Time in Distributed Systems

1 Introduction

Time is a fundamental concept in distributed systems. Unlike centralized systems where a single reference clock can be used, distributed systems face unique challenges due to physical separation of components and network delays.

1.1 Needs for Precision Time

Precise time synchronization is critical for many distributed applications:

- **Stock market buy and sell orders** - Ensuring fair ordering of trades, preventing fraud, and maintaining transaction order integrity
- **Secure document timestamps** (with cryptographic certification) - Providing non-repudiation for legal documents and contracts
- **Distributed network gaming and training** - Maintaining consistent game state across all players
- **Aviation traffic control and position reports** - Ensuring safety through accurate position tracking
- **Multimedia synchronization for real-time teleconferencing** - Aligning audio and video streams across multiple endpoints
- **Event synchronization and ordering** - Critical for distributed databases and transaction processing
- **Network monitoring, measurement and control** - Accurate performance metrics and troubleshooting
- **Distributed algorithms** - Many algorithms require synchronized time or logical ordering
- **Cybersecurity** - Detecting attacks and correlating security events across systems

Deep Dive

Time synchronization issues are at the heart of many distributed system challenges. Consider blockchain technology, where consensus mechanisms often rely on timestamps to order transactions. Bitcoin, for example, includes a timestamp in each block to establish chronological order. However, to prevent timestamp manipulation, the protocol implements rules regarding how much a timestamp can deviate from the network's perceived time.
Similarly, in high-frequency trading, microsecond differences in timing can translate to millions of dollars. Financial exchanges implement sophisticated timestamping mechanisms and regulations (like MiFID II in Europe) mandate synchronization to UTC with microsecond precision.

Interactive Example

Consider a distributed auction system running across multiple datacenters globally. Bidders submit bids from different locations, and the system must determine the winning bid based on arrival time.

- What happens if Server A and Server B have clocks that differ by 200ms?
- How would this impact fairness if a bid arrives at Server A at its local time 10:15:30.100 and another bid arrives at Server B at its local time 10:15:30.000?
- What mechanisms could you implement to ensure fair ordering of bids?

Practice Exercise

Exercise 1: Consider a distributed banking system where transactions need to be processed in the correct order across multiple server nodes.

(a) Describe three specific problems that could occur if the clocks of the servers are not synchronized.

(b) For each problem, propose a solution that either relies on clock synchronization or uses a logical ordering mechanism instead.

(c) A bank processes transactions at two data centers located 1000 km apart. If the network latency between sites is typically 10ms, what is the minimum precision of clock synchronization needed to ensure transactions are ordered correctly?

2 Physical Time

Before discussing time in distributed systems, we need to understand the physical basis of time and how it's measured and distributed.

2.1 Definitions of Time

Time has been defined and measured in progressively more precise ways throughout history:

1. **Solar Time**
    - Based on earth's rotation relative to the sun
    - Today: 1 sec $\sim$ 1 day / 86400
    - Problem: Earth's rotation is not constant and gradually slows down (by about 1.7 milliseconds per century)
    - Results in unpredictable variations that make it unsuitable for precise timing
2. **Atomic Time**
    - Based on state transitions in atoms (defined and maintained by BIH in Paris)
    - 1 second = time a cesium-133 atom needs for exactly 9,192,631,770 state transitions
    - This defines TAI (International Atomic Time)
    - A TAI-day is about 3 msec shorter than an astronomical day
    - BIH (Bureau International de l'Heure) inserts leap seconds when the difference between a day and a TAI-day exceeds 800 msec
    - This adjusted time is UTC (Universal Time Coordinated), the global reference for time measurement

Deep Dive

Modern atomic clocks operate by measuring the resonant frequency needed to make electrons in atoms transition between energy levels. The most advanced atomic clocks can measure time with a precision of 1 second in 15 billion years.
The Global Positioning System (GPS) relies heavily on precise time synchronization. Each satellite contains multiple atomic clocks and broadcasts time signals. Since radio waves travel at the speed of light, a timing error of just 1 nanosecond would result in a positioning error of about 30 cm.
Time standards organizations like NIST (US), PTB (Germany), and NPL (UK) maintain collections of atomic clocks and contribute to the international time standard. The global UTC is actually a weighted average of over 400 atomic clocks worldwide.

