JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY (JKUAT)

COLLEGE OF ENGINEERING AND TECHNOLOGY (COETEC)

SCHOOL OF ELECTRICAL, ELECTRONIC AND INFORMATION ENGINERRING (SEEIE)

DEPARTMENT OF TELECOMMUNICATION AND INFORMATION ENGINEERING (TIE)

DISTRIBUTED COMPUTING AND ALGORITHMS

ASSIGNMENT 1

GROUP 4

NAME	REGISTRATION NUMBER
ONYANGO WINSTONE	ENE221-0129/2021
ELIJAH SUNKULI	ENE221-0161/2021
JACKSON OCHIENG	ENE221-0136/2021
RAYMOND KIPKORIR	ENE221-0124/2021
ALDAD KIPLAGAT	ENE221-0123/2021
JEMMIMAH MWITHALII	ENE221-0242/2021

ALGORITHMS FOR DISTRIBUTED COMPUTING SYSTEMS

LOAD BALANCING ALGORITHMS

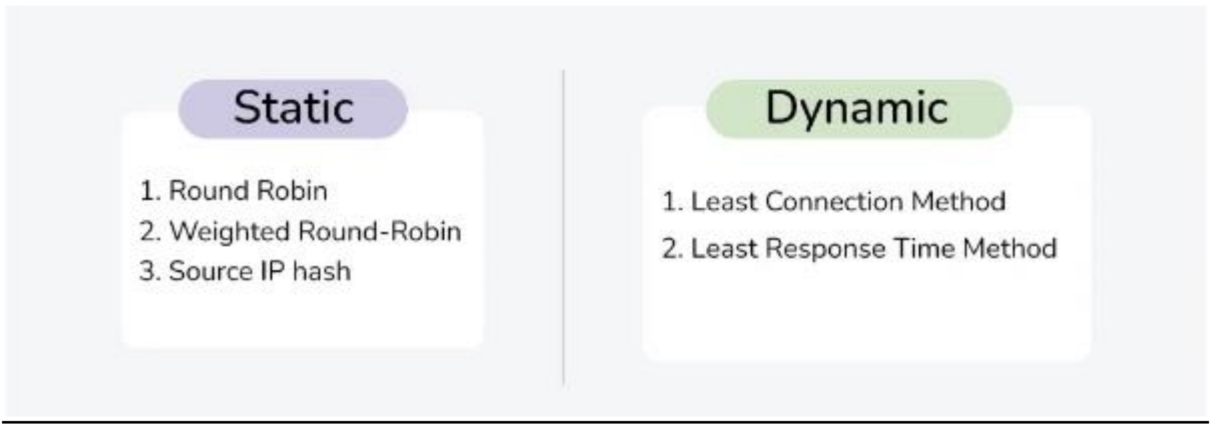
Load balancing is the process of distributing incoming workload or network traffic across multiple servers, nodes, or network paths to ensure no single component becomes overwhelmed.

To control traffic across servers in a network, load-balancing algorithms are important.

- The goal is to make sure that no single server is overloaded when too many users visit an application.
- It monitors the health of servers and redirects traffic away from unresponsive or slow ones.

Load balancing algorithms can be broadly categorized into two types:

1. Dynamic load balancing
2. Static load balancing.



STATIC LOAD BALANCING

Static load balancing is a load distribution approach where the assignment of tasks to servers or nodes is predetermined and does not change based on real-time system conditions.

