



Adama Science and Technology University
School of Electrical Engineering & Computing
Department of Computer Science and Engineering

Introduction to Distributed Systems

Text Book:

S. Tanenbaum and Maarten Van Steen, Distributed Systems: Principles and Paradigms, 2nd Ed.

Compiled by: Sufian K.

E-mail: sufainedir156@gmail.com

Chapter 5

Synchronization

Introduction

- In Ds the machine or node work independently
 - but sometime it they needs to cooperate or synchronize

Introduction

- In DSs the processes work independently
 - but sometime it they needs to cooperate or synchronize
- Cooperation is partly supported by naming; it allows processes to at least share resources (entities)
- Synchronization deals on how to ensure that **processes do not simultaneously access a shared resource;**
 - how events can be ordered such as two processes sending messages to each other.
 - **Coordination of action b/n processes.**
- **Synchronization** -Setting the time order of the set of events caused by concurrent processes.

Introduction

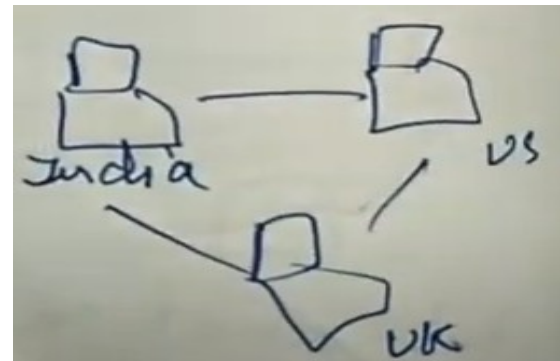
- Synchronization is vital in distributed systems to ensure **consistent and coordinated** behavior across multiple nodes. Reducing time and lose of data
- It helps maintain data integrity, prevent race conditions, and enforce ordering guarantees.
- By synchronizing their actions, **distributed nodes** can communicate, share resources, and achieve a consistent state despite the challenges of network delays, failures, and concurrency.

Communication & Synchronization

- **Why do processes communicate in DS?**
 - To exchange messages
 - To synchronize processes
- **Why do processes synchronize in DS?**
 - To coordinate access of shared resources (mutual exclusion)
 - To order events

Clock Synchronization

- It is the mechanism to synchronize the time of all computers in the distributed environments(DS).
- Synchronizing time of all computers in DS is important.
 - Physical clock
 - Logical clock
 - Mutual exclusion



Clock Synchronization Principles

- **Principle I:** Every machine asks a **time server** for the accurate time at least once every $\delta/2\rho$ seconds (NTP)
- But you need an accurate measure of round trip delay, including interrupt handling and processing incoming messages.
- **Principle II:** Let the time server scan all machines periodically, calculate an average, and inform each machine how it should adjust its time relative to its present time.
- Ok, you'll probably get every machine in sync. Note: you don't even need to propagate UTC time (why not?)

Physical Clock

Problem: Clock Skew – clocks gradually get out of synch and give different values

Solution: Universal Coordinated Time (UTC): Formerly GMT

- Based on the number of transitions per second of the cesium 133 atom (very accurate).
- At present, the real time is taken as the average of some 50 cesium-clocks around the world – International Atomic Time
- Introduces a leap second from time to time to compensate that days are getting longer.
- UTC is **broadcasted** through short wave radio (with the accuracy of +/- 1 msec) and satellite (with the accuracy of +/- 0.5 msec).

