

Big Data Analytics: Lecture 1.6 – Basics of Distributed Computing

Dr. Tariq Mahmood

Fall 2025

1 Core Concepts of Distributed Computing

A distributed system is a collection of autonomous computing elements that appears to its users as a single, coherent system. These systems are fundamental to big data analytics, providing the foundation for scalable and resilient data processing.

- **Multiple Nodes:** The system is composed of multiple independent computers (nodes) that are interconnected via a network. They communicate and coordinate by passing messages to one another.
- **Concurrency:** All nodes in the system can operate in parallel and execute tasks simultaneously. This is the key to the high performance of distributed systems.
- **No Shared Clock:** There is no single global clock that synchronizes all nodes. Each node has its own system clock, and the clocks can drift apart. This lack of a common time reference creates challenges for ordering events.
- **Independent Failures:** Individual nodes can fail at any time without causing the entire system to crash. The system must be designed to detect and handle such partial failures gracefully.
- **Transparency:** A key goal of a distributed system is to hide its complexity from the end-user. It should appear as a single, centralized system. This includes location transparency (users don't know where resources are located) and failure transparency (users may not notice that a node has failed and been replaced).

1.1 Core Features

- **Scalability:** The system should be able to handle a growing amount of work by adding more nodes (scaling out). A well-designed distributed system allows for the addition and removal of nodes without service interruption.
- **Fault Tolerance:** The system must remain operational even in the face of node failures. This is typically achieved through redundancy, where data and computations are replicated across multiple nodes.
- **Resource Sharing:** Every node has its own hardware resources (CPU, memory, disk), but software resources and data can be shared and accessed across the network, enabling collaboration between nodes.
- **Performance:** A primary objective is to execute large-scale computations and queries efficiently. By parallelizing tasks across many nodes, results for complex queries can be returned in a reasonable time (seconds), avoiding long hangs.

2 Communication Methods

Communication is the cornerstone of a distributed system, enabling nodes to coordinate their actions. Different patterns of communication are used depending on the task.

2.1 Broadcast

- **Definition:** One-to-All communication. A single source node sends the same message to every other node in the network.
- **Variants:**
 - **Flooding:** A simple but inefficient method. When a node receives the message, it forwards it to all its neighbors except the one it received it from. This creates many redundant messages.
 - **Tree-based broadcast:** A more efficient method that uses a spanning tree of the network graph. The message is sent along the edges of the tree, ensuring each node receives the message exactly once, thus avoiding redundancy.
- **Used in:** System-wide updates, replicating data to all nodes, and failure detection mechanisms (e.g., sending heartbeat messages).
- **Advantages:** Simple concept, ensures all nodes get the information.
- **Disadvantages:** Can lead to network congestion, especially with flooding. Not scalable for very large networks.

2.2 Multicast

- **Definition:** One-to-Many communication. A message is sent from a single source to a specific *subset* of nodes in the network (a multicast group).
- **Guarantees:**
 - **Reliable multicast:** Ensures that all intended recipients in the group receive the message.
 - **Atomic multicast:** A stronger guarantee ensuring that the message is delivered to either all members of the group or none at all. This is an all-or-nothing delivery property.
- **Used in:** Group communication, consensus algorithms (where a proposal is sent to a quorum of nodes), and collaborative applications.
- **Advantages:** More efficient than broadcast when the target is a subset of nodes, reducing unnecessary network traffic.
- **Disadvantages:** More complex to manage group membership and ensure reliability compared to broadcast.

2.3 Gossip / Epidemic Protocol

- **Definition:** A decentralized communication model where information spreads through the network like a biological epidemic. Each node periodically and randomly shares information with a few of its peers.
- **Mechanism:** Information gradually propagates through the entire network as nodes repeatedly "gossip" with their neighbors.
- **Used in:** Large-scale distributed systems, databases (like Cassandra) for failure detection and state synchronization, and blockchain networks for transaction propagation.
- **Advantages:** Highly scalable and fault-tolerant. There is no single point of failure.
- **Disadvantages:** Information propagation is not instantaneous (eventual consistency), and there are no guarantees about the delivery time.

2.4 Unicast

- **Definition:** One-to-One communication. This is the most basic form of point-to-point messaging between a single sender and a single receiver.
- **Role:** It is the fundamental building block for more complex communication patterns and higher-level algorithms.
- **Reliability:** Protocols like TCP are built on top of unicast to ensure reliable message delivery (guaranteeing ordering and no loss) over inherently unreliable networks like the internet.
- **Advantages:** Simple, direct, and efficient for two-party communication.
- **Disadvantages:** Inefficient for sending the same data to multiple nodes.

2.5 Anycast

- **Definition:** A message is sent to a group of nodes that are logically identified by the same address, but the network delivers the message to only *one* of them, typically the "nearest" or best node based on routing topology.
- **Used in:** Load balancing (directing a client to one of several identical servers) and selecting the nearest server to reduce latency (e.g., DNS services, Content Delivery Networks).
- **Advantages:** Improves availability and performance by routing traffic intelligently.
- **Disadvantages:** Can be more complex to configure and debug than other methods.

