Parallel & Distributed Computing Notes

What is Parallel Computing?

Definition: Parallel computing involves multiple processors (or cores) executing tasks simultaneously to speed up processing.

Example: Running a machine learning model on a GPU where multiple cores handle different computations at the same time.

Real-Life Applications of Parallel Computing

1.Space & Aerospace Technology
NASA's Space Simulations – Simulates planetary landings, space weather, and interstellar travel using parallel computing.
Satellite Image Processing – Parallel computing speeds up image analysis for Earth observation and space missions.
Mars Rover Navigation – Uses parallel processing to analyze terrain data and plan safe routes in real-time.

2.Artificial Intelligence & Machine Learning
Deep Learning Training – Neural networks train faster using parallel execution on GPUs.
Self-Driving Cars – Parallel computing helps process sensor data, detect objects, and make split-second driving decisions.

3.High-Performance Computing (HPC) & Supercomputers
Weather Forecasting – Supercomputers simulate climate models, analyzing vast amounts of atmospheric data in parallel.
Scientific Simulations – Used in physics (Large Hadron Collider), chemistry (molecular simulations), and biology (protein folding).

4.Video Games & Graphics Processing
Ray Tracing in Games – Real-time lighting and shadow rendering use parallel processing on GPUs.
Game AI & Physics – NPC behaviors and game physics are calculated in parallel to improve realism.
5.Big Data & Cloud Computing
Search Engines (Google, Bing) – Web crawlers and indexing algorithms use parallel execution across thousands of machines.
Big Data Analytics (Hadoop, Spark) – Distributed frameworks process massive datasets in parallel to extract insights faster.

Short Notes: Parallel computing is ideal for tasks that can be broken into smaller, independent subtasks. It is commonly used in scientific simulations, data analysis, and graphics rendering.

What is Distributed Computing?

Distributed computing is a computing paradigm where multiple computers (nodes) work together over a network to complete a task. Unlike a single system, distributed computing distributes processing across various machines, improving efficiency, scalability, and fault tolerance.

Real-Life Applications of Distributed Computing

1.Cloud Computing (AWS, Google Cloud, Azure)
Distributed servers host applications and data worldwide.
Example: Google Drive allows users to store and access files globally.

2.Search Engines (Google, Bing, DuckDuckGo)
Web searches are processed across multiple servers for speed and accuracy.
Example: Google Search splits queries among thousands of machines.
3.Financial Trading & Blockchain

Distributed ledgers like Bitcoin & Ethereum ensure decentralized transactions.
Example: Stock exchanges use distributed computing for high-frequency trading.
4. Space & Aerospace Technology

NASA's Deep Space Network (DSN) uses distributed systems to communicate with spacecraft.
Example: The James Webb Space Telescope sends massive amounts of data to distributed servers for analysis.
SETI@home: A distributed computing project analyzing radio signals for signs of extraterrestrial life.
5. Autonomous Vehicles & Smart Cities

Tesla's AI and self-driving technology use distributed computing to process sensor data in real-time.
Traffic management systems analyze road data across a distributed network to reduce congestion.
6.Online Gaming & Streaming (Netflix, YouTube, Discord, CSGO)

Game servers distribute loads among multiple machines to handle millions of players.
Streaming platforms use CDNs (Content Delivery Networks) to optimize video delivery.
7.Scientific Research & AI

CERN's Large Hadron Collider (LHC) uses distributed computing to analyze petabytes of physics data.

AI-powered protein folding simulations (e.g., AlphaFold) rely on distributed computing for biological research.

Types of Distributed Systems

1. Client-Server Systems

Central server provides services to multiple clients.
Example: Web applications, online banking, email servers.
2. Peer-to-Peer (P2P) Systems

Each node acts as both a client and a server.
Example: BitTorrent, decentralized networks like IPFS.
3. Cloud Computing Systems

Virtualized resources distributed across data centers worldwide.
Example: Google Cloud, AWS, Microsoft Azure.
4. Distributed Databases

Data is stored and processed across multiple servers.
Example: Google Spanner, Amazon DynamoDB, Apache Cassandra.
5. Distributed Computing Frameworks

Designed to handle large-scale parallel tasks.
Example: Apache Hadoop, Apache Spark, Kubernetes.
6. Real-Time Distributed Systems

Process time-sensitive data from multiple sources.
Example: Air traffic control, space mission control systems, self-driving cars.

