

UNIT 6

Coordination

Distributed Systems (CAC352) — BCA III/VI

⚠ Please note

*These notes were prepared with AI assistance and **may contain errors or omissions**. Please cross-check definitions, algorithms and numerical examples against your prescribed textbook (e.g., Tanenbaum & van Steen, Distributed Systems), your class notes and your instructor before relying on them for exams.*

6.1 Clock Synchronization

In a distributed system each computer has its own clock and there is no shared global clock. Clocks drift: a hardware timer runs slightly faster or slower than real time. If a clock has **drift rate** ρ (max), then two clocks can differ by up to $2\rho\Delta t$ after time Δt . To keep two clocks within a bound δ , they must be resynchronised at least every $\delta / (2\rho)$ seconds.

Physical clocks

- **TAI** (International Atomic Time) — counted by atomic (caesium) clocks.
- **UTC** (Coordinated Universal Time) — TAI adjusted with leap seconds to follow the Earth's rotation; broadcast by radio and GPS.
- **GPS** satellites carry atomic clocks and send time signals; receivers achieve accuracy of tens of nanoseconds.

Cristian's algorithm

A client asks a **time server** (with a UTC receiver) for the time. If the request is sent at T_0 and the reply, containing server time T_s , arrives at T_1 , the client sets its clock to $T_s + (T_1 - T_0)/2$, assuming symmetric delays. The accuracy is within $\pm((T_1 - T_0)/2 - T_{\min})$, where $T_{\min}$ is the minimum one-way delay. A clock must never be set backwards abruptly; it is slowed down instead.

Example 6.1 — Cristian's algorithm

A client sends a request at $T_0 = 10:00:00.000$ and receives the reply at $T_1 = 10:00:00.400$. The reply says the server time was 10:00:00.150. What should the client set its clock to?

Solution

Round-trip = 0.400 s, so estimated one-way delay = 0.200 s.

New time = 10:00:00.150 + 0.200 = **10:00:00.350**.

Berkeley algorithm

No machine has a UTC source. A **coordinator (time daemon)** polls all machines for their times, computes the **average** (ignoring outliers), and tells each machine the *adjustment* (not the absolute time) to apply. This avoids the effect of message delay on absolute values.

Example 6.2 — Berkeley algorithm

The coordinator's clock reads 3:00, machine A reads 2:50 and machine B reads 3:25. Compute the adjustments.

Solution

Differences from coordinator: coordinator 0, A -10 min, B +25 min. Average = $(0 - 10 + 25)/3 =$ **+5 min**.

Adjustments: coordinator +5 min, A +5 - (-10) = **+15 min**, B +5 - 25 = **-20 min** (slow the clock down).

All clocks then read 3:05.

Network Time Protocol (NTP) and other approaches

NTP synchronises clients over the Internet using a hierarchy of servers called **strata**: stratum 0 = reference clocks (atomic/GPS), stratum 1 = servers directly attached to them, stratum 2 = clients of stratum 1, and so on. Typical accuracy: a few milliseconds on a LAN, tens of ms over the Internet. With four timestamps ($T_1 \dots T_4$) it estimates offset $\theta = ((T_2 - T_1) + (T_3 - T_4))/2$ and delay $\delta = (T_4 - T_1) - (T_3 - T_2)$. In **wireless sensor networks**, the **Reference Broadcast Synchronization (RBS)** protocol compares the times at which receivers hear the same broadcast, removing sender-side non-determinism.

6.2 Logical Clocks

Often only the **order** of events matters, not the real time. Lamport (1978) defined the **happened-before** relation $\rightarrow$:

- If a and b are events in the same process and a occurs before b, then $a \rightarrow b$.
- If a is the sending of a message and b is its receipt, then $a \rightarrow b$.

- Transitivity: if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$.
- If neither $a \rightarrow b$ nor $b \rightarrow a$, the events are **concurrent** ($a \parallel b$).

Lamport logical clocks

Each process P_i keeps a counter L_i . **Rules:** (1) before each event (including a send), $L_i = L_i + 1$; (2) a message carries its timestamp $t = L_i$; (3) on receipt, $L_j = \max(L_j, t) + 1$. The clock guarantees $a \rightarrow b \Rightarrow L(a) < L(b)$, but not the converse. To obtain a **total order**, break ties by process id: (L, i) .