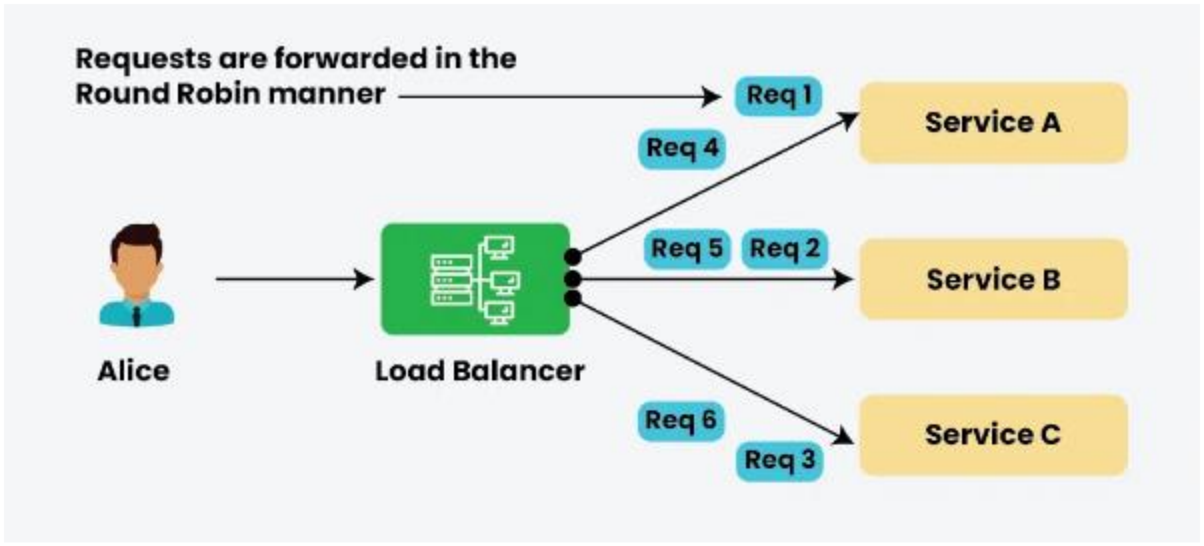
In this method, decisions are made using fixed rules, before the system begins processing workload. The rules include:

- Round Robin.
- Weighted Round Robin.
- Hashing.

Because it does not monitor current traffic or resource usage, static load balancing works best in environments where the workload is predictable and the capabilities of each node are wellknown. It is simple, fast, and low-overhead, making it useful in certain telecom systems, embedded networks, and distributed environments where stability and low complexity are priorities.

- <u>**Round Robin Load Balancing Algorithm**</u>

The Round Robin algorithm is a simple static load balancing approach in which requests are distributed across the servers in a sequential or rotational manner. It is easy to implement but it doesn't consider the load already on a server so there is a risk that one of the servers receives a lot of requests and becomes overloaded.



Python Code Implementation

We implemented a basic Round Robin load balancing algorithm by distributing incoming requests evenly among a list of servers. The first request goes to the first server, the second one goes to the second server, the third request goes to the third server and it continues further for all the requests.

When to use Round Robin Load Balancing Algorithm

1. Ideal for applications where all servers have similar capacity and performance.
2. Works well for evenly distributed workloads, such as basic web requests.
3. Best suited for simple environments without complex resource needs.
4. Useful in setups where request order matters less than balanced distribution across servers.

<u>**Benefits and Drawbacks of Round Robin Load Balancing Algorithm**</u>

Benefits:

1. Simplicity: Easy to implement and understand.
2. Fairness: Ensures that each server gets an equal share of the load.

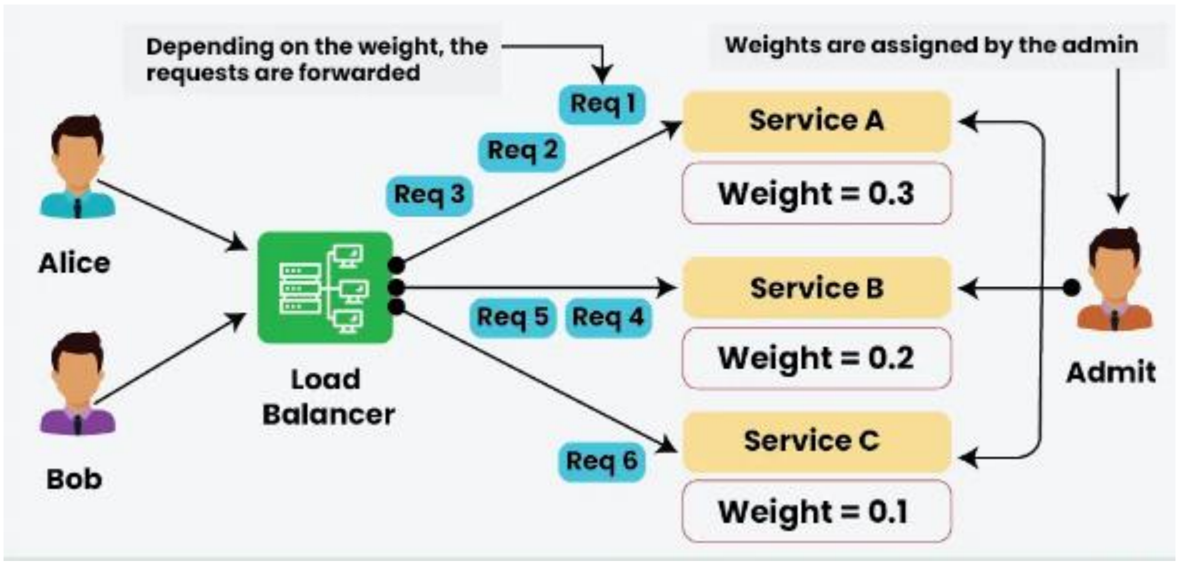
Drawbacks:

1. Unequal Capacities: Doesn't consider the varying capacities of servers; treats all servers equally.
2. Predictability: May not be optimal for scenarios with heterogeneous server capacities.

