Distributed Systems

Short notes for B.Tech CSE / IT · MCA · M.Tech

Uncodemy Handwritten Notes · free for students

I. Introduction

- **Distributed system** = a set of independent computers that appear to users as a single coherent system.
- Computers communicate only by **passing messages** over a network. No shared memory, no global clock.
- Examples: the web, online banking, cloud storage, multiplayer games, peer-to-peer file sharing.

➤ Goals

1. **Resource sharing**: share hardware, data and services.
2. **Transparency**: hide the fact that the system is distributed.
3. **Openness**: standard interfaces, easy to extend.
4. **Scalability**: work well as users and machines grow.
5. **Fault tolerance**: keep working when parts fail.

➤ Types of transparency

Transparency	Hides
Access	How a resource is accessed (local vs remote)
Location	Where a resource is
Migration	That a resource moved
Relocation	That it moved while in use
Replication	That there are many copies
Concurrency	That many users share it
Failure	Failure and recovery of a component

➤ Challenges

- Heterogeneity, security, partial failures, concurrency, no global time, network delay.

- **Fallacies of distributed computing**: the network is reliable, latency is zero, bandwidth is infinite (all false).

2. System Models and Architecture

- **Client-server**: clients request, servers respond. Simple but the server can be a bottleneck.
- **Peer-to-peer (P2P)**: every node is both client and server (e.g. BitTorrent).
- **Multi-tier**: presentation tier, logic tier, data tier.
- **Middleware**: software layer that hides heterogeneity (RPC, RMI, message queues, CORBA).

➤ Interaction models

Model	Assumption
Synchronous	Known bounds on message delay, process speed and clock drift
Asynchronous	No bounds on delays or speeds (like the internet)

➤ Failure models

- **Crash (fail-stop)**: process stops. **Omission**: messages lost.
- **Timing**: response too late. **Byzantine (arbitrary)**: process behaves randomly or maliciously.

> **EXAM POINT** Byzantine fault tolerance needs $n \geq 3f + 1$ processes to tolerate f faulty ones.

3. Communication

➤ RPC (Remote Procedure Call)

- Calling a procedure on another machine like a local call.
- **Client stub** packs (marshals) parameters into a message; **server stub** unpacks and calls the real procedure.

```
client -> client stub -> network -> server stub -> server
       <-  unmarshal  <-  reply  <-  marshal    <-
```

- Problems: different data formats, pointers cannot be passed, failures of network or server.

- Call semantics: **maybe**, **at-least-once**, **at-most-once**, **exactly-once** (ideal, hard).

➤ RMI and message passing

- **Java RMI**: remote method calls on remote objects; uses stub and skeleton, RMI registry.
- **Message-oriented middleware**: queues; sender and receiver need not be online together (asynchronous).
- **Persistent vs transient**, **synchronous vs asynchronous** communication.
- **Group communication / multicast**: send to a group of processes.

4. Time and Clock Synchronisation

- Each machine has its own clock; clocks **drift**. There is no global time.

➤ Physical clock sync

- **Cristian's algorithm**: client asks a time server.

FORMULA Cristian: new client time = server time + (round trip time / 2).

EXAMPLE Request sent at 10:00:00.000, reply received at 10:00:00.040 with server time 10:00:05.000. RTT = 40 ms. Set clock to 10:00:05.020.

- **Berkeley algorithm**: master polls all machines, computes the average, tells each one how much to adjust. No machine needs a perfect time source.
- **NTP**: hierarchy of time servers (strata) used on the internet.

➤ Logical clocks

- **Happened-before (→)**: $a \rightarrow b$ if a comes before b in the same process, or a is a send and b is the matching receive. It is transitive.
- **Lamport clock** rules:
  1. Before each event, $C = C + 1$.
  2. Send the timestamp with every message.
  3. On receive: $C = \max(C, \text{received timestamp}) + 1$.
- If $a \rightarrow b$ then $C(a) < C(b)$. The reverse is not always true.

EXAMPLE P1 events a(1), send m (2). P2 has event b(1), then receives m: $C = \max(1, 2) + 1 = 3$.

- **Vector clock**: each process keeps a vector of n counters. VC(a) < VC(b) if and only if a → b. Detects concurrent events.