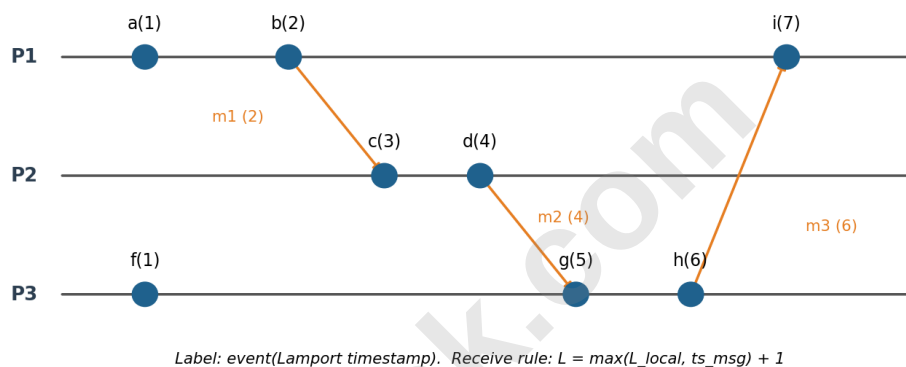


Figure 6.1 — Lamport timestamps for three processes.

Example 6.3 — Vector clocks

Three processes P_1, P_2, P_3 use vector clocks $V = (v_1, v_2, v_3)$. P_1 executes an internal event, then sends message m to P_2 . Independently, P_3 executes an internal event. Give the vector timestamps and determine whether P_2 's receive and P_3 's event are ordered.

Solution

Vector clock rules: increment own entry before each event; a message carries the sender's vector; on receive take the component-wise max, then increment own entry.

P_1 internal event: $(1,0,0)$. P_1 sends m : $(2,0,0)$. P_3 event: $(0,0,1)$.

P_2 receives m : $\max((0,0,0), (2,0,0)) = (2,0,0)$, then increment own entry $\rightarrow (2,1,0)$.

Compare $(2,1,0)$ and $(0,0,1)$: neither is $\leq$ the other, so the two events are **concurrent**. Vector clocks capture causality exactly: $V(a) < V(b) \Leftrightarrow a \rightarrow b$.

Vector clocks are used for **causally ordered multicast** (a message is delivered only after all messages that causally precede it), and in systems such as Dynamo-style stores to detect conflicting updates.

6.3 Mutual Exclusion

Processes must access a shared resource (critical section) one at a time. A good algorithm guarantees **safety** (at most one process at a time), **liveness** (no deadlock) and **fairness/no starvation**.

Centralised algorithm

A coordinator grants permission. A process sends REQUEST; if the resource is free, the coordinator replies GRANT; otherwise it queues the request. On exit the process sends RELEASE and the coordinator grants the next. **3 messages per entry**. Easy and fair, but the coordinator is a single point of failure and a bottleneck.

Distributed algorithm — Ricart–Agrawala

A process that wants the resource multicasts REQUEST(timestamp, id) to all others. A receiver replies OK immediately if it is not interested, or if the requester's (timestamp, id) is smaller (earlier) than its own request; otherwise it defers the reply until it leaves the critical section. The requester enters when it has received OK from **all $N - 1$** processes. Cost: **$2(N - 1)$ messages** per entry. Any crashed process blocks everybody, so it is even less robust than the centralised algorithm.

Token ring algorithm

Processes are arranged in a logical ring and a single **token** circulates. A process may enter the critical section only while holding the token, and passes it on when done. Messages per entry: 1 to unbounded (token circulates even if nobody wants it). Problems: token loss must be detected and regenerated; crash of a process breaks the ring.

Decentralised (voting) algorithm

Each resource is replicated n times, each with its own coordinator. A process needs permission from a **majority $m > n/2$** of coordinators. It tolerates coordinator crashes but may cause starvation when many processes compete.

Algorithm	Messages per entry/exit	Delay before entry (message times)	Problems
Centralised	3	2	Coordinator crash
Distributed (Ricart–Agrawala)	$2(N - 1)$	$2(N - 1)$	Crash of any process
Token ring	$1 \dots \infty$	$0 \dots N - 1$	Lost token, ring crash

Algorithm	Messages per entry/exit	Delay before entry (message times)	Problems
Decentralised (voting)	3mk (k tries)	2m	Starvation, low efficiency

6.4 Election Algorithms

Many algorithms need one process to act as coordinator. An **election algorithm** chooses one process, usually the one with the **highest id**, after the current coordinator crashes; all processes must agree on the result.

Bully algorithm

When process P notices the coordinator is down, it sends ELECTION to all processes with higher ids. If none answers, P wins and sends COORDINATOR to all. If a higher process answers OK, P stops and that process takes over the election. Eventually the highest live process becomes coordinator ('bullies' the others). Worst-case messages $O(N^2)$.