- <u>**Weighted Round Robin Balancing Algorithm**</u>

The Weighted Round Robin algorithm is also a static load balancing approach which is much similar to the round-robin technique. The only difference is, that each of the resources in a list is provided a weighted score. Depending on the weighted score the request is distributed to these servers.

1. Servers with higher weights are given a larger proportion of the requests.
2. The distribution is cyclic, similar to the round-robin technique, but with each server receiving a number of requests proportional to its weight.
3. If a server reaches its processing capacity, it may start rejecting or queuing additional requests, depending on the server's specific behavior.



<u>**When to use Weighted Round Robin Load Balancing Algorithm**</u>

1. When servers have different capacities or performance levels.
2. Ideal for environments where servers vary in resources (CPU, memory, etc.).

3. Useful when you want to maximize resource utilization across all servers.
4. Helps in preventing smaller servers from overloading while efficiently using larger servers.

Benefits and Drawbacks of Weighted Round Robin Load Balancing Algorithm

Benefits:

1. Capacity Consideration: Accounts for different server capacities by assigning weights.
2. Flexibility: Can be adjusted to handle varying workloads effectively.

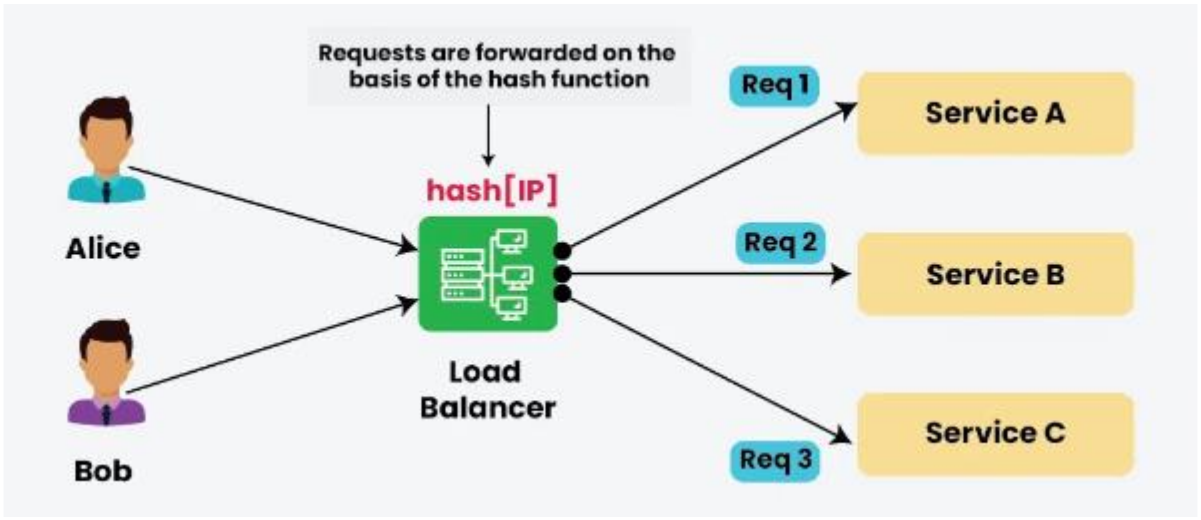
Drawbacks:

1. Complexity: More complex than simple Round Robin.
2. Maintenance: Requires adjusting weights as server capacities change.

Source IP Hash Load Balancing Algorithm

The Source IP Hash Load Balancing Algorithm is a method used in network load balancing to distribute incoming requests among a set of servers based on the hash value of the source IP address. This algorithm aims to ensure that requests originating from the same source IP address are consistently directed to the same server.

If the load balancer is configured for session persistence, it ensures that subsequent requests from the same source IP address are consistently directed to the same server. This is beneficial for applications that require maintaining session information or state.



Python code Implementation

We implemented a load balancing algorithm that distributed incoming requests across a set of servers based on the hash of the source IP address. The goal was to ensure that requests coming from the same source IP address are consistently routed to the same server.

