Agenda

Parallel and Distributed Processing:

Motivation (Size of data and complexity of processing);

Storing data in parallel and distributed systems:

Shared Memory vs. Message Passing;

Strategies for data access: Partition, Replication, and Messaging.

Distributed Systems:

Motivation (size, scalability, cost-benefit),

Client-Server vs. Peer-to-Peer models,

Cluster Computing: Components and Architecture

1. Motivation: Why Parallel and Distributed Processing?

Challenges Driving Adoption:

- **Exploding Data Size** (Big Data, IoT, AI/ML)
- **Increased Processing Complexity** (Real-time analytics, deep learning)
- **Scalability & Performance Limits** (Single machines can’t handle workloads efficiently)

Benefits:

- ✔ **Faster Computation** (Divide tasks across multiple nodes)
- ✔ **Higher Fault Tolerance** (Redundancy in distributed systems)
- ✔ **Cost-Effective Scaling** (Horizontal vs. vertical scaling)

Summary of Key Differences

Aspect	Shared Memory Architecture	Distributed Memory Architecture
Memory Access	Single shared memory space	Local memory for each processor
Communication	Direct access to shared data	Message passing between processors
Programming Model	Easier, with fewer complexities	More complex, requiring explicit communication
Scalability	Limited scalability due to contention	High scalability with minimal contention
Cache Coherence	Required for shared data consistency	Not applicable as each processor has local memory
Performance Bottlenecks	Contention and synchronization overhead	Network latency and data transfer overhead

2. Storing Data in Parallel & Distributed Systems
Two Fundamental Models:

Shared Memory

All processors access a **common memory space**

Fast but requires **synchronization** (locks, semaphores)

Example: Multi-core CPUs, GPUs

Message Passing

Nodes communicate via **explicit messages**

More scalable but **higher latency**

Example: Distributed systems (Hadoop, MPI)

Trade-offs:

- Shared memory suffers from **contention** (race conditions).
- Message passing adds **communication overhead**.

3. Strategies for Data Access

A. Partitioning (Sharding)

- **Divide data into chunks** stored across nodes (e.g., hash-based, range-based).
- **Pros:** Balanced load, scalable.
- **Cons:** Joins become expensive.

B. Replication

- **Duplicate data** across nodes for fault tolerance & faster reads.
- **Pros:** Redundancy, improved read performance.
- **Cons:** Consistency challenges (CAP theorem).

C. Messaging (Communication)

- **Nodes exchange data** via messages (e.g., MapReduce, MPI).
- **Pros:** Flexible, works in loosely coupled systems.
- **Cons:** Network latency, serialization overhead.

4. Key Takeaways

- **Parallel/Distributed systems** solve scalability & speed challenges.
- **Shared Memory vs. Message Passing** differ in communication style.
- **Partitioning, Replication, and Messaging** optimize data access.

Title: Distributed Systems: Concepts and Architectures
Subtitle: Motivation, Models, and Cluster Computing

Slide 2: Motivation for Distributed Systems

Why Use Distributed Systems?

1.Handling Massive Data (Size)

1. Big Data, IoT, AI/ML require distributed storage & processing.

2.Scalability Needs

1. Vertical scaling (bigger machines) is limited; horizontal scaling (more machines) is cost-effective.

3.Cost-Benefit Advantage

1. Commodity hardware vs. supercomputers.
2. Fault tolerance & high availability.

Visual: Graph showing scalability comparison (single machine vs. distributed).

Slide 3: Client-Server Model

Key Features:

Centralized Control (Server manages resources).

Clients Request, Servers Respond (HTTP, databases).

Examples: Web apps (Frontend + Backend), Cloud services (AWS, Google Cloud).

Pros & Cons:

☑ Pros	✖ Cons
Simple to manage	Single point of failure
Secure & centralized	Scalability limits (server bottleneck)

Visual: Diagram of clients connecting to a central server.

Slide 4: Peer-to-Peer (P2P) Model

Key Features:

Decentralized (No central server; nodes = peers).

Peers Share Resources (Files, compute power).

Examples: BitTorrent, Blockchain (Bitcoin), Skype (earlier versions).

Pros & Cons:

☑ Pros	✖ Cons
Highly scalable	Security risks (malicious peers)
Fault-tolerant (no single point of failure)	Harder to manage

Visual: Mesh network diagram showing P2P connections.

Slide 5: Client-Server vs. P2P Comparison

Aspect	Client-Server	Peer-to-Peer
Control	Centralized	Decentralized
Scalability	Limited by server	Highly scalable
Use Case	Web apps, cloud	File sharing, blockchain

Visual: Side-by-side comparison table.

Slide 6: Cluster Computing

What is a Cluster?

•**Group of connected computers** working as a single system.

•Used for **high-performance computing (HPC), Big Data, AI**.

Key Components:

1. **Nodes** (Individual machines).
2. **Networking** (High-speed interconnects like InfiniBand).
3. **Scheduler** (Manages task distribution, e.g., Kubernetes, YARN).
4. **Storage** (Distributed FS like HDFS, Lustre).

Visual: Cluster architecture diagram.

Slide 7: Cluster Architecture Types

1. High-Performance Computing (HPC) Clusters

•Used for scientific computing (weather modeling, simulations).

•**Example:** Supercomputers (IBM Summit).

2. Load-Balancing Clusters

•Distributes traffic evenly (e.g., web servers).

•**Example:** NGINX load balancer.

3. High-Availability (HA) Clusters

•Ensures minimal downtime (failover mechanisms).

•**Example:** Database clusters (PostgreSQL HA).

Visual: Icons representing each cluster type.

Slide 8: Summary & Key Takeaways

1. **Distributed systems** solve scalability, cost, and data challenges.
2. **Client-Server** (centralized) vs. **P2P** (decentralized) serve different needs.
3. **Cluster computing** enables HPC, Big Data, and fault-tolerant systems.