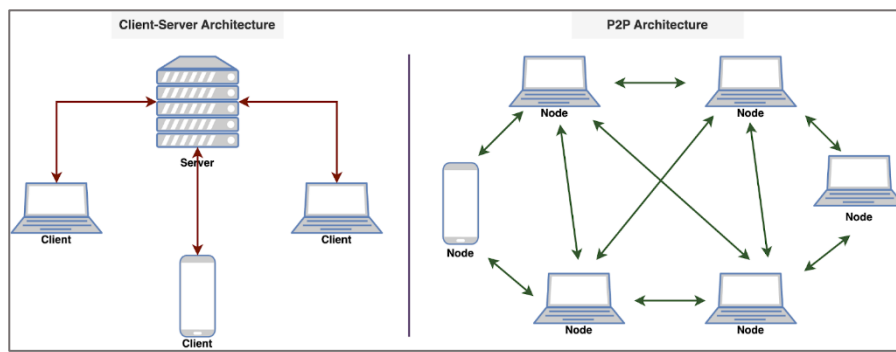


Peer Processes

Peer processes include applications running on peers in a P2P architecture

iii. Peer-to-peer (P2P) Architecture

In a peer-to-peer (P2P) architecture, all nodes in the network act as both clients and servers. Peers share resources, data, and responsibilities, communicating directly with each other without relying on a central server. This decentralized approach improves scalability, resilience, and resource utilization. Examples include file-sharing systems like BitTorrent, where peers simultaneously download and upload file segments, distributing the network load efficiently across all participants.



| Aspect | Client-Server Architecture | Peer-to-Peer (P2P) Architecture |
|----------------------|--|--|
| Structure | Centralized: Clients request services from centralized servers. | Decentralized: All nodes act as both clients and servers. |
| Scalability | Limited by server capacity; requires scaling up servers. | Highly scalable; adding more peers increases capacity. |
| Failure Tolerance | Single points of failure; if the server fails, services are disrupted. | High fault tolerance; peers can redistribute load if one fails. |
| Resource Management | Resources managed by central servers. | Resources shared and managed by all peers. |
| Security | Easier to implement centralized security policies. | Security can be more complex due to the decentralized nature. |
| Performance | Performance can degrade under high load on the server. | Performance improves with more peers sharing the load. |
| Cost | More expensive to implement | Less expensive to implement |
| Use Cases | Web applications, email services, enterprise applications. | File sharing (e.g., BitTorrent), blockchain, decentralized networks. |
| Example Technologies | HTTP/HTTPS, DNS, SQL Databases, REST APIs. | BitTorrent, Bitcoin, IPFS, Gnutella. |

1.7.2. Fundamental Model

Fundamental Model of a system gives the formal definition of the system. It provides a conceptual framework to understand and analyze the behavior and design of distributed systems.

i. Interaction Model

The interaction model describes the communication patterns and data exchange between components in a distributed system. It defines the mechanisms for message passing, synchronization, and coordination among distributed entities. This model helps in understanding the timing, reliability, and ordering of interactions, addressing challenges like network latency, message loss, and concurrency.

ii. Failure Model

The failure model outlines the types and nature of failures that can occur in a distributed system and their impact on the system's behavior. It includes considerations for hardware failures, software bugs, network issues, and process crashes. This model guides the design of fault tolerance mechanisms, such as redundancy, replication, and failover strategies, to ensure system reliability and availability despite failures.

iii. Security Model

The security model defines the policies, mechanisms, and protocols to protect a distributed system against unauthorized access, data breaches, and other security threats. It encompasses authentication, authorization, encryption, and auditing to safeguard data integrity, confidentiality, and availability. This model addresses potential vulnerabilities and ensures that security measures are integrated into all aspects of the system's architecture and operation.