Short Notes: Distributed computing is essential for large-scale systems like cloud computing, where resources are spread across multiple locations. It enables fault tolerance and scalability.

Key Differences Between Parallel & Distributed Computing

Parallel Computing: Focuses on speeding up tasks using multiple processors within a single machine.

Distributed Computing: Focuses on coordinating tasks across multiple machines over a network.

Short Notes: Parallel computing is about performance, while distributed computing is about scalability and fault tolerance.

Advantages & Challenges of Distributed Computing

Advantages:

Scalability: Easily add more machines.

Fault Tolerance: If one system fails, others continue.

Cost Efficiency: Uses commodity hardware instead of expensive supercomputers.

Challenges:

Network Latency: Communication delay between nodes.

Security Risks: More exposed to cyber threats.

Synchronization Issues: Managing multiple nodes is complex.

Short Notes: Distributed systems are powerful but require careful design to handle challenges like consistency and security.

System Models & Architectural Models

System models define how computing components interact and communicate. They help in designing efficient and reliable computing architectures.

Physical Model: Deals with the actual hardware and network infrastructure.
Examples:
Single Processor Systems – Traditional computers with a single CPU.
Multiprocessor Systems – Multiple CPUs sharing memory (e.g., multi-core processors).
Cloud Infrastructure – Data centers with virtualized computing resources.

Use Case:
NASA's supercomputers use high-performance clusters to run space simulations.

Architectural Model: Defines the structure of the system, including components and their interactions.

Types of Architectural Model

Client-Server: A centralized model where clients request services from servers.
Example: A website where the browser (client) requests pages from a web server.

Peer-to-Peer (P2P): A decentralized model where peers share resources directly.
Example: BitTorrent, where users share files directly with each other.

Use Case:
Cloud gaming services (e.g., NVIDIA GeForce Now) use the client-server model for streaming games from powerful remote servers.

Fundamental Model: Focuses on logical aspects like communication, failure, and security.

Short Notes: These models help in designing efficient and reliable distributed systems by addressing different layers of abstraction.

Types of Fundamental Model
Failure Types:

Crash Failure: A server stops working unexpectedly.

Omission Failure: A message is lost in transmission.

Byzantine Failure: System behaves unpredictably due to malicious activity.

Security Threats:

Man-in-the-middle attacks: Hackers intercept communication.
Data breaches: Unauthorized access to sensitive data.

Use Case:
Space mission control systems use failure models to handle potential communication delays and hardware failures in deep-space missions (e.g., Mars Rover)

Static vs. Dynamic Interconnection Networks

Multiprocessor systems require interconnection networks for communication between processors. These networks are classified as:

Static Networks: Fixed connections between processors, e.g., Mesh, Hypercube. Suitable for predictable communication patterns.
Dynamic Networks: Connections can be reconfigured, e.g., Crossbar, Multistage Networks. Suitable for flexible communication.
Role of Network Interfaces & Topologies:

Optimize communication by reducing latency.
Maximize bandwidth to enhance efficiency.
Short Notes:

Static networks are simple but rigid.
Dynamic networks are flexible but complex.
Efficient topology improves communication.

Static and Dynamic Interconnection Networks

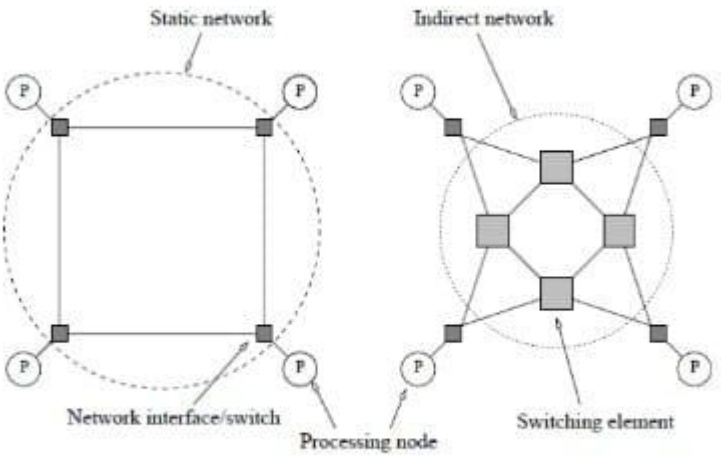


Figure :Classification of interconnection networks: (a) a static network; and (b) a dynamic network.