2.6 Reduce / Convergecast

- **Definition:** Many-to-One communication. Multiple nodes send messages that converge toward a single root node. Often, the data is aggregated or combined as it moves up a spanning tree structure.
- **Used in:** The 'Reduce' phase of MapReduce, data aggregation in sensor networks, and collecting results from worker nodes to a leader node.
- **Worked Example:** In a MapReduce job, multiple Mapper nodes process data and emit intermediate key-value pairs. In the Reduce phase, all values for a given key are sent from many Mappers to a single Reducer node responsible for that key. This is a convergecast pattern.

2.7 All-to-All Communication

- **Definition:** Every node in a group sends data to every other node in that group. This is also known as a shuffle operation in big data contexts.
- **Characteristics:** It is a very communication-intensive and expensive operation, as it can generate a large amount of network traffic.
- **Used in:** Crucial for many parallel algorithms, such as distributed matrix multiplication and the shuffle phase in MapReduce and Spark, where data is redistributed across nodes based on keys.

3 Message Ordering

In a distributed system, the network can delay, drop, or reorder messages. Without explicit control, different nodes can observe events in different sequences, leading to inconsistencies. Ordering methods provide guarantees about the sequence of message delivery.

3.1 FIFO (First-In, First-Out) Ordering

- **Guarantee:** Messages sent by a single sender are delivered to a receiver in the same order they were sent.
- **Limitation:** There is no guarantee on the order of messages from *different* senders.
- **Used in:** TCP sockets provide FIFO ordering. Kafka partitions also guarantee that messages within a single partition are delivered in order.
- **Worked Example:**
 - Sender P1 sends message m1, then m2.
 - Sender P2 sends message n1, then n2.
 - A receiver must deliver m1 before m2, and n1 before n2.
 - However, the interleaving is not guaranteed. A valid observed sequence could be **m1, n1, m2, n2** or **n1, m1, n2, m2**. An invalid sequence would be **m2, m1, n1, n2**.

3.2 Causal Ordering

- **Guarantee:** Preserves cause-and-effect relationships. If the sending of message m2 is influenced by (happens-after) the receipt of message m1, then all processes must deliver m1 before they deliver m2.
- **Implementation:** Achieved using logical clocks like Lamport timestamps or vector clocks.
- **Used in:** Flink streaming operators, collaborative real-time editing systems, and NoSQL database replication.
- **Worked Example:**
 - Process P1 sends message m1.
 - Process P2 receives m1 and, as a result, sends message m2.
 - The sending of m2 is causally dependent on m1.
 - Causal ordering ensures that any other process (P3) that receives both messages must deliver m1 before it delivers m2.

3.3 Total Ordering

- **Guarantee:** The strongest ordering guarantee. All processes deliver all messages in the exact same sequence, regardless of who sent them.
- **Characteristics:** While all processes agree on a single order, that order might not necessarily match the real-time sending order of concurrent messages.
- **Used in:** State machine replication, consensus protocols, and systems like ZooKeeper and Kafka (for log replication) where a consistent global sequence of operations is critical.
- **Worked Example:**
 - P1 sends m1 and P2 sends n1 concurrently.
 - If Process A delivers the messages in the sequence **m1, then n1**, then Total Ordering guarantees that Process B, C, and all others must also deliver them in the sequence **m1, then n1**.
 - They will never disagree (e.g., B seeing **n1, then m1**).

4 Synchronization Methods

Synchronization is crucial in distributed systems to coordinate actions and maintain consistency in the absence of a shared clock and shared memory.

4.1 Clock Synchronization

Nodes must agree on the time to order events or schedule tasks.

4.1.1 Cristian's Algorithm

- **Concept:** A client synchronizes with a time server that is assumed to have an accurate time.
- **Process:**
 1. A client sends a time request to the server.
 2. The server responds with its current time, T.
 3. The client measures the round-trip time (RTT) for the request-response cycle. It assumes the one-way latency is RTT / 2.
 4. The client sets its clock to $T + (\text{RTT} / 2)$ to compensate for the network delay.
- **Used in:** Early versions of NTP.

4.1.2 Berkeley Algorithm

- **Concept:** A democratic approach used in systems where no machine has a guaranteed accurate time source.
- **Process:**
 1. A coordinator node (master) polls all other nodes (slaves) for their time.
 2. The master computes an average time, accounting for propagation delays.
 3. The master tells each node how much to adjust its clock (forward or backward).
- **Used in:** Clusters before NTP became widespread.

4.1.3 Network Time Protocol (NTP)

- **Concept:** The internet standard for time synchronization, based on a hierarchical system of time sources and round-trip time estimation. It's accurate to milliseconds.
- **Strata (Hierarchy of Time Sources):**
 - **Stratum 0:** High-precision sources like atomic clocks or GPS. These are the reference clocks.
 - **Stratum 1:** Servers directly connected to Stratum 0 devices.
 - **Stratum 2:** Servers that sync with Stratum 1 servers.
 - **Stratum N:** Servers that sync with Stratum N-1 servers.
- **Timestamp Exchange:** Each NTP exchange involves 4 timestamps to calculate delay and offset:
 - **T1:** Client sends request (client's departure time).
 - **T2:** Server receives request (server's arrival time).
 - **T3:** Server sends response (server's departure time).
 - **T4:** Client receives response (client's arrival time).
- **Client Estimates:**
 - Round-trip delay = $(T4 - T1) - (T3 - T2)$
 - Clock offset = $\frac{(T2 - T1) + (T3 - T4)}{2}$
- **Used in:** Widely across the internet and in data clusters (Hadoop, Spark) to synchronize log timestamps.

