

Unit III Distributed Computing Algorithms

3.1 Distributed Computing Algorithms: Communication and coordination in distributed systems Distributed consensus algorithms (Other consensus algorithms • Viewstamped Replication • RAFT • ZAB • Mencius • Many variants of Paxos (Fast Paxos, Egalitarian Paxos etc) Fault tolerance and recovery in distributed systems,



Distributed Computing Algorithms (Simplified)

In Distributed Systems, multiple computers (nodes) work together to perform tasks. But they must coordinate, communicate, and handle failures.



1. Communication and Coordination in Distributed Systems

- Nodes communicate via messages (not shared memory).
- They need to coordinate to:
 - Keep data consistent
 - Perform tasks in the correct order
 - Handle failure without crashing the system



Example: In Google Docs, multiple users editing the same document — the system keeps everyone in sync.



2. Distributed Consensus Algorithms

These algorithms help all nodes agree on a single value or decision, even if some nodes fail.

♦ Popular Consensus Algorithms:

✓ Viewstamped Replication

- A way to replicate data across multiple servers.
- Uses a primary-backup model.
- If the primary fails, a new one is elected.

✓ RAFT

- Easy-to-understand alternative to Paxos.
- Follows steps: Leader Election, Log Replication, and Safety
- Very popular in modern systems like etcd, Kubernetes.

✓ ZAB (ZooKeeper Atomic Broadcast)

- Used in Apache ZooKeeper.
- Ensures leader election and data consistency in a cluster.

✓ Mencius

- An optimization of Paxos for systems with low latency and high throughput.
- Rotates the leader role among nodes.

✓ Paxos (and Variants)


- Complex but very powerful.
- Ensures that distributed nodes agree on values.
- Variants:
 - Fast Paxos – Faster agreement
 - Egalitarian Paxos – All nodes are equal (no fixed leader)

3. Fault Tolerance and Recovery in Distributed Systems

Fault tolerance means the system keeps working even if some nodes fail.

 Techniques:

- Replication: Copy data to multiple nodes.
- Timeouts and Heartbeats: Detect failures quickly.
- Checkpoints and Logs: Recover from failures using saved data.
- Retry and Redundancy: Retry failed operations or use backup nodes.

 *Example:* If one server in a bank system fails, another server takes over without losing any transaction.

3.2 Load balancing and resource allocation strategies: Weighted Round Robin, Least Connection, Randomized Load Balancing, Dynamic Load Balancing, Centralized Load Balancing, Distributed Load Balancing, Predictive Load Balancing


Load Balancing & Resource Allocation Strategies

Load Balancing is the process of **distributing work evenly** across multiple servers or resources to avoid overload, improve performance, and ensure reliability.

Think of it like distributing customers among multiple counters in a bank — we want **all counters working efficiently**.


1. Weighted Round Robin

- Each server is assigned a **weight** based on its capacity.
- Requests are sent in a **round-robin** manner, but **more powerful servers get more requests**.

 *Example:* If Server A has weight 3 and Server B has weight 1, A will handle 3 requests for every 1 request B gets.


2. Least Connection

- New request goes to the server with the **fewest active connections**.
- Great for systems with long or uneven request times.

 *Example:* If Server A has 2 users and Server B has 5, the next user goes to A.

3. Randomized Load Balancing

- Requests are distributed to **random servers**.
- Simple, but can be unfair if one server randomly gets too many requests.

 *Example:* Like rolling a dice to choose a server.

4. Dynamic Load Balancing

- The system **constantly checks the load** on each server and **adapts** accordingly.
- More responsive to real-time changes than static methods.

🧠 *Example:* If Server A is suddenly overloaded, the system shifts new requests to Server B and C.

🧠 5. Predictive Load Balancing

- Uses **AI or machine learning** to **predict future loads** and make smart decisions.
- Allocates resources **before** the load spikes.

🧠 *Example:* If historical data shows traffic increases every Friday at 5 PM, it adds more servers before that time.

🏢 6. Centralized Load Balancing

- A **central controller** decides how to distribute tasks.
- Easier to manage, but a **single point of failure**.

🧠 *Example:* One load balancer in front of multiple servers.

🌐 7. Distributed Load Balancing

- Each server or a group of servers **makes its own load decisions**.
- More robust and scalable, but harder to coordinate.

🧠 *Example:* Peer-to-peer systems where each node helps balance the load.

✅ Summary Table

Strategy	Description
Weighted Round Robin	Servers get tasks based on capacity (weight)
Least Connection	Chooses the server with fewest active connections
Randomized Load Balancing	Picks a server at random
Dynamic Load Balancing	Adjusts based on current server load
Predictive Load Balancing	Uses past data to predict future load
Centralized Load Balancing	One controller manages the distribution
Distributed Load Balancing	Decisions made at server level (no single controller)

3.3 Applying AI techniques to optimize distributed computing algorithms: Machine Learning for Resource Allocation, Reinforcement Learning for Dynamic Load Balancing, Genetic Algorithms for Task Scheduling, Swarm Intelligence for Distributed Optimization

Applying AI in Distributed Computing (Made Easy)

Distributed computing systems involve **multiple machines working together**. To make them faster and smarter, we can use **AI techniques** like machine learning, genetic algorithms, and more.

Let's break it down 🙋



1. Machine Learning for Resource Allocation

- **ML models** learn patterns in resource usage (CPU, memory, network).
- Then they predict **how much resource** is needed for each task.
- This helps the system **allocate resources smartly**, avoiding overload.



Example: A cloud system uses past data to decide how much memory a new app will need.



2. Reinforcement Learning for Dynamic Load Balancing

- RL is like **trial and error learning**.
- The system tries different load balancing strategies and **learns the best one over time**.
- It adapts to changes (like sudden traffic spikes) in real-time.



Example: A video streaming service uses RL to keep users connected to the smoothest server.



3. Genetic Algorithms for Task Scheduling

- Tasks are like genes, and schedules are like chromosomes.
- GA evolves **better task schedules** using **selection, crossover, and mutation**.

- Helps find **near-optimal schedules** quickly, even in complex environments.

🧠 *Example:* A supercomputer schedules 1,000 tasks across 100 CPUs using genetic algorithms to minimize processing time.

🐜 4. Swarm Intelligence for Distributed Optimization

- Inspired by **ants, bees, birds** (collective behavior).
- Swarm-based algorithms like **Ant Colony Optimization (ACO)** and **Particle Swarm Optimization (PSO)** help:
 - Find **shortest paths**
 - **Balance loads**
 - Solve distributed problems efficiently

🧠 *Example:* A logistics system uses ACO to find the best delivery paths in real time across multiple servers.

✅ Summary Table

AI Technique	Used For	Benefit
Machine Learning	Resource Allocation	Predicts and allocates resources intelligently
Reinforcement Learning	Dynamic Load Balancing	Learns and adapts to traffic changes
Genetic Algorithms	Task Scheduling	Finds near-optimal schedules efficiently

AI Technique	Used For	Benefit
Swarm Intelligence (ACO/PSO)	Distributed Optimization	Fast, decentralized problem solving