2.2 UTC Broadcasts

Multiple mechanisms exist to distribute the accurate UTC time globally:

- **Shortwave radio broadcasting stations**
  - Examples: WWV (US), MSF (UK), DCF77 (Germany)
  - Typical accuracy: 1.0 msec
  - Vulnerable to atmospheric conditions
- **Satellite-based distribution**
  - GPS, GLONASS, Galileo constellations
  - Extremely accurate: 1.0 μsec or better
  - Global coverage
  - Requires line-of-sight to satellites
- **Internet-based distribution**
  - NTP (Network Time Protocol) servers
  - Variable accuracy, typically >> 1ms
  - Dependent on network conditions
- **Phone line services**
  - Dial-up time services
  - Limited accuracy (>> 1ms)
  - Mostly obsolete today
- **Commercially available receivers**

- Can be connected to computer systems
- Range from inexpensive DCF77 receivers to precision GPS timing receivers
- Used in critical infrastructure, financial systems, and telecommunications

Interactive Example

Let's explore how UTC time distribution affects a global banking system:

- Bank A uses GPS-based time synchronization (1.0 μsec accuracy)
- Bank B uses NTP over the internet (10 ms typical accuracy)
- Bank C uses radio-based synchronization (1.0 msec accuracy)

For each bank:

1. What is the maximum potential time discrepancy between its branches globally?
2. How does this affect high-frequency trading systems that execute transactions in microseconds?
3. What type of financial transactions might be problematic with each synchronization method?

Practice Exercise

Exercise 2: Time Signal Propagation and Accuracy

(a) Calculate the theoretical maximum time synchronization error for a system using GPS time signals. Consider that:

- GPS satellites orbit at approximately 20,200 km
- Radio signals travel at the speed of light (300,000 km/s)
- Atmospheric effects can delay signals by up to 100 ns

(b) A datacenter wants to synchronize all servers to within 1 microsecond. Compare and evaluate three different time synchronization technologies that could achieve this precision. For each, identify potential failure modes and necessary redundancy measures.

(c) Research and report on how your country's critical infrastructure (power grid, telecommunications, etc.) maintains time synchronization. What would be the impact of a 1-second error in synchronization?

3 Computer Clocks

Each computer in a distributed system has its own time-keeping mechanism, which presents fundamental challenges for coordination.

3.1 Physical Clock Architecture

- Each node has its own private **physical clock**
- Physical clocks are hardware devices typically based on:
  - **Quartz crystals** - Most common, affordable but less precise
  - **Temperature-compensated crystal oscillators (TCXOs)** - Better stability
  - **Oven-controlled crystal oscillators (OCXOs)** - High precision, used in servers
  - **Rubidium oscillators** - Very high precision, expensive
- The crystal/oscillator vibrates at a specific frequency (e.g., 32.768 kHz)
- After a specified number of oscillations, the clock increments a register, adding one **clock-tick** to a counter that represents the passing of time: $H_i(t)$
- **Resolution**: Period between clock updates - modern systems typically offer nanosecond resolution

3.2 Hardware to Software Clock Translation

The operating system maintains a software clock by converting the hardware clock ticks:

$$C_i(t) = \alpha H_i(t) + \beta \tag{1}$$

Where:

- $C_i(t)$ is the software clock time at node i
- $H_i(t)$ is the hardware clock counter (number of ticks)
- α is a scaling factor (to convert ticks to time units)
- β is an offset (for setting the clock to a specific time)

This software clock:

- Approximates the physical time t at process p_i
- Is typically implemented as a 64-bit word in modern systems
- Represents time in nanoseconds since an epoch (e.g., January 1, 1970)
- Can be adjusted by synchronization protocols

3.3 Clock Resolution and Event Distinction

- Successive events can only be distinguished if the **clock resolution** is smaller than the time interval between events
- Modern computer clocks typically have resolutions in the range of:
  - Desktop/server systems: 1-100 nanoseconds
  - Embedded systems: 1 microsecond to 1 millisecond
  - IoT devices: Can be as coarse as 10 milliseconds
- High-performance computing and financial systems may require sub-nanosecond precision