4.2 Logical Clocks

When precise physical time is not available or necessary, logical clocks are used to order events based on causality.

- **Lamport Timestamps:** Assigns a counter (integer) to each event. It ensures that if event A happens-before event B, then $L(A) \downarrow L(B)$. However, it cannot distinguish concurrent events.
- **Vector Clocks:** Each node maintains a vector of counters (one for each process). This provides a more precise causal history and can distinguish between causally related events and concurrent events. Used in DynamoDB and Cassandra for conflict resolution.

4.3 Barrier Synchronization

- **Concept:** A checkpoint in a program. All participating processes must reach the barrier before any of them are allowed to proceed.
- **Worked Example:** In Hadoop MapReduce or Spark, a barrier exists between the Map and Reduce stages. All Mapper tasks must complete successfully before the framework begins the Reduce tasks.

4.4 Mutual Exclusion

- **Concept:** Ensures that only one process at a time can access a shared resource (a critical section), preventing data corruption.
- **Algorithms:**
 - **Centralized:** A single coordinator node grants a lock to processes wishing to enter the critical section.
 - **Distributed (Ricart-Agrawala):** A node wanting to enter the critical section must request and receive permission from all other nodes.
 - **Token-based (Suzuki-Kasami):** A single "token" circulates among the nodes. A node can enter the critical section only if it holds the token.
- **Used in:** ZooKeeper for distributed locks and leader election. HBase uses it to ensure only a single master node is active at a time.

4.5 Election Algorithms

- **Concept:** A procedure to select a single node to act as a leader or coordinator, which is a form of synchronizing authority. This is often needed when a current leader fails.
- **Algorithms:**
 - **Bully Algorithm:** The node with the highest process ID "bullies" its way into becoming the leader by sending election messages and asserting its priority.
 - **Ring Algorithm:** Nodes are arranged in a logical ring and pass election messages around the ring until a leader is chosen.
- **Used in:** ZooKeeper for leader election and Kafka for controller election.

5 Summary of Concepts in Big Data Frameworks

Communication Algorithm	Algo-	Big Data Tools / Frameworks	How It's Used
Unicast (one-to-one)		Hadoop HDFS, Apache Kafka, Spark	Data blocks sent from DataNode to Client (HDFS); Kafka producer sends message to a broker; Spark task sends status to driver.

Communication Algorithm	Algo-	Big Data Tools / Frameworks	How It's Used
Broadcast (one-to-all)		Apache Spark, Hadoop MapReduce	Spark uses broadcast variables to efficiently share read-only data with all worker nodes; MapReduce broadcasts job configurations from the Resource-Manager to all nodes.
Multicast (one-to-many)		Apache Flink, Apache Storm, Kafka	Stream data is pushed from one source operator to multiple downstream tasks (DAG edges); Kafka topics can multicast messages to multiple consumers in a consumer group.
Gossip / Epidemic		Apache Cassandra, Amazon DynamoDB, Apache Flink	Used for cluster membership, failure detection, and disseminating state information in a decentralized and scalable manner.
Anycast		Kubernetes, Kafka	Used for load balancing — a client request is routed to one of many available Kafka brokers or service instances, not all of them.
Reduce / Convergecast (many-to-one)		Hadoop MapReduce, Spark, Flink	The Reduce phase aggregates results from multiple mappers; Spark's 'reduceByKey' and Flink's aggregations collect data at a single point.
All-to-All Communication (shuffle)		Hadoop MapReduce, Spark, Flink, Hive	The shuffle phase where mappers send data to all reducers based on keys; Spark shuffles data among all workers for wide transformations.
Message Ordering (FIFO, causal, total)		Kafka, Pulsar, ZooKeeper	Kafka maintains partition order (FIFO); distributed logs and systems like ZooKeeper rely on strong total ordering to ensure consistent operations across the cluster.
Barrier Synchronization		Spark, MPI for big data	Spark's barrier execution mode for synchronizing parallel tasks; DAG stage completion requires all tasks in a stage to finish before the next stage begins.
Mutual Exclusion (distributed locks)		ZooKeeper, HBase, Hadoop YARN	ZooKeeper provides distributed locks for coordination and is used for leader election; HBase uses this for its master election.

Table 1: Communication and Synchronization in Practice

5.1 Citations

- Tanenbaum, A. S., & Van Steen, M. (2017). *Distributed Systems: Principles and Paradigms*. Pearson. (General concepts of distributed computing).
- Kleppmann, M. (2017). *Designing Data-Intensive Applications*. O'Reilly Media. (Practical applications of ordering and consensus in databases).