When to use Source IP Hash Load Balancing Algorithm

1. Ideal for applications needing session consistency, like online banking, where the same user must connect to the same server throughout a session.
2. Useful when users from specific regions should connect to dedicated servers for better performance or compliance.
3. Effective when a few IPs generate most of the traffic, ensuring balanced load distribution without random switching.

Benefits and Drawbacks of Source IP Hash Load Balancing Algorithm:
Benefits:

1. Consistency: Ensures requests from the same source IP always go to the same server, maintaining session state.
2. Predictability: Useful when connection persistence is critical.

Drawbacks:

1. Limited Distribution: May lead to uneven load distribution if certain source IPs are more active.
2. Scaling Challenges: Adding or removing servers may disrupt session persistence.

DYNAMIC LOAD BALANCING ALGORITHMS

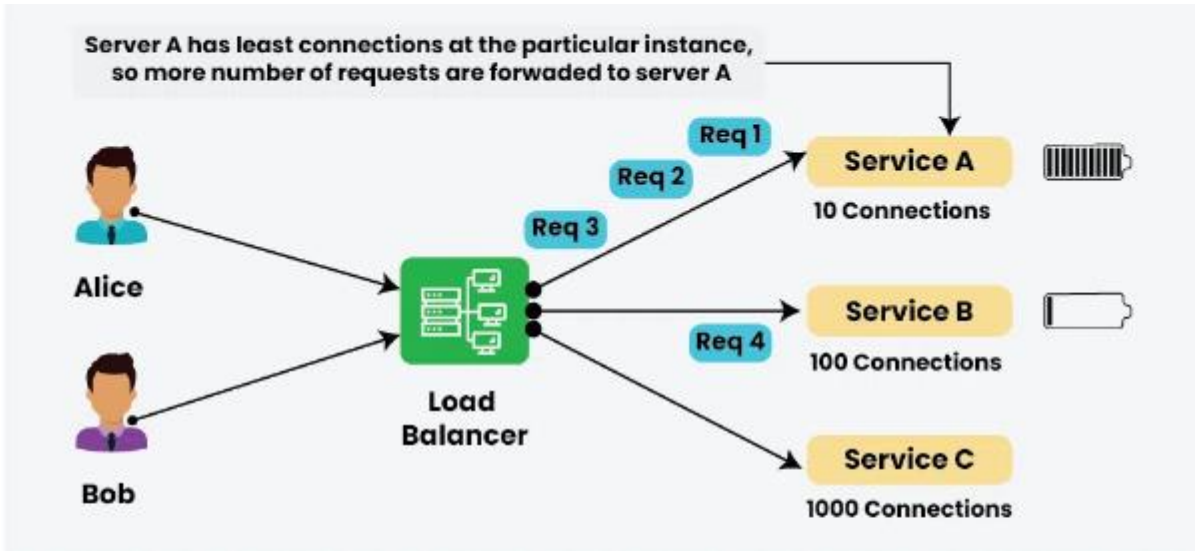
Dynamic load balancing involves making real-time decisions about how to distribute incoming network traffic or computational workload across multiple servers or resources. This approach adapts to the changing conditions of the system, such as variations in server load, network traffic, or resource availability.

- **Least Connection Method Load Balancing Algorithm**

The Least Connections algorithm is a dynamic load balancing approach that assigns new requests to the server with the fewest active connections. The idea is to distribute incoming workloads in a way that minimizes the current load on each server, aiming for a balanced distribution of connections across all available resources.

To do this load balancer needs to do some additional computing to identify the server with the least number of connections.

This may be a little bit costlier compared to the round-robin method but the evaluation is based on the current load on the server.



Python Code Implementation

We implemented a Load Balancing Algorithm that distributes incoming requests across a set of servers and should aim to minimize the number of active connections on each server by directing new requests to the server with the fewest active connections. This ensures a balanced distribution of the workload and prevents overload on individual servers.

When to use Least Connection Load Balancing Algorithm

1. Ideal for applications where some requests take longer to process than others (e.g., video streaming or large file uploads).
2. Useful when some connections stay active longer, as it ensures new requests go to servers with fewer active connections.
3. Great for systems with fluctuating traffic, as it balances based on real-time server load rather than just counting requests.

Benefits and Drawbacks of Least Connection Load Balancing Algorithm:

Benefits:

Balanced Load: Distributes traffic to servers with the fewest active connections, preventing overloading.

Dynamic: Adapts to changing server workloads.

Drawbacks:

Ignored Capacities: Ignores server capacities; a server with fewer connections may still have less capacity.