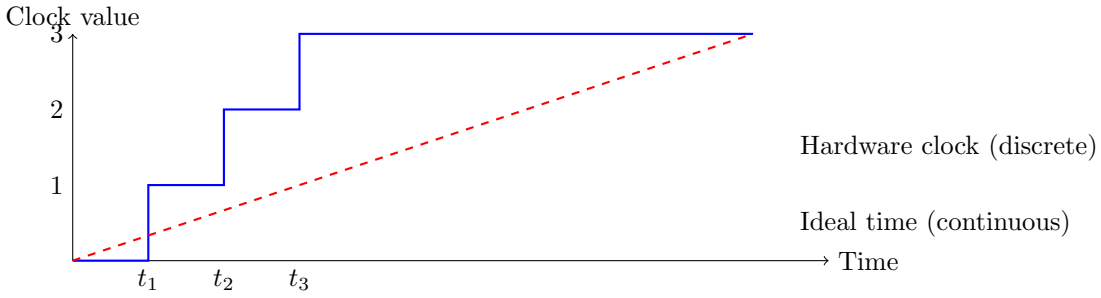

Figure 1: Digital clock advancing in discrete steps versus continuous time

Deep Dive

Different operating systems handle time in unique ways:
Linux:

- Maintains multiple clocks: CLOCK_REALTIME (wall clock time), CLOCK_MONOTONIC (monotonically increasing time), CLOCK_PROCESS_CPUTIME_ID (per-process CPU time)
- Uses a combination of hardware timers like High Precision Event Timer (HPET), Time Stamp Counter (TSC), and others
- Recent kernels support nanosecond resolution

Windows:

- Provides GetSystemTimeAsFileTime() for wall clock time and QueryPerformanceCounter() for high-resolution timing
- Internally uses ACPI Power Management Timer or TSC

Clock Synchronization Mechanisms:

- Both the scaling factor (α) and offset (β) can be adjusted during synchronization
- Modern systems often use adaptive algorithms that adjust the rate of the clock (changing α) rather than making abrupt changes to the time (changing β)

Interactive Example

Let's analyze real computer clock behavior:
Imagine we have a system with a quartz crystal oscillating at 32.768 kHz (typical for many computers):

1. How many oscillations occur in 1 second?
2. If the counter increments after every oscillation, what is the theoretical resolution of this clock?
3. If the system represents time in nanoseconds using a 64-bit integer, how many years can it represent before overflow?
4. If a high-performance algorithm needs to distinguish events that are 100 microseconds apart, is this clock sufficient? What if events are 10 microseconds apart?

Now consider the implications of scaling: If $\alpha = 1.0000001$ (very slight error), how much time error accumulates after one day?

Practice Exercise

Exercise 3: Computer Clock Implementation and Analysis

(a) A computer uses a crystal oscillator with a frequency of 14.318 MHz. If the hardware counter increments after every 12 oscillations:

- What is the resolution of the hardware clock in microseconds?
- How many ticks occur in one minute?
- If the crystal has a drift rate of 10ppm (parts per million), what is the maximum time error after 24 hours?

(b) The operating system sets its software clock with $\alpha = 30.517$ and $\beta = 1577836800000000000$ (nanoseconds since epoch for January 1, 2020).

- Write the equation for the software clock $C(t)$ in terms of hardware ticks $H(t)$
- If the hardware clock shows 1,000,000 ticks, what time (in seconds since epoch) does the software clock show?
- If we need to adjust the clock forward by 1.5 seconds, what new value of β should be used?

(c) Consider a system that generates events at an average rate of 10,000 events per second:

- What is the probability that two events occur within the same clock tick if the clock resolution is 1ms? 100s? 1s?
- How does this impact event ordering in a distributed system?
- Design a solution that ensures correct event ordering even when events occur within the same clock tick.

4 Drift and Skew

Even high-quality clocks are imperfect, leading to two critical challenges in distributed systems: drift and skew. These phenomena fundamentally limit the accuracy of time synchronization in distributed systems.

4.1 Basic Concepts

- Computer clocks, like any other clocks, tend not to be in perfect agreement!
- **Clock skew (offset)**: The instantaneous difference between the times

shown on two clocks at a given point in real time

$$\text{Skew}(t) = |C_i(t) - C_j(t)| \tag{2}$$

- **Clock drift**: The difference in the rate at which clocks count time
  - Ordinary quartz clocks drift by $\sim$ 1sec in 11-12 days (approximately 10^{-6} secs/sec or 1 PPM)
  - High precision quartz clocks drift rate is $\sim$ 10^{-7} or 10^{-8} secs/sec (0.1-0.01 PPM)
  - Atomic clocks may have drift rates down to 10^{-13} secs/sec

4.2 Causes of Clock Drift

Several factors contribute to clock drift:

- **Temperature variations** - Crystal oscillation frequency changes with temperature
- **Crystal aging** - Oscillation properties change over time
- **Power supply voltage fluctuations** - Affects oscillator behavior
- **Manufacturing differences** - No two crystals are identical
- **Environmental factors** - Vibration, shock, humidity, radiation, etc.