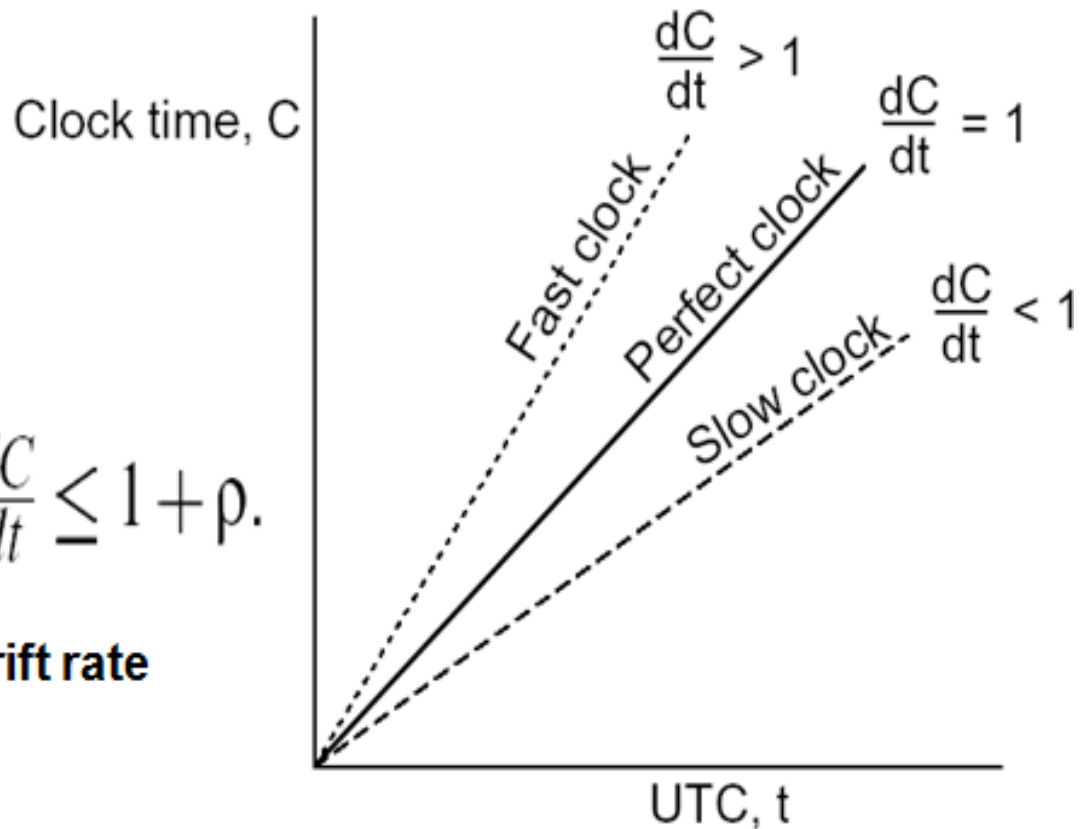
...Cont'd

- **Problem:** Suppose we have a distributed system with a UTC-receiver somewhere in it, we still have to distribute its time to each machine.
- **Basic principle:**
 - Every machine has a timer that generates an interrupt H (typically 60) times per second.
 - There is a clock in machine p that **ticks** on each timer interrupt. Denote the value of that clock by $C_p(t)$, where t is UTC time.
 - Ideally, we have that for each machine p , $C_p(t) = t$, or, in other words, $dC/dt = 1$

...Cont'd

In practice: $1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho$.

Where ρ is the max. drift rate

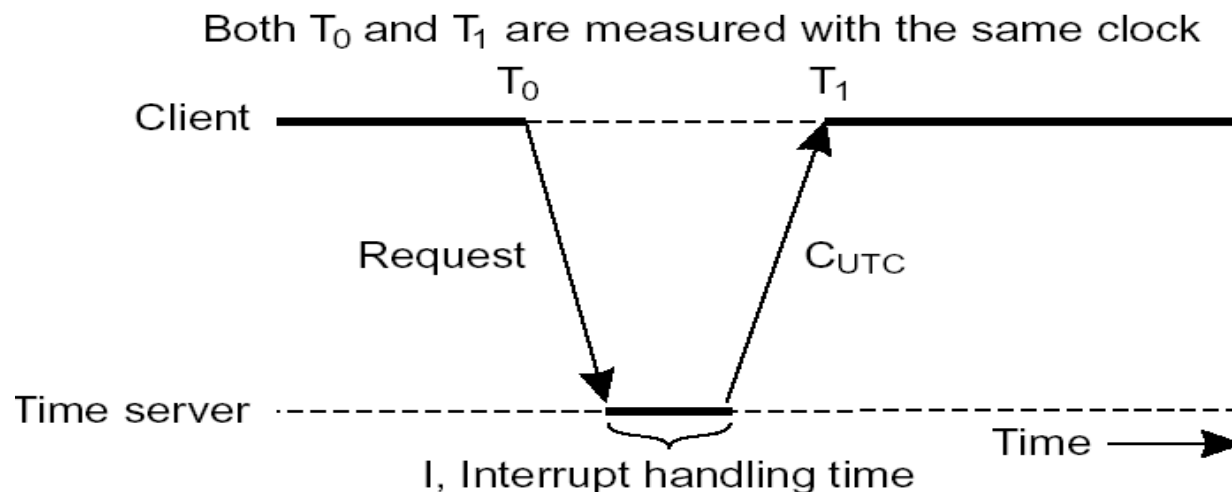


Goal: Never let two clocks in any system differ by more than δ time units => synchronize at least every $\delta/2\rho$ seconds.

Clock Synchronization Algorithm

- **Critian's Algorithm**

- Suitable when machine has UTC receiver; machine called Time Server.
- no more than every $\delta/2\rho$ seconds, each machine sends a message to the time server asking the current time
- Message propagation: Estimate $(T_1 - T_0)/2 + C_{UTC}$
- Time must never run backward: Introduce change gradually



...Cont'd

- **The Berkeley Algorithm**

- Suitable when machine has no UTC
- The time server asks periodically every machine for its time
- The received times are averaged and each machine is notified of the amount of the time it should adjust

Logical Clocks

- ❑ Internal consistency of the clocks rather than being close to the real time is required = Logical Clocks
- ❑ All processes should agree on time - which comes first
- ❑ Lamport Timestamp(“**happens before**”) relation
 - $a \rightarrow b$ = “a happens before b”; means all processes **agree** that first event **a** occurs, then event **b** occurs
 - **a** and **b** are events in same process, and **a** occurs before **b**, then $a \rightarrow b$ is true
- **happens before** is a transitive relation
 - if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$
- for every event **a**, we can assign a time value **C(a)** on which all processes agree; if $a \rightarrow b$, then $C(a) < C(b)$

...Cont'd

- ❑ E.g. Totally Ordered Multicasting(Bank) – All messages are delivered in same order to each receiver
 - assume sender sends message and no message is lost
 - all messages are put in a local queue **ordered** by **timestamp**
 - each receiver **multicasts** an acknowledgement
 - then all processes will have the same copy of the local queue

Mutual Exclusion(ME)

- **Problem:** A number of processes in a distributed system want exclusive access to some resource.
- **Basic solutions:**
 - Via a centralized server.
 - Completely distributed, with no topology imposed.
 - Completely distributed, making use of a (logical) ring.
- **In ME – Process is excluded from accessing shared resources except allowed one**

3-Algorithms to implement ME in DS

1. **Centralized Algorithm** – Coordinator is appointed and is in charge of granting permissions.
 - 3 Messages required – request, grant, release
2. **Distributed Algorithm** – Total ordering of all events.
 - Process enters critical region(CR) when it gets reply
 - When it finishes, sends OK message to all
 - 2 processes can't enter CR at the same time
3. **Token Ring** - Organize processes in a **logical** ring, and let a token be passed between them. The one that holds the token is allowed to enter the critical region-if it wants to