1.8 Resource Sharing and Web Challenges

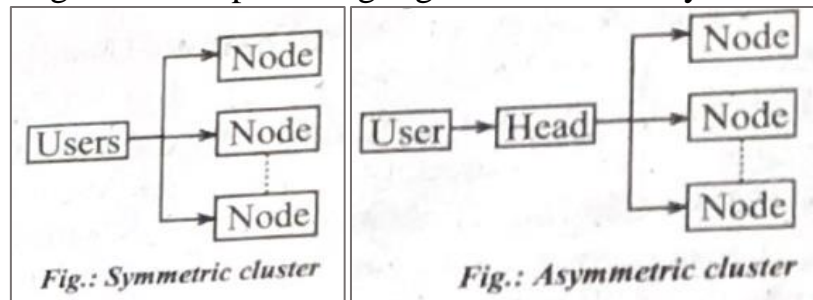
1.9 Types of Distributed System: Grid, Cluster, Cloud

1.9.1. Distributed Computing System

Deployed for solving complex computational problems with the use of parallel processing.

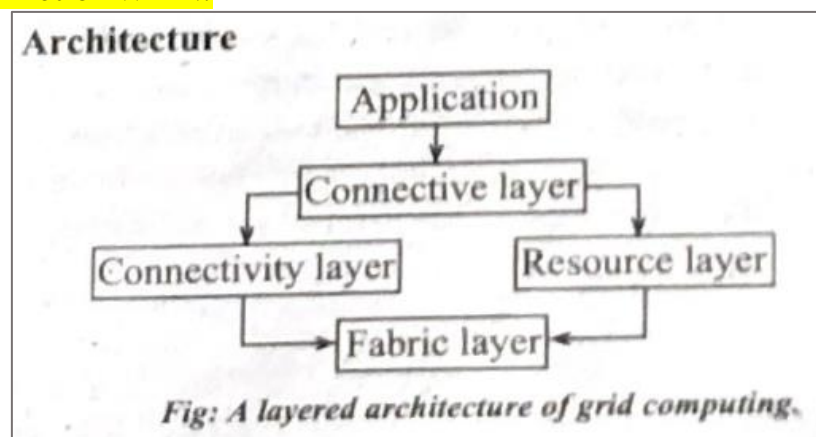
i. Cluster Computing System

Involves nodes working closely together within a single location to perform high-computational tasks. The nodes are connected by a high-speed LAN. In symmetric cluster, user interacts with the nodes directly. In asymmetric cluster, a head node acts as a gateway between the nodes and the user. Here, all traffic must pass through the head providing high level of security.



ii. Grid Computing System

Involves nodes working on a common task by pooling together their resources. Unlike Clusters, Grid systems are often geographically dispersed and connect over the internet or WAN.



Fabric Layer:

Encompasses the **underlying hardware and software resources**, such as computers, storage systems, and networks. This layer is responsible for providing the basic functionalities needed to access and manage these resources. It's the **interface for accessing resources by higher layers**.

Connectivity Layer:

Provides the **communication protocols** and **authentication mechanisms** necessary for establishing secure and reliable connections between the resources. It includes network protocols for data transfer and communication, as well as security protocols for authenticating users and resources.

Resource Layer:

Handles the management and **control of individual resources**, such as processors, storage, and network bandwidth. It includes mechanisms for **resource allocation, monitoring**, and provisioning, ensuring that resources are used efficiently and are available as needed. This layer abstracts the complexities of the underlying fabric and presents a unified interface for accessing and managing resources

Collective Layer:

Operates above the individual resource level, **coordinating multiple resources** to perform higher-level functions. This layer includes services for **resource discovery, scheduling, load balancing, and data management**. It ensures that the grid operates as a cohesive system, optimizing resource utilization and enabling complex, distributed applications to function seamlessly.