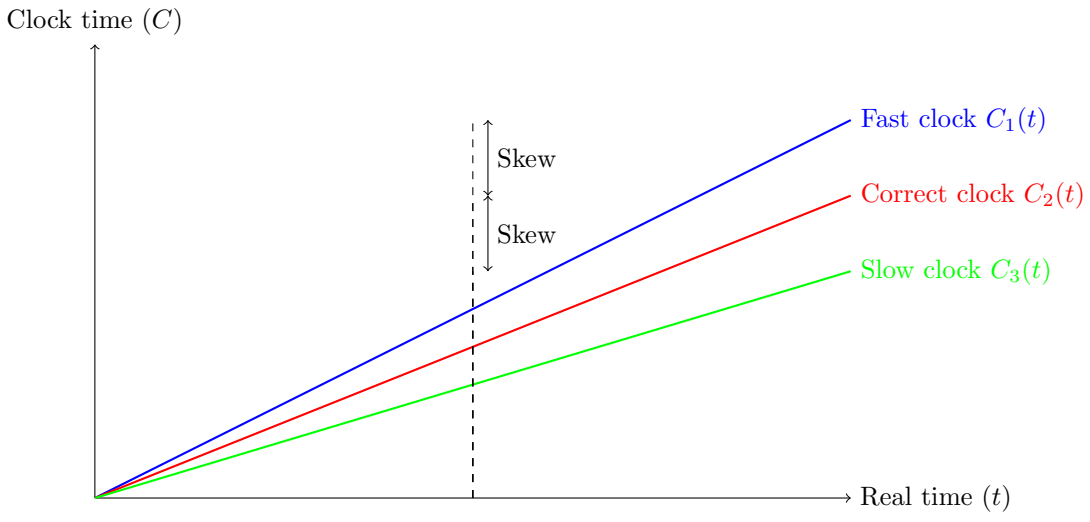

Figure 2: Clock drift and skew illustrated with three clocks running at different rates

4.3 Mathematical Model of Clock Drift

Clock manufacturers specify a maximum drift rate ρ (rho) in secs/sec or parts per million (PPM):

$$1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho \tag{3}$$

Where:

- $C(t)$ is the clock's reported time as a function of the real time t
- $\frac{dC}{dt}$ is the rate at which the clock advances
- A perfect clock would have $\frac{dC}{dt} = 1$ (advancing at exactly the rate of real time)
- A fast clock has $\frac{dC}{dt} > 1$, a slow clock has $\frac{dC}{dt} < 1$

4.4 Implications for Synchronization

If two clocks have a maximum drift rate of ρ, and we want to keep them synchronized within a maximum skew of δ, we must resynchronize at least every:

$$\frac{\delta}{2\rho} \text{ seconds} \tag{4}$$

This formula is derived as follows:

- In the worst case, one clock runs at rate $(1 + \rho)$ while the other runs at rate $(1 - \rho)$
- The relative drift between them is 2ρ per second
- To maintain a maximum skew of δ, we must resynchronize before the accumulated drift reaches δ
- Time to reach skew $\delta = \frac{\delta}{2\rho}$

Example: For clocks with a drift rate $\rho = 10^{-6}$ that need to be synchronized within $\delta = 10ms$:

$$\frac{10 \times 10^{-3}}{2 \times 10^{-6}} = 5000 \text{ seconds} = 83.33 \text{ minutes} \tag{5}$$

Deep Dive

The physics behind crystal oscillators explains much of their behavior: Quartz is a piezoelectric material, meaning it generates an electric charge when mechanical stress is applied, and conversely, it deforms when an electric field is applied. This property creates a feedback loop that sustains oscillation at a specific resonant frequency determined by the crystal's physical dimensions and structure.
Temperature affects the crystal's elasticity and dimensions, changing its resonant frequency. This is why high-precision timekeeping often uses temperature-compensated or oven-controlled oscillators:

- **Temperature-Compensated Crystal Oscillators (TCXOs)** use circuit components with temperature coefficients that counteract the crystal's temperature dependence.
- **Oven-Controlled Crystal Oscillators (OCXOs)** maintain the crystal at a constant temperature, often slightly above ambient, to eliminate temperature fluctuations.

