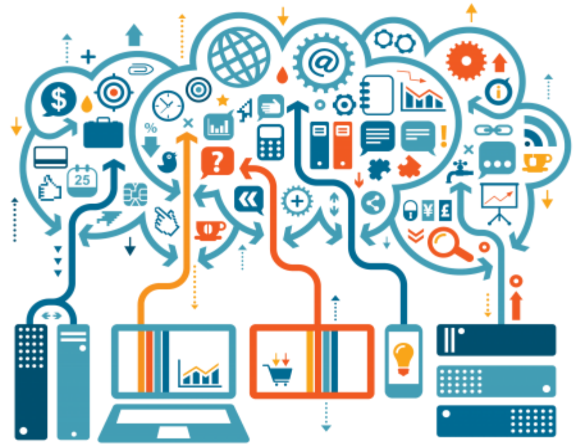

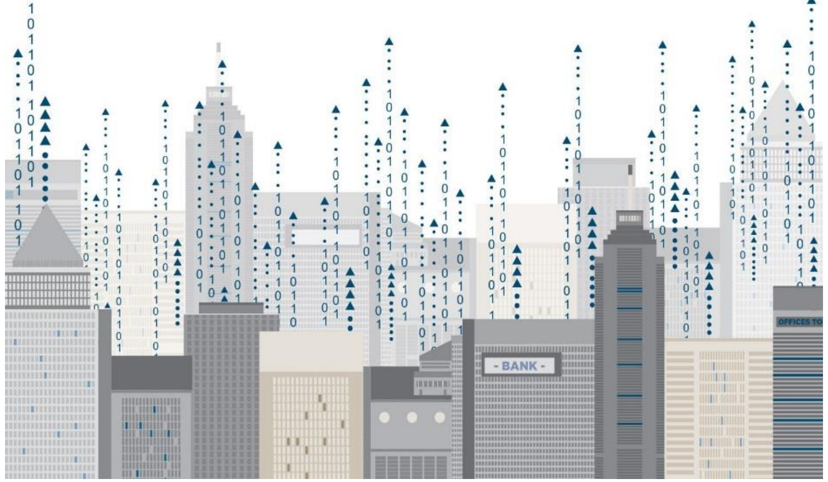

Big Data Analytics

Fall 2025

Lecture 1.6 – Basics of Distributed Computing

Core Concepts

- Multiple Nodes (connected together through a network)
- Concurrency (all nodes work together at the same time)
- No shared clock (every node has its own clock)
- Independent Failures (any node can go down any time)
- Transparency (It seems to the user as if there is only one system serving only him and nothing is distributed)

Core Features

- Scalability: you can add and remove nodes whenever you want – the system should not go down)
- Fault Tolerance: any node can go down any time, but the distributed system keeps on running
- Resource sharing: every node has its own set of hardware resources and there are software resources – all can be shared
- Performance: the goal is that a big query should not hang and results should come within a reasonable time (not in minutes) – many optimized queries finish in seconds

Communication methods

Broadcast

- One-to-All Communication
- A single node (source) sends a message to all other nodes.
- Variants:
  - Flooding (each node forwards to all neighbors, except sender).
  - Tree-based broadcast (uses spanning tree to avoid redundancy).
- Used in: updates, replicated data, failure detection.

Multicast

- One-to-Many Communication
- A message is sent to a *subset* of nodes.
- Ensures **reliable multicast** (all intended recipients get the message) or **atomic multicast** (all-or-nothing delivery).
- Used in: group communication, consensus, collaborative apps.

Gossip/Epidemic

- Each node randomly shares information with a few peers
- Information spreads like an epidemic
- Used in: large-scale systems, databases, blockchain networks

Unicast

- One to One communication
- Basic point-to-point messaging between nodes
- Foundation for higher-level algorithms
- Usually ensures **reliable message delivery** over unreliable networks

Anycast

- A message is delivered to any one node from a group
- Often used in load balancing or nearest server selection

Reduce / Convergecast

- **Many-to-One Communication**
- Multiple nodes send messages toward a root (e.g., aggregation along a spanning tree).
- Used in: MapReduce, sensor networks, leader collection.