Application Layer:

Consists of the application that operate within a virtual organization and uses grid computing environment

iii. Cloud Computing System

Cloud computing is a technology that provides users **with access to computing resources** such as servers, storage, databases, networking, software, and analytics **over the internet**

It's characteristics are:

- **On-Demand Self-Service:** Users can provision computing resources automatically without requiring human intervention from the service provider. This allows for **quick and easy access to services and resources**.
- **Broad Network Access:** Cloud services are **accessible over the internet from a wide range of devices**, including laptops, smartphones, and tablets. This ensures that users can access their applications and data from anywhere at any time, provided they have an internet connection.
- **Multi-Tenancy and Resource Pooling:** Cloud providers use multi-tenancy to **serve multiple customers (tenants) from the same physical resources**. Resources such as storage, processing power, and memory are dynamically allocated and reallocated based on user demand, optimizing resource utilization and cost efficiency.
- **Rapid Elasticity and Scalability:** Cloud computing allows for rapid scaling of resources, both up and down, to meet changing demands. Users can quickly add more resources or reduce them as needed, ensuring that they have the right amount of computing power at any given time without overprovisioning.

- **Measured Service:** Cloud services automatically monitor and report resource usage, providing transparency and enabling a pay-as-you-go model. Users are billed based on their actual consumption of resources, such as computing power, storage space, and data transfer, ensuring cost efficiency and accountability.

Advantages:

- Affordable (pay per use model)
- Adaptable
- Multi-tenant (provides services to multiple customers)
- Reliable
- Scalable
- Secure

Cloud Computing Service Models:

- SaaS (Software as a Service)
- IaaS (Infrastructure as a Service)
- PaaS (Platform as a Service)

Cloud Computing Deployment Models:

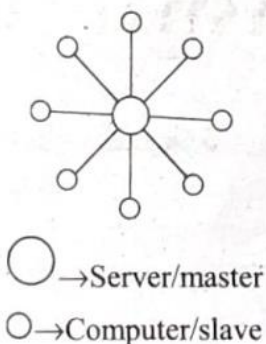


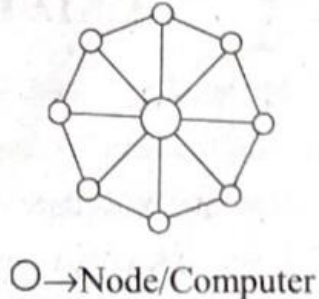

- Public Clouds
- Private Clouds
- Hybrid Clouds

1.9.2. Distributed Information System

A distributed information system integrates and manages data from multiple, distributed sources, providing users with unified access to diverse data sets. These systems often employ middleware to handle the communication, data integration, and consistency across different databases and applications. Distributed information systems are used in environments where data is spread across various location to distribute information across several servers.

1.9.3. Distributed Pervasive System

Distributed pervasive systems, also known as ubiquitous computing systems, involve integrating computing capabilities into everyday objects and environments, allowing for seamless interaction and communication. These systems are characterized by their pervasive nature, often employing sensors, mobile devices, and smart objects to collect and process data continuously. Applications include smart homes, healthcare monitoring, and environmental sensing, where the goal is to enhance user experience and functionality by embedding intelligence into the surrounding environment.

| Aspect | Centralized Systems | Distributed Systems |
|-------------------|---|---|
| Structure | Single central server manages all resources and operations. | Multiple interconnected nodes share resources and tasks. |
| Scalability | Limited by the capacity of the central server. | Highly scalable; can add more nodes to increase capacity. |
| Failure Tolerance | Single point of failure; if the central server fails, the system fails. | High fault tolerance; system continues to function if one or more nodes fail. |
| Performance | Can become a bottleneck under heavy load. | Generally better performance under heavy load due to distribution of tasks. |
| Security | Easier to implement and manage security policies centrally. | More complex security management due to distributed nature. |
| Maintenance | Easier to maintain and update as all resources are centralized. | More challenging to maintain and update due to distribution. |
| Global Clock | Present | Absent |
| Visualization |  <p>  → Server/master  → Computer/slave </p> |  <p>  → Node/Computer </p> |
| Examples | Mainframe systems, traditional web hosting. | Peer-to-peer networks, cloud computing, block-chain. |