In ultra-precise applications like GPS satellites, atomic clocks are used, which rely on the resonant frequency of atoms (typically cesium, rubidium, or hydrogen) and are much less susceptible to environmental effects. The military and scientific research often require extremely precise timing. The US Naval Observatory, for instance, maintains a "Master Clock" based on an ensemble of more than 100 atomic clocks. This system provides timing with an estimated uncertainty of about 100 picoseconds per day.

Interactive Example

Let's explore the practical implications of clock drift:
Imagine two distributed database servers that initially set their clocks to exactly the same time.

- Server A's clock has a drift rate of +2 ppm (runs fast by 2 microseconds per second)
- Server B's clock has a drift rate of −3 ppm (runs slow by 3 microseconds per second)

1. What will be the clock skew between these servers after 1 hour?
2. After 1 day?
3. After 1 week?
4. If transactions need to be ordered with a precision of 10ms, how frequently must these servers resynchronize their clocks?
5. If server B's time is used to generate timestamps for a security audit log, how much time discrepancy could an attacker exploit over a weekend (60 hours) if they know about this drift?

Working through the solution:

- Combined drift rate: $2 + 3 = 5$ ppm $= 5 \times 10^{-6}$ sec/sec
- Skew after 1 hour: $5 \times 10^{-6} \times 3600 = 0.018$ seconds $= 18$ ms
- Skew after 1 day: $5 \times 10^{-6} \times 86400 = 0.432$ seconds $= 432$ ms
- Skew after 1 week: $5 \times 10^{-6} \times 604800 = 3.024$ seconds
- To maintain 10ms precision: Resync every $\frac{10 \times 10^{-3}}{5 \times 10^{-6}} = 2000$ seconds $= 33.33$ minutes
- Potential time discrepancy after 60 hours: $3 \times 10^{-6} \times 60 \times 3600 = 0.648$ seconds $= 648$ ms

Practice Exercise

Exercise 4: Clock Drift Analysis and Mitigation

(a) A distributed system consists of four nodes with the following clock drift rates:

- Node A: +1.5 ppm
- Node B: −2.0 ppm
- Node C: +0.5 ppm
- Node D: −1.0 ppm

Calculate:

- The maximum clock skew that could develop between any two nodes after 24 hours without synchronization
- The maximum skew after 1 week
- The synchronization interval required to keep all clocks within 5ms of each other

(b) Design a synchronization strategy for a distributed system where:

- The system consists of 100 nodes spread across 5 different data centers
- Each data center has its own GPS receiver for time synchronization
- The maximum drift rate of any node is 2 ppm
- The system requires synchronization within 1ms for correct operation
- Network latency between data centers ranges from 20ms to 200ms
- Network latency within a data center is less than 1ms

Your design should include:

- Hierarchical structure for time distribution
- Synchronization intervals at each level
- Methods to handle network jitter and asymmetric delays
- Failover mechanisms if a GPS receiver fails

(c) A critical financial trading system requires that all events be ordered with microsecond precision. The system uses servers with clocks that have a maximum drift rate of 0.1 ppm.

- What is the maximum time between synchronizations to maintain this precision?
- If syncing the clocks takes 10ms and introduces a system pause, what percentage of system time would be spent on synchronization?
- Design an alternative approach using logical clocks that could reduce the synchronization overhead while maintaining correct event ordering.

Partial solutions:

5 Synchronization

5.1 Internal/External Synchronization

5.1.1 External synchronization

- Synchronization of process' clocks C_i with an authoritative external source S.
- Let $\delta > 0$ be the synchronization bound and S be the source of UTC.
- Then $|S(t) - C_i(t)| < \delta$ for $i = 1, 2, \ldots, N$ and for all real times t.
- We say that clocks C_i are **accurate** within the bound of δ

5.1.2 Internal synchronization

- Synchronization of process' clocks C_i with each other.
- Let $\delta > 0$ be the synchronization bound and C_i and C_j are clocks at processes p_i and p_j, respectively.
- Then $|C_i(t) - C_j(t)| < \delta$ for $i, j = 1, 2, \ldots, N$ and for all real times t.
- We say that clocks C_i, C_j **agree** within the bound of δ

Note that clocks that are internally synchronized are not necessarily externally synchronized. i.e., even though they agree with each other, they drift collectively from the external source of time.