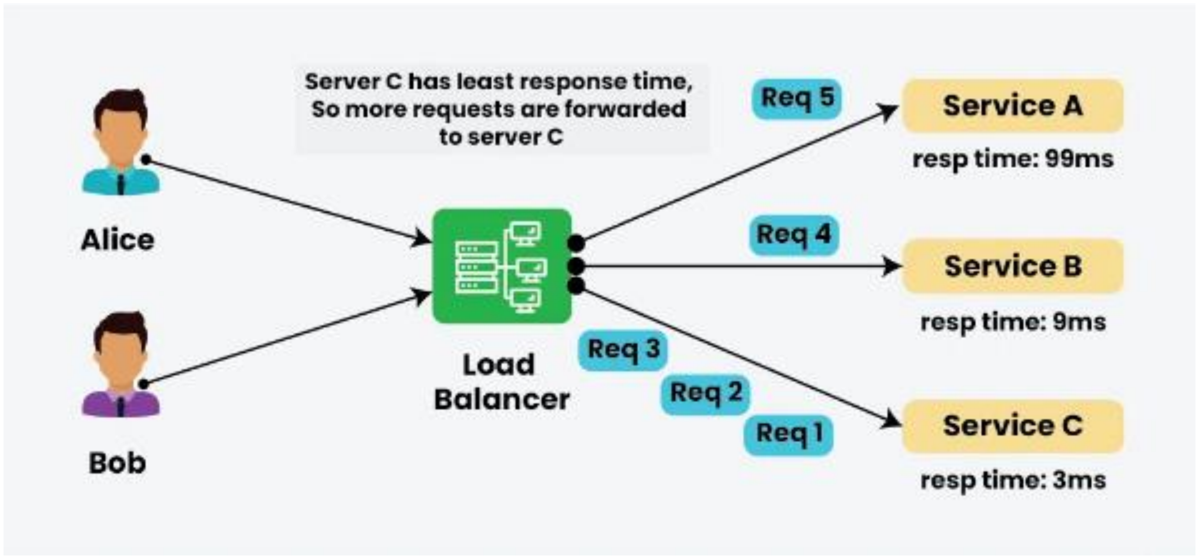
Sticky Sessions: May not be suitable for scenarios requiring session persistence.

- **Least Response Time Method Load Balancing Algorithm**

The Least Response method is a dynamic load balancing approach that aims to minimize response times by directing new requests to the server with the quickest response time.

It considers the historical performance of servers to make decisions about where to route incoming requests, optimizing for faster processing.

The dynamic aspect comes from the continuous monitoring of server response times and the adaptive nature of the algorithm to route incoming requests to the server with the historically lowest response time.



Python Code Implementation

We implemented a Load Balancing Algorithm that distributes incoming requests across a set of servers and should aim to minimize the response time by directing new requests to the server with the least accumulated response time. This ensures a balanced distribution of the workload and helps optimize the overall system performance.

When to use Least Response Time Load Balancing Algorithm

1. Ideal for applications with heavy, fluctuating user traffic where response time matters.
2. Great for apps like e-commerce sites or streaming services, where a quick response improves user experience.
3. Works well when servers have different load levels, as it directs traffic to the server that’s both available and responds the fastest.

Benefits and Drawbacks of Least Response Time Load Balancing Algorithm:

Benefits:

Optimized Performance: Directs traffic to servers with the quickest response times, optimizing overall system performance.

Adaptable: Adjusts to changes in server responsiveness over time.

Drawbacks:

Historical Bias: Heavily influenced by past response times, may not always reflect current server capabilities.

Complex Implementation: Requires tracking and managing historical response times.

COORDINATION AND SYNCHRONISATION ALGORITHMS

Coordination and synchronization algorithms ensure that distributed systems behave in an orderly manner even though nodes operate independently. Because there is no global clock and no shared memory, these algorithms help nodes agree on resource access, event ordering, timing, and shared state.

Examples of theories in the area:

Mutual Exclusion Algorithms - Ensure only one node accesses a shared resource at a time (e.g., a database or a shared file).

Lamport & Vector Clocks - Lamport clocks order events logically (“this happened before that”).Vector clocks track causality between multiple nodes and detect conflicting updates.

Clock Synchronization - Algorithms like Cristian’s and NTP try to maintain synchronized clocks across machines, despite network delays.

Practical Applications:

1. Ensuring only one database server updates subscriber data at a time.
2. Synchronizing logs across telecom sites for troubleshooting.
3. Coordinating distributed tasks in microservices.