12

Network Topologies: Pros, Cons, and Examples

Key Takeaways:
Mesh & Crossbar: Best for high-reliability systems where performance and fault tolerance are critical.

Star & Tree: Commonly used in LANs and enterprise networks due to their simplicity and scalability.

Hypercube & Fat Tree: Ideal for supercomputing and cloud data centers where low latency and high bandwidth are essential.

1. Completely Connected (Mesh)
Pros: Every node is connected to every other node, ensuring high reliability and fast communication.

Cons: Expensive to implement and difficult to scale as the number of nodes increases.

Example: Used in military networks and data centers where reliability is critical.

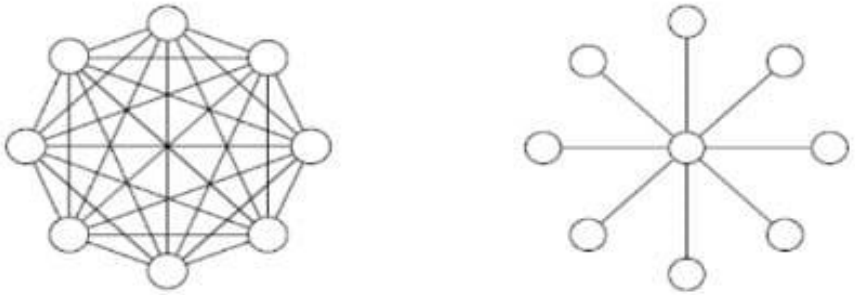
2. Star
Pros: Centralized management makes it easy to set up and scale.

Cons: If the central hub fails, the entire network goes down.

Example: Commonly used in Ethernet LANs and home Wi-Fi setups.

Network Topologies: Completely Connected and Star Connected Networks

✓ Example of an 8-node completely connected network.



(a) (b)

Figure: (a) A completely-connected network of eight nodes; (b) a Star connected network of nine nodes.

3. Bus
Pros: Simple and cost-effective to implement.
Cons: If the main backbone fails, the entire network is affected.

Example: Found in small office networks and older Ethernet systems.

4. Tree
Pros: Scalable and hierarchical, making it suitable for large networks.

Cons: Expensive and complex to implement.

Example: Used in ISP backbones and corporate networks.

5. Crossbar
Pros: High-speed and efficient communication between nodes.

Cons: Very expensive and limited scalability.

Example: Found in supercomputers and high-performance computing (HPC) systems.

6. Multistage
Pros: Cost-effective and fault-tolerant.

Cons: Complex design and implementation.

Example: Used in telephone networks and data centers.

7. Hypercube
Pros: Highly scalable with low latency.

Cons: Difficult to physically implement due to its complex structure.

Example: Ideal for supercomputing applications.

8. Fat Tree
Pros: High bandwidth and low congestion, making it suitable for large-scale systems.

Cons: Expensive to implement.

Example: Widely used in cloud computing and enterprise data centers.

Conclusion:
Network topologies involve trade-offs between cost, scalability, and performance.

The choice of topology depends on the specific application and requirements:

High-reliability systems: Mesh and Crossbar.

LANs and enterprise networks: Star and Tree.

Supercomputing and cloud data centers: Hypercube and Fat Tree.

Memory Models in Parallel Systems

Shared vs. Distributed Memory
Shared Memory: All processors access the same memory space.

Example: A gaming console (PlayStation, Xbox) uses shared memory for processing graphics faster.

Distributed Memory: Each processor has its own memory and communicates via messages.

Short Notes: Shared memory is simpler but less scalable, while distributed memory is more scalable but complex to manage.

Multi-Core Architecture
Definition: Each core executes separate tasks in parallel, improving processing speed.

Example: Modern CPUs (Intel i9, AMD Ryzen) use multi-threading.

Short Notes: Multi-core architectures are the backbone of modern parallel computing, enabling faster and more efficient processing.