5.2 Synchronization in a Synchronous System

In a synchronous system we have:

- Known upper (max) and lower (min) bound for communication delay.
- Known maximum clock drift.
- Known maximum time taken for each computational step.

We synchronize by:

- Time server sends its local time t_s to a client.
- Ideally, client sets clock to $t_s + T_{tran}$ (Unknown!)
- The client sets its local clock to $t_s + (max + min)/2$.
- Skew is at most $(max - min)/2$

5.3 Synchronization in an Asynchronous System

- Cristian's algorithm
- The Berkeley algorithm
- Network time protocol (NTP)

5.3.1 Cristian's Algorithm

- A node is a time server TS (presumably with access to UTC). How can the other nodes be synced?
- Periodically, at least every $\delta/2\rho$ seconds, each machine sends a message to the server asking for the current time and the TS responds.

Steps:

- Client p sends request (m_r) to time server S.
- S inserts its time t immediately before reply (m_t) is returned.
- p measures how long it takes ($T_{round} = T_1 - T_0$) from m_r is sent to m_t is received.
- p sets its local clock to $t + T_{round}/2$.

5.3.2 Accuracy of Cristian's Algorithm

- Assume min = minimal message delay
- t is in the interval $[T_0 + min, T_1 - min]$
- Uncertainty on $t = T_{round} - 2min$
- Estimated Accuracy $= T_{round}/2 - min$

Notes on Cristian's Algorithm:

- Monotonicity:
  - Jumps in time backward not permitted
  - Jumps forward may be confusing
  - Receiver adjusts clock rate α: $C_i(t) = \alpha H_i(t) + \beta$
- Improve precision:
  - By taking several measurements and taking the smallest round trip
  - Or use an average after throwing out the largest values

5.3.3 The Berkeley Algorithm

Designed for internal synchronization.

(a) The time daemon asks all the other machines for their clock values

(b) The machines answer their offset

(c) The time daemon tells each how to adjust its clocks

5.3.4 Network Time Protocol (NTP)

- Synchronization of clients relative to UTC on an internet-wide scale
- Reliable, even in the presence of extensive loss of connectivity
- Allow frequent synchronization (relative to clock drift)
- Tolerant against disturbance
- <1ms within LAN
- 1-10 ms internet scale

NTP Stratum

- Never synchronize with servers at lower stratum

NTP Modes

- Multicast (for quick LANs, low accuracy)
  - Server periodically sends its actual time to all clients/leaves in the LAN
- Procedure-call (medium accuracy)
  - Server responds to requests with its actual timestamp
  - Like Cristian's algorithm
- Symmetric mode (high accuracy)
  - Used to synchronize between pairs of servers with resp. high and low stratum
  - In all cases, the UDP is used

Messages exchanged between a pair of NTP peers

- Exchanges local timestamps to estimate offset o_i and delay d_i:

$$o_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2 \tag{6}$$

$$d_i = T_{i-2} - T_{i-3} + T_{i-1} - T_i \tag{7}$$

- NTP server filters pairs $\langle o_i, d_i \rangle$, saves 8 latest
- Use the o_i with smallest d_i (the smaller delay, the better accuracy)

6 Logical Time and Logical Clocks

6.1 Theoretical Foundation

- Inherent characteristic of a distributed system:
  - Absence of a global clock
  - Absence of 100% accurately synchronized clocks
- Impact: Due to the absence of global clocks, it is difficult to reason about the temporal ordering of events in distributed system, e.g., scheduling of events is more difficult.

6.2 Logical Clocks in a Distributed System

- What is important is usually not when things happened but in what order they happened. An integer counter works well in a centralized system.
- However, in a DS, each system has its own logical clock, and you can run into problems if one "clock" gets ahead of others (like with physical clocks).
- RELIABLE WAY OF ORDERING EVENTS IS REQUIRED
- We need a rule to synchronize the logical clocks

6.3 Event Ordering

Example with replicated database:

- A=100; B=100
- A'=(100+100)+10%=220
- B'=(100+10%)+100=210
- Updates need to be performed in the **same order** at all sites of a replicated database.