All-to-All Communication

- Every node sends data to every other node.
- Expensive, but important in parallel algorithms (e.g., matrix multiplication).

Message Ordering

Rules

- In a distributed system:
  - Nodes exchange **asynchronous messages**.
  - Network can delay, drop, or reorder messages.
  - If ordering isn't controlled, different nodes may **see events in different sequences**, causing inconsistency.
- Ordering methods define **rules** to ensure all processes agree on the **sequence of message delivery**.

List

- **FIFO ordering**: messages delivered in the order sent by each sender.
- **Causal ordering**: respects causality (based on Lamport timestamps or vector clocks).
- **Total ordering**: all nodes deliver messages in the same order.
- Used in: state machine replication, consensus protocols.

FIFO

- Messages from the same sender are delivered in the order sent.
- No guarantee across multiple senders.
- Example:
  - If P1 sends m1 then m2, all receivers must deliver m1 before m2.
  - But messages from P2 may interleave differently.

Sender P1: m1 → m2
Sender P2: n1 → n2

Receiver may see: m1, n1, m2, n2

FIFO

- Messages from the same sender are delivered in the order sent.
- No guarantee across multiple senders.
- Example:
  - If P1 sends m1 then m2, all receivers must deliver m1 before m2.
  - But messages from P2 may interleave differently.
- Kafka partitions, TCP sockets

Sender P1: m1 → m2
Sender P2: n1 → n2

Receiver may see: m1, n1, m2, n2

Causal Ordering

- Preserves cause-and-effect relationships between messages.
- If message m1 may have influenced m2, then all processes must deliver m1 before m2.
- Achieved using Lamport timestamps or vector clocks.
- Example:
  - P1 sends m1.
  - P2 receives m1 and then sends m2.
  - Causal ordering requires that everyone deliver m1 before m2.
- Flink streaming operators, collaborative editing systems, NoSQL replication

P1: m1 ——▶ P2: receives m1 → sends m2
All processes: must see m1 before m2

Total Ordering

- All processes deliver all messages in the same order, regardless of senders.
- Strongest guarantee.
- Example: If m1 and n1 are sent concurrently by different processes:
  - Process A delivers m1, n1
  - Then every process must deliver m1, n1 in the same order.
- ZooKeeper, Kafka log replication, state machine replication.

All processes see: m1 → n1 OR n1 → m1
But never disagree

Synchronization Methods

Why?

- In a distributed system:
  - There's **no global clock** and no shared memory.
  - Nodes must **coordinate actions** so the system behaves consistently.
  - Synchronization ensures that events across processes **line up correctly** for correctness.

Clock

- Nodes must agree on time to order events or schedule tasks.
- Cristian:
  - One machine with accurate time (time server).
  - A client asks the server for time.
  - Client adjusts its clock based on round-trip delay estimate.
- Used: Early NTP

Client → Server (time request)
Server → Client (time T)
Client sets: T + (RTT / 2)

Clock

- Nodes must agree on time to order events or schedule tasks.
- Berkeley:
  - No machine has perfect time
  - One node polls all others, computes average, and tells each node how much to adjust
- Used: Clusters before NTP

Clock - NTP

- Based on **hierarchical strata (levels)** and **round-trip time estimation**.
- **1. Strata (Hierarchy of Time Sources)**
- **Stratum 0:** High-precision time sources (atomic clocks, GPS, radio clocks).
- **Stratum 1:** Servers directly connected to stratum 0 devices.
- **Stratum 2:** Servers synced to stratum 1 servers.
- **Stratum N:** Higher levels synced recursively.

Clock - NTP

- Each NTP exchange involves 4 timestamps:
  - T1: Client sends request (client time).
  - T2: Server receives request (server time).
  - T3: Server sends response (server time).
  - T4: Client receives response (client time).
- Client estimates
  - Round-trip delay = (T4 – T1) – (T3 – T2)
  - Clock offset = ((T2 – T1) + (T3 – T4)) / 2
- This helps compensate for network delays.