Interconnection Networks
Mesh Network: Direct links between nodes, good for high-speed communication.

Bus Network: All devices share a single connection, simple but slow.

Ring Network: Nodes form a ring, data moves in one direction.

Short Notes: The choice of network topology impacts the performance and scalability of parallel systems.

Communication in Distributed Systems

Message Passing Model
Definition: Processes communicate by sending messages instead of using shared memory.

Example: WhatsApp – Messages are exchanged between different devices.

Short Notes: Message passing is essential for distributed systems where shared memory is not feasible.

Remote Procedure Call (RPC)
Definition: Allows a program on one machine to execute code on another machine as if it were local.

Example: A banking app requesting a balance check from a remote server.

Short Notes: RPC simplifies distributed programming but requires careful handling of network issues.

Remote Method Invocation (RMI)
Definition: Similar to RPC but specific to Java applications.

Example: A Java-based e-commerce site retrieving product details from a database server.

Short Notes: RMI is widely used in enterprise applications for seamless communication between distributed components.

Message-Oriented Middleware (MOM)
Definition: Acts as a middle layer between distributed applications to ensure reliable messaging.

Examples: RabbitMQ, Apache Kafka.

Example: Stock trading apps using real-time messaging to update share prices.

Short Notes: MOM is critical for building scalable and reliable distributed systems.

Scalability & Performance Studies

Vertical vs. Horizontal Scaling
Vertical Scaling: Adding more resources (e.g., RAM, CPU) to a single machine.

Example: Upgrading from 8GB RAM to 16GB.

Horizontal Scaling: Adding more machines to a system.

Example: Adding new servers to a data center.

Short Notes: Vertical scaling is limited by hardware, while horizontal scaling offers unlimited growth potential.

Load Balancing & Fault Tolerance
Load Balancing: Distributes traffic across multiple servers to prevent overload.

Fault Tolerance: If one server fails, others take over.

Example: Google Search Engine balances user queries across multiple servers.

Short Notes: Load balancing and fault tolerance are essential for maintaining high availability in distributed systems.

Case Studies
Google Spanner: A globally distributed relational database system.

Amazon DynamoDB: A NoSQL database with auto-scaling and fault tolerance.

Example: Amazon DynamoDB is used by major e-commerce sites to handle millions of users.

Short Notes: These case studies highlight the real-world applications of distributed systems in handling massive scale and complexity.

Exam Questions & Answers

Q. 01 (A)

Question: How does the architecture of an ideal parallel computer, particularly in the context of the Parallel Random Access Machine (P-RAM) model, accommodate different subclasses within the P-RAM framework?

Answer: The architecture of an ideal parallel computer, particularly in the context of the Parallel Random Access Machine (P-RAM) model, accommodates different subclasses within the P-RAM framework by providing a shared memory space that all processors can access simultaneously. The P-RAM model includes subclasses such as EREW (Exclusive Read Exclusive Write), CREW (Concurrent Read Exclusive Write), and CRCW (Concurrent Read Concurrent Write), which define how multiple processors can access memory concurrently. These subclasses help in optimizing the performance and efficiency of parallel algorithms by controlling memory access patterns.

Short Notes: P-RAM is a theoretical model used to design and analyze parallel algorithms. Different subclasses of P-RAM handle memory access conflicts differently, which can impact the performance of parallel computations.

Q. 01 (B)

Question: What are the differences between static and dynamic interconnection networks in multiprocessor systems, and how do network interfaces and network topologies play a role in optimizing communication within these networks?

Answer: Static interconnection networks have fixed connections between processors, such as mesh or hypercube topologies, which are suitable for predictable communication patterns. Dynamic interconnection networks, like crossbar switches or multistage networks, allow flexible connections that can be reconfigured based on communication needs. Network interfaces and topologies play a crucial role in optimizing communication by minimizing latency and maximizing bandwidth, ensuring efficient data transfer between processors.

Short Notes: Static networks are simpler and more predictable but less flexible. Dynamic networks offer greater flexibility and can adapt to varying communication demands but are more complex.

Q. 02 (A)

Question: What are the main types of distributed systems? Also, define some advantages and disadvantages associated with their implementation?