6.4 Events and Logical Clocks

- Leslie Lamport's 1978 paper: *Time, Clocks, and the Ordering of Events in Distributed Systems*
  - Theoretical Foundation
  - Logical Clocks
  - Partial and Total Event Ordering
  - Towards distributed mutual exclusion
  - MUST KNOW FOR ANY COMPUTER SCIENTIST

6.4.1 System Model

- A distributed system is a collection of sequential processes p_i, $i = 1, 2, \ldots N$.
- A process p_i has state s_i
- Each process executes a sequence of actions:
  - Sending a message
  - Receiving a message
  - Performing an internal computation that alters its state
- The sequence of events within a single process is totally ordered $e_i \rightarrow e_i'$
- The history of process p_i is the sequence of events that takes place therein $history(p_i) = h_i = \langle e_i^0, e_i^1, e_i^2, \ldots \rangle$

6.4.2 Happened-Before Relation

- The happened-before relation captures the dependencies between events, denoted $\rightarrow$:
  1. $a \rightarrow b$ if a and b are events in the same process, and a occurred before b.
  2. $a \rightarrow b$ if a is the event of sending a message m by one process and b is the event of receipt of the same message m by another process.
  3. If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$, i.e., happened-before relation is transitive.
- That is, past events causally affect future events.

6.4.3 Concurrent Events

- Two distinct events a and b are concurrent $(a||b)$ if not ($a \rightarrow b$ or $b \rightarrow a$).
- We cannot say whether one event happened-before another.
- For any two events a and b in a distributed system, either $a \rightarrow b$, $b \rightarrow a$, or $a||b$.

6.4.4 Logical Clocks

- There is a clock C_i at each process p_i
- The clock C_i can be thought of as a function that assigns a number $C_i(a)$ to any event a, called the timestamp of event a, at p_i
- These clocks can be implemented by counters and have no relation to physical time.

6.4.5 Conditions Satisfied by System of Clocks

- For any events a and b: if $a \rightarrow b$, then $C(a) < C(b)$
- This implies the following two conditions:

  C1 For any two events a and b in a process P_i, if a occurs before b, then $C_i(a) < C_i(b)$.

  C2 If a is the event of sending a message m by process P_i and b is the event of receiving the same message by process P_j, then $C_i(a) < C_j(b)$.

6.4.6 Implementation Rules

IR1 Before P_i timestamps an event: $C_i := C_i + 1$

IR2a P_i sends m: [IR1] and piggy-back timestamp $t = C_i$: $m' = \langle m, t \rangle$

IR2b P_j receives $m' = \langle m, t \rangle$: $C_j := \max(C_j, t)$, followed by [IR1]

6.4.7 Total Ordering of Events

- Lamport's happened-before relation defines an irreflexive partial order among the events
- Total ordering (denoted by $\Rightarrow$) can be obtained by using process-id to break tie if timestamps are equal:
- $a \Rightarrow b$ iff

  1. $C_i(a) < C_j(b)$ or
  2. $C_i(a) = C_j(b)$ and $P_i < P_j$

- Allows processes to agree on order everywhere based on timestamp

6.5 Vector Clocks

- Lamport: $e \to f$ implies $C(e) < C(f)$
- Vector clocks: $e \to f$ iff $C(e) < C(f)$
- Allows nodes to order events in happens-before order based on timestamps
- Vector timestamps: Each node maintains an array of N counters
  - $V_i[i]$ is the local clock for process p_i
  - In general, $V_i[j]$ is the latest info the node has about what p_j's local clock is.

6.5.1 Implementation Rules for Vector Clocks

VC1 Initially $V_i[j] = 0$ for $i, j = 1 \dots N$

VC2 Before P_i timestamps an event: $V_i[i] := V_i[i] + 1$

VC3 P_i sends m: piggy-back vector timestamp $t = V_i$: $m' = \langle m, t \rangle$

VC4 P_j receives $m' = \langle m, t \rangle$: $V_i[j] := \max(V_i[j], t_i[j])$, $I \neq j$

6.5.2 Comparison of Vector Clocks

- $V = V'$ iff $V[j] = V'[j]$ for all $j = 1, 2, \dots, N$
- $V \leq V'$ iff $V[j] \leq V'[j]$ for all $j = 1, 2, \dots, N$
- $V < V'$ iff $V \leq V'$ and $V \neq V'$