Example 6.4 — Bully algorithm

Processes 1–7; coordinator 7 crashes. Process 4 detects this. Show the election.

Solution

4 sends ELECTION to 5, 6, 7. 5 and 6 reply OK; 7 does not answer. 4 drops out.

5 sends ELECTION to 6, 7 → 6 replies OK. 6 sends ELECTION to 7 → no reply.

6 has no live higher process, so it wins and sends **COORDINATOR** to all. **6 is the new coordinator.**

Ring algorithm

Processes form a logical ring. A process that detects the failure sends an ELECTION message containing its id to its successor; each process appends its own id and forwards it (skipping crashed nodes). When the message returns to the initiator, it contains all live ids; the initiator picks the highest and circulates a COORDINATOR message announcing it. Two simultaneous elections give the same result.

Elections in wireless and large-scale systems

In wireless ad hoc networks, an election is done by building a spanning tree from the source and propagating the best (e.g., highest battery) candidate up the tree. In large P2P systems, a small set of **super-peers** is elected so that each has a bounded number of clients and they are well spread over the overlay.

6.5 Location Systems

Location systems determine the position of a node. **GPS** uses ≥ 24 satellites in orbit. A receiver measures the travel time of signals from several satellites and computes distances, i.e., **trilateration**: the intersection of spheres around the satellites. Three satellites determine a 3-D position if the receiver clock is perfect; because receiver clocks are cheap and inaccurate, a **fourth satellite** is used to solve for the clock offset as well.

- **Logical positioning** — network coordinates estimate latency between nodes (e.g., **Vivaldi** assigns each node a coordinate so that Euclidean distance approximates round-trip time), useful in choosing nearby replicas or peers.
- **Indoor / wireless positioning** — signal strength or time-of-flight of Wi-Fi, Bluetooth or cell towers, triangulated to infer location.
- **Location-based services** — geofencing, navigation, proximity search.

6.6 Distributed Event Matching

In publish–subscribe systems, **subscribers** register **subscriptions** and **publishers** send **notifications** (events). The system must **match** every notification to the subscriptions it satisfies and deliver it. In *topic-based* systems matching is by a named topic; in *content-based* systems subscriptions are predicates over attributes (e.g., *price < 100 AND category = 'books'*).

- **Centralised matching** — one broker stores all subscriptions; simple but a bottleneck.
- **Distributed matching** — a network of brokers; options are flooding notifications to all brokers, flooding subscriptions, or **selective routing** where brokers forward a notification only towards brokers that have matching subscriptions (subscriptions are aggregated to reduce state).
- **Partitioning** — the attribute space or subscription set is split among brokers (e.g., by hashing on a DHT) so that each broker matches a subset in parallel.

6.7 Gossip-Based Coordination

Gossip (epidemic) protocols achieve coordination through repeated, random pairwise exchanges. They are simple, scalable and robust to failures. Uses (beyond dissemination, see Unit 4):

- **Aggregation** — computing averages, sums, maxima or network size. Each node periodically picks a random peer and both replace their values with the average; all values converge exponentially fast to the global mean.
- **Peer-sampling service** — each node keeps a small **partial view** of the network and exchanges (part of) it with random neighbours, producing a continuously changing random overlay.
- **Overlay construction** — gossip with a ranking function converges to structured topologies (ring, tree, DHT) — e.g., T-Man.

Example 6.5 — Gossip averaging

Four nodes hold values 10, 20, 30, 40. Nodes gossip in pairs by averaging. Show convergence.

Solution

Global average = $(10 + 20 + 30 + 40)/4 = 25$.

Round 1: (10, 20) → 15, 15; (30, 40) → 35, 35.

Round 2: pair (15, 35) → 25, 25 and the other (15, 35) → 25, 25. All nodes now hold **25**.

The sum is conserved at every exchange, which is why the average is preserved.

Key Takeaways

- ✓ Physical clocks need synchronisation: Cristian (time server), Berkeley (average), NTP (strata).
- ✓ Lamport clocks give partial causal order ($a \rightarrow b \Rightarrow L(a) < L(b)$); vector clocks characterise causality exactly.
- ✓ Mutual exclusion: centralised (3 msgs), Ricart–Agrawala ($2(N-1)$ msgs), token ring, voting — each with different failure weaknesses.
- ✓ Election: bully (highest live id wins), ring (circulate list of ids).
- ✓ GPS uses trilateration with 4 satellites; Vivaldi gives logical coordinates.
- ✓ Event matching routes notifications to matching subscriptions; gossip achieves aggregation and overlay construction with simple random exchanges.

— End of Unit 6 —