Relevance To Modern Telecom, Distributed System and Cloud:

1. Used in handover coordination between base stations.
2. Ensures consistency in distributed databases.
3. Synchronizes events in 5G core functions deployed across multiple servers.
4. Used in cloud microservices to manage shared state across pods.

CONSENSUS ALGORITHMS

Consensus algorithms allow distributed systems to agree on a single value, even if some nodes fail or messages are lost.

This is a hard problem because:

- Nodes may crash,
- Network may delay or duplicate messages,
- No single machine has a complete view.

Practical Applications:

1. Distributed databases (CockroachDB).
2. Logging systems
3. Distributed configuration stores.

Relevance To Modern Telecom, Distributed System and Cloud:

- Ensures routers share a consistent routing table.
- Ensures cloud-based telecom services stay consistent during failures.

SCHEDULING ALGORITHMS

Scheduling algorithms decide which task runs where and when. They aim to optimize:

1. Response time,
2. Resource utilization,
3. Fairness,

4. Deadlines

Examples of theories in the area:

Round Robin — equal time slices

Priority scheduling — important tasks go first

Earliest Deadline First (EDF) — best for real-time systems

Practical Applications:

- CPU scheduling in distributed nodes
- Task scheduling in cloud platforms
- Real-time packet scheduling in routers

Relevance To Modern Telecom, Distributed System and Cloud:

- Ensures voice-over-IP (VoIP) packets are processed before bulk data.
- Helps 5G base stations allocate radio resources efficiently.
- Determines which cloud microservice gets CPU/GPU time.
- Used in load schedulers like Kubernetes.

FAULT TOLERANCE TECHNIQUES

Fault tolerance ensures a system continues working even when some components fail.

Examples of theories in the area:

Replication - Multiple copies of data or services stay updated. Can be active-active or active-passive.

Checkpointing & Rollback - System saves snapshots and returns to them if something breaks.

Leader Election Algorithms - Automatically choose a coordinator after failure.

Practical Applications:

1. Keeping telecom databases online 24/7.
2. Ensuring a cluster continues even if one server dies.
3. Maintaining financial transaction systems.

Relevance To Modern Telecom, Distributed System and Cloud:

- Used in SCADA systems to avoid blackouts.

- Guarantees continuous service in 5G networks.
- Ensures telco services remain up even during node failures.

DISTRIBUTED SEARCH AND RETREIVAL ALGORITHMS

These algorithms allow searching for data spread across many nodes.

Examples of theories in the area:

Flooding - Simple but wastes bandwidth. Sends the same query everywhere.

Distributed Hash Tables (DHTs) - Much more efficient. Uses consistent hashing to find data quickly.

Practical Applications:

1. Peer-to-peer networks.
2. Distributed file systems.
3. Decentralized search engines.

Relevance To Modern Telecom, Distributed System and Cloud:

- Used in peer-to-peer VoIP (like old Skype).
- Helps distributed DNS systems find data.
- Used in cloud storage systems like Cassandra, DynamoDB.
- Used in distributed caching systems e.g., CDN edge caches.

DATA PARTITIONING AND SHARDING

Sharding breaks large datasets into smaller pieces.

Examples of theories in the area:

Horizontal Partitioning - Split by rows (e.g., customers 1–1M on server A, next million on B).

Vertical Partitioning - Split by columns (e.g., profile data vs financial data).

Consistent Hashing - Avoids massive data movement when nodes join or leave.

Practical Applications:

1. Scaling large databases easily.
2. Handling millions of customer records.
3. Increasing throughput in distributed systems.

Relevance To Modern Telecom, Distributed System and Cloud:

- Used in call detail record (CDR) processing.
- Cloud platforms use sharding to scale SQL and NoSQL databases.
- Used in 5G network slicing data storage.

LARGE SCALE DATA PROCESSING ALGORITHMS

Large-scale data processing algorithms handle huge datasets by distributing work.

Examples of theories in the area:

MapReduce;

Map phase - split task across nodes

Shuffle phase - group data

Reduce phase - aggregate results

Stream Processing;

Processes real-time data flows

Ideal for IoT, telecom traffic, online systems\

Practical Applications:

1. Analyzing logs
2. Big data analysis
3. Machine learning pipelines
4. Real-time analytics (fraud detection, traffic patterns)

Relevance To Modern Telecom, Distributed System and Cloud:

- Used to analyze call detail records (CDRs). Helps detect network congestion in real time. Processes billions of events from IoT devices. Powers cloud big data systems like Kafka.