Answer: The main types of distributed systems include client-server systems, peer-to-peer systems, and cloud-based systems. Advantages of distributed systems include scalability, fault tolerance, and resource sharing. Disadvantages include complexity in design, potential security vulnerabilities, and challenges in maintaining consistency and synchronization across the system.

Short Notes: Client-server systems centralize resources, while peer-to-peer systems distribute resources among all nodes. Cloud-based systems offer on-demand resource allocation and scalability.

Q. 02 (B)
Question: What are the key characteristics and consequences of distributed systems? How do different distributed computing models address the challenges posed by such systems?

Answer: Key characteristics of distributed systems include concurrency, lack of a global clock, and independent failure of components. These characteristics lead to challenges such as ensuring consistency, handling partial failures, and managing communication delays. Different distributed computing models, such as the client-server model, peer-to-peer model, and distributed shared memory model, address these challenges by providing mechanisms for coordination, fault tolerance, and efficient communication.

Short Notes: Distributed systems must handle issues like network partitioning and data replication. Models like MapReduce and distributed hash tables (DHTs) are used to manage large-scale data processing and storage.

Q. 03
Question: What are the main differences between the physical, architectural, and fundamental models in distributed computing system design? How do these models contribute to the efficient functioning of distributed systems?

Answer: The physical model in distributed computing system design deals with the actual hardware and network infrastructure. The architectural model defines the structure of the system, including components and their interactions. The fundamental model focuses on the logical aspects, such as communication, failure, and security models. These models contribute to the efficient functioning of distributed systems by providing a clear framework for design, implementation, and optimization, ensuring that the system meets its performance, reliability, and scalability goals.

Short Notes: Physical models consider factors like network latency and bandwidth. Architectural models help in organizing system components and their interactions. Fundamental models address logical challenges like ensuring data consistency and handling failures.

Q. 04 (A)
Question: In a distributed system, what are the key differences between synchronous Remote Procedure Call (RPC) and asynchronous RPC in terms of communication models and their impact on system performance and scalability?

Answer: Synchronous Remote Procedure Call (RPC) blocks the caller until the response is received, ensuring that the caller and callee are synchronized. Asynchronous RPC allows the caller to continue executing other tasks while waiting for the response, improving system performance and scalability by reducing idle time. However, asynchronous RPC can complicate error handling and state management.

Short Notes: Synchronous RPC is simpler to implement but can lead to performance bottlenecks. Asynchronous RPC improves responsiveness but requires careful management of callbacks and state.

Q. 04 (B)
Question: How does Message-Oriented Middleware, such as the Message-Passing Interface (MPI), differ from traditional socket-based communication in distributed systems? Discuss the advantages and disadvantages of each approach in terms of reliability, scalability, and ease of implementation.

Answer: Message-Oriented Middleware (MOM), such as the Message-Passing Interface (MPI), differs from traditional socket-based communication by providing higher-level abstractions for message passing, including features like message queues and guaranteed delivery. Socket-based communication is lower-level and requires more manual management of connections and data transfer. MOM offers advantages in reliability and scalability but can be more complex to implement. Socket-based communication is simpler and more flexible but may require additional effort to ensure reliability and scalability.

Short Notes: MOM is suitable for systems requiring reliable and scalable message delivery. Socket-based communication offers more control but requires more effort to manage.

Q. 05
Question: How does distributed shared memory (DSM) bridge the gap between the simplicity of the shared memory programming model and the scalability benefits of distributed memory systems, and what are the key challenges associated with ensuring data consistency and coherence in a distributed shared memory system?

Answer: Distributed Shared Memory (DSM) bridges the gap between the simplicity of shared memory programming and the scalability of distributed memory systems by providing a virtual shared memory space across multiple nodes. This allows programmers to use familiar shared memory paradigms while benefiting from the scalability of distributed systems. Key challenges in DSM include ensuring data consistency and coherence, as multiple nodes may access and modify the same data simultaneously. Techniques like cache coherence protocols and distributed locking mechanisms are used to address these challenges.

Short Notes: DSM simplifies programming but introduces complexity in managing data consistency. Coherence protocols like MESI (Modified, Exclusive, Shared, Invalid) are used to maintain data consistency across nodes.