> **REMEMBER** Lamport gives a consistent order. Vector clocks also tell you when two events are concurrent.

5. Mutual Exclusion and Election

➤ Distributed mutual exclusion

Algorithm	Idea	Messages per entry
Centralised	One coordinator grants access	3 (request, grant, release)
Ricart–Agrawala	Ask all others, enter when all reply	2(n − 1)
Lamport's algorithm	Timestamped requests in queues	3(n − 1)
Token ring	Only token holder can enter	1 to ∞ (token keeps moving)
Maekawa	Ask only a voting set of size about sqrt(n)	About 3·sqrt(n)

> **EXAMPLE** Ricart–Agrawala with n = 5 processes: 2(5 − 1) = 8 messages per critical section entry.

➤ Election algorithms

- Pick a new coordinator when the current one fails.
- **Bully algorithm**: a process sends ELECTION to all higher ID processes. If none answers, it wins and sends COORDINATOR. Highest live ID always wins.
- **Ring algorithm**: ELECTION message goes around the ring collecting IDs; the highest becomes coordinator.

6. Consistency and Replication

- **Replication** = keeping copies on many machines for reliability and speed.
- Problem: keeping copies consistent when updated.

Consistency models (strict to weak)

Model	Guarantee
Strict	Read returns the latest write (needs global time)
Linearizable	Operations appear instant, in real-time order
Sequential	All see the same order, which respects each process's program order
Causal	Causally related writes seen in the same order by all
FIFO (PRAM)	Writes of one process seen in order
Eventual	If updates stop, all copies become equal over time

- **Primary-backup** (passive) vs **active replication** (all replicas process every request).
- **Quorum**: N replicas, read quorum R, write quorum W.

> **FORMULA** Quorum rules: $R + W > N$ (read sees latest write) and $W > N / 2$ (no two conflicting writes).

> **EXAMPLE** $N = 5$, $W = 3$, $R = 3$. $R + W = 6 > 5$ and $W = 3 > 2.5$, so both rules hold.

CAP theorem

- During a network **partition** a system must choose between **consistency** and **availability**.

7. Fault Tolerance and Agreement

- **Redundancy**: information (error codes), time (retry), physical (extra hardware / replicas).
- **Checkpointing**: save state regularly; roll back after failure. **Coordinated** checkpoints avoid the domino effect.
- **Message logging**: replay logged messages after recovery.

Commit protocols

- **Two-Phase Commit (2PC)**:
  1. Phase 1 (voting): coordinator sends PREPARE; each participant votes YES or NO.
  2. Phase 2 (decision): if all YES → COMMIT, else ABORT.

- Problem: blocking if the coordinator crashes after participants vote YES.
- **Three-Phase Commit (3PC)** adds a pre-commit phase to reduce blocking.

➤ Consensus

- All correct processes must agree on one value.
- **FLP result**: in a fully asynchronous system, no deterministic algorithm can guarantee consensus if even one process may crash.
- Practical algorithms: **Paxos**, **Raft** (leader election + log replication).

8. Distributed File Systems and Shared Memory

- **DFS** = files stored on many machines but used like local files. Examples: NFS, AFS, HDFS.
- **NFS**: stateless server (older versions), remote access model.
- **AFS**: whole-file caching on the client, callback on change.
- **HDFS**: big blocks (default 128 MB), each replicated (default 3); **NameNode** keeps metadata, **DataNodes** store blocks.
- **Distributed shared memory (DSM)**: gives the illusion of shared memory over machines without real shared memory.
- **Naming**: DNS is a hierarchical, distributed naming service.

9. Most Asked Questions (practice)

1. Define distributed systems. Explain goals and types of transparency.
2. Explain RPC with a diagram. What are RPC call semantics?
3. Explain Cristian's and Berkeley clock synchronisation.
4. Explain Lamport clocks and vector clocks with examples.
5. Compare distributed mutual exclusion algorithms.
6. Explain Bully and Ring election algorithms.
7. Explain consistency models. What is eventual consistency?
8. Explain the two-phase commit protocol and its problem.
9. What is the CAP theorem? Explain quorum-based replication.
10. Explain the architecture of HDFS or NFS.

TIP Lamport / vector clocks, election, mutual exclusion and 2PC appear in almost every paper. Draw the timeline diagrams neatly with timestamps.