Clock

- Nodes must agree on time to order events or schedule tasks.
- Network Time Protocol:
  - Hierarchical time synchronization
  - Internet standard; accurate to milliseconds
- Used: Hadoop clusters, Spark clusters (to sync log timestamps)
- Based on **hierarchical strata (levels)** and **round-trip time estimation**.
- **1. Strata (Hierarchy of Time Sources)**
- **Stratum 0:** High-precision time sources (atomic clocks, GPS, radio clocks).
- **Stratum 1:** Servers directly connected to stratum 0 devices.
- **Stratum 2:** Servers synced to stratum 1 servers.
- **Stratum N:** Higher levels synced recursively.

Logical Clock

- When physical time is not available or reliable, use **logical time**.
- **Lamport Timestamps**:
  - Each event gets a logical counter.
  - If A → B (happens-before), then L(A) < L(B).
  - Doesn’t capture concurrency precisely.
- **Vector Clocks**:
  - Each node keeps a vector of counters (one per process).
  - Can distinguish between **causal order** and **concurrent events**.
- Used in: causal multicast, DynamoDB, Cassandra for conflict resolution.

Barrier

- All processes must reach a barrier before any can proceed.
- Think of it like a checkpoint.
- Example: In Spark or Hadoop MapReduce, all mappers must finish before reducers start.

```
P1 ----┐
P2 ----┼--> Barrier --> All proceed
P3 ----┘
```

Mutual Exclusion

- Ensures **only one process at a time** can access a shared resource (like memory).
- **Algorithms:**
  - **Centralized** (one coordinator grants lock).
  - **Distributed (Ricart–Agrawala)**: nodes request permission from all others.
  - **Token-based (Suzuki–Kasami)**: token circulates; whoever has it can enter critical section.
- Used in:
  - ZooKeeper → leader election, distributed locks.
  - HBase → single master active at a time.

Election Algorithms

- Select one leader (synchronization of authority).
- **Bully Algorithm**: highest ID node becomes leader.
- **Ring Algorithm**: nodes pass messages in a ring until leader chosen.
- Used in:
    - ZooKeeper leader election
    - Kafka controller election.

Communication Algorithm	Big Data Tools / Frameworks	How It's Used
Unicast (one-to-one)	Hadoop HDFS, Apache Kafka, Spark	Data blocks sent from DataNode → Client (HDFS); Kafka producer → broker; Spark task → driver.
Broadcast (one-to-all)	Apache Spark, Hadoop MapReduce	Spark uses broadcast variables to share read-only data across workers; MapReduce job configurations broadcast from JobTracker/ResourceManager to all nodes.
Multicast (one-to-many)	Apache Flink, Apache Storm, Kafka	Stream data pushed from one source operator to multiple downstream tasks (multicast DAG edges). Kafka topics can multicast messages to multiple consumers in a group.
Gossip / Epidemic	Apache Cassandra, Amazon DynamoDB, Apache Flink	Gossip protocol used for cluster membership, failure detection, and state dissemination.
Anycast	Kubernetes (with big data on containers), Kafka	Load balancing — client request goes to *one* broker or service instance, not all.

Communication Algorithm	Big Data Tools / Frameworks	How It’s Used
Reduce / Convergecast (many-to-one)	Hadoop MapReduce, Spark, Flink	Reduce phase aggregates results from multiple mappers; Spark’s reduceByKey, Flink’s aggregations.
All-to-All Communication (shuffle)	Hadoop MapReduce, Spark, Flink, Hive	Shuffle phase — mappers send data to all reducers based on keys; Spark shuffle exchanges data among all workers.
Message Ordering (FIFO, causal, total)	Kafka, Pulsar, ZooKeeper	Kafka maintains partition order (FIFO); ZooKeeper ensures total order of operations; distributed logs rely on strong ordering.
Barrier Synchronization	Spark, MPI for big data (HPC style)	Spark’s barrier() mode for synchronization in parallel tasks; DAG stage completion requires all tasks to finish before moving on.
Mutual Exclusion (distributed locks)	ZooKeeper, HBase, Hadoop YARN	ZooKeeper provides distributed locks and leader election; HBase master election uses it.