CT30A3401
Distributed Systems
Lecture 6

Bilal Naqvi, PhD.
syed.naqvi@lut.fi

Coordination

- Due to distributed nature and need for communication between processes two aspects are of prime importance
  - **Synchronization**
    - Process synchronization: involves making sure that one process waits for another to complete its operation
    - Data synchronization: involves making sure that two sets of data are the same
  - **Coordination**
    - The goal is to manage the interactions and dependencies between activities in a distributed system
      - coordination in distributed systems is often much more difficult compared to that in uniprocessor or multiprocessor systems

Clock synchronization

- In a centralized system, time is unambiguous
- When a process wants to know the time, it simply makes a call to the operating system
- If process A asks for the time, and then a little later process B asks for the time, the value that B gets will be higher than (or possibly equal to) the value A got
- In a distributed system, achieving agreement on time is not trivial
  - Example: banking system (multiple withdrawal from ATM)

Physical clocks

- All computers have a circuit for keeping track of time
  - managed usually by a precisely machined quartz crystal
  - challenges when multiple CPUs (each with own clock) get involved
- Clock skew: This difference in time values between different CPUs
  - a system has *n* computers, all *n* crystals will run at slightly different rates, causing the (software) clocks gradually to get out of sync and give different values when read out
  - can lead to program failures

Physical clocks

- More important in case of real-time systems
  - external physical clocks might be needed.
  - for efficiency and redundancy, multiple physical clocks are generally considered desirable
    - how do we synchronize them with real-world clocks?
    - how do we synchronize the clocks with each other?

UTC

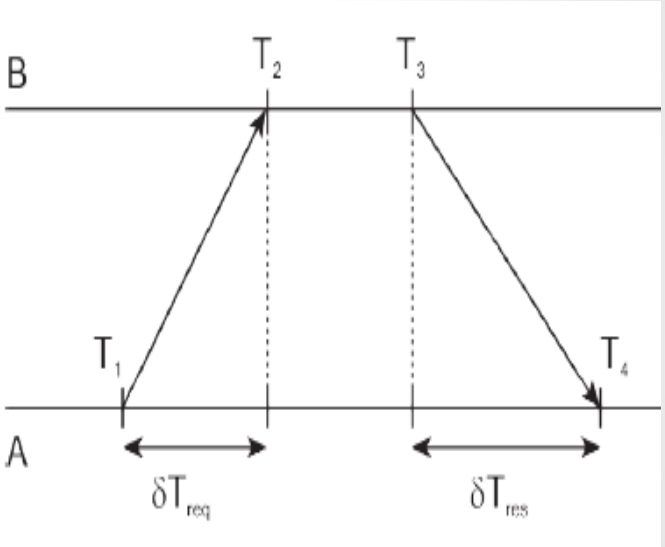
- The basis for keeping global time is a called Universal Coordinated Time (UTC)
- UTC is the basis of all modern civil timekeeping and is a worldwide standard
- For precision and time keeping:
  - some radio stations around the world broadcast a short pulse at the start of each UTC second
  - several earth satellites also offer a UTC service
- UTC receivers are commercially available, and many computers are equipped with one

Clock synchronization algorithms

- If one machine has a UTC receiver, the goal is keeping all the other machines synchronized to it
- If no machines have UTC receivers, each machine keeps track of its own time, and the goal is to keep all the machines together as well as possible
- Need for synchronization algorithms

Cristian clock synchronization

- Let clients contact a time server
- The latter can accurately provide the current time, for example, because it is equipped with a UTC receiver or an accurate clock
- The problem, of course, is that when contacting the server, message delays will have outdated the reported time



Network time protocol (NTP)

- Designed for larger scale internet
- Primary and secondary servers synchronized with each other
  - if UTC source fails primary can become secondary, etc.
- Eight pairs of values are buffered, finally taking the minimal value found as the best estimation for the delay between the two servers.
- Three methods of synchronization
  - Multicast mode
  - Procedure call mode
  - Symmetric mode

Network time protocol (NTP)

- **Multicast mode**
  - used on high-speed LANs
  - server sends time to all servers on LAN at once
  - each reset clocks (assuming a small delay)
  - not highly accurate
- **Procedure-call mode**
  - effectively Cristian's algorithm
  - server accepts requests and replies with the time
  - used when multicast not supported or higher accuracy required
- **Symmetric mode**
  - used where highest accuracy is required
  - messages exchanged, and data built up to improve accuracy of synchronization over time
  - each message sent contains timing info about the previous message received (time sent, time received) and time it is sent

How NTP works

Applying NTP symmetrically should, in principle, also let B adjust its clock to that of A. However, if B's clock is known to be more accurate, then such an adjustment would be foolish. To solve this problem, NTP divides servers into strata. A server with a **reference clock** such as a UTC receiver or an atomic clock, is known to be a **stratum-1 server** (the clock itself is said to operate at stratum 0). When A contacts B, it will adjust only its time if its own stratum level is higher than that of B. Moreover, after the synchronization, A's stratum level will become one higher than that of B. In other words, if B is a stratum-k server, then A will become a stratum-$(k+1)$ server if its original stratum level was already larger than k. Due to the symmetry of NTP, if A's stratum level was *lower* than that of B, B will adjust itself to A.

Reference: Distributed Systems, Third edition, Maarten van Steen and Andrew S. Tanenbaum

The Berkeley algorithm

- Clock synchronization algorithms mostly have a passive time server
  - machines periodically ask it for the time and server responds to their queries
- Berkeley algorithm follows the opposite approach
  - the time server (actually, a time daemon) is active, polling every machine from time to time to ask what time it is there.
  - based on the answers, it computes an average time and tells all the other machines to advance their clocks to the new time or slow their clocks down
- Suitable for a system in which no machine has a UTC receiver
- Typically, an internal clock synchronization algorithm

Clock synchronization in wireless networks

- In wireless networks, nodes are resource constrained, and multi-hop routing is expensive.
- It is often important to optimize algorithms for energy consumption
- **Reference broadcast synchronization (RBS)**
  - instead of aiming to provide all nodes UTC time, it aims at merely internally synchronizing the clocks
  - a sender broadcasts a reference message that will allow its receivers to adjust their clocks
  - the time to propagate a signal to other nodes is roughly constant, provided no multi-hop routing is assumed
  - propagation time in this case is measured from the moment that a message leaves the network interface of the sender

How RBS works?

- when a node broadcasts a reference message m, each node simply records the time T that it received m
  - T is read from their local clock
  - Ignoring clock skew, two nodes then can exchange each other's delivery times in order to estimate their mutual, relative offset
- Based on shared values each node will know the value of other node's clock relative to its own value
- Offsets are stored, there is no need to adjust its own clock, which saves energy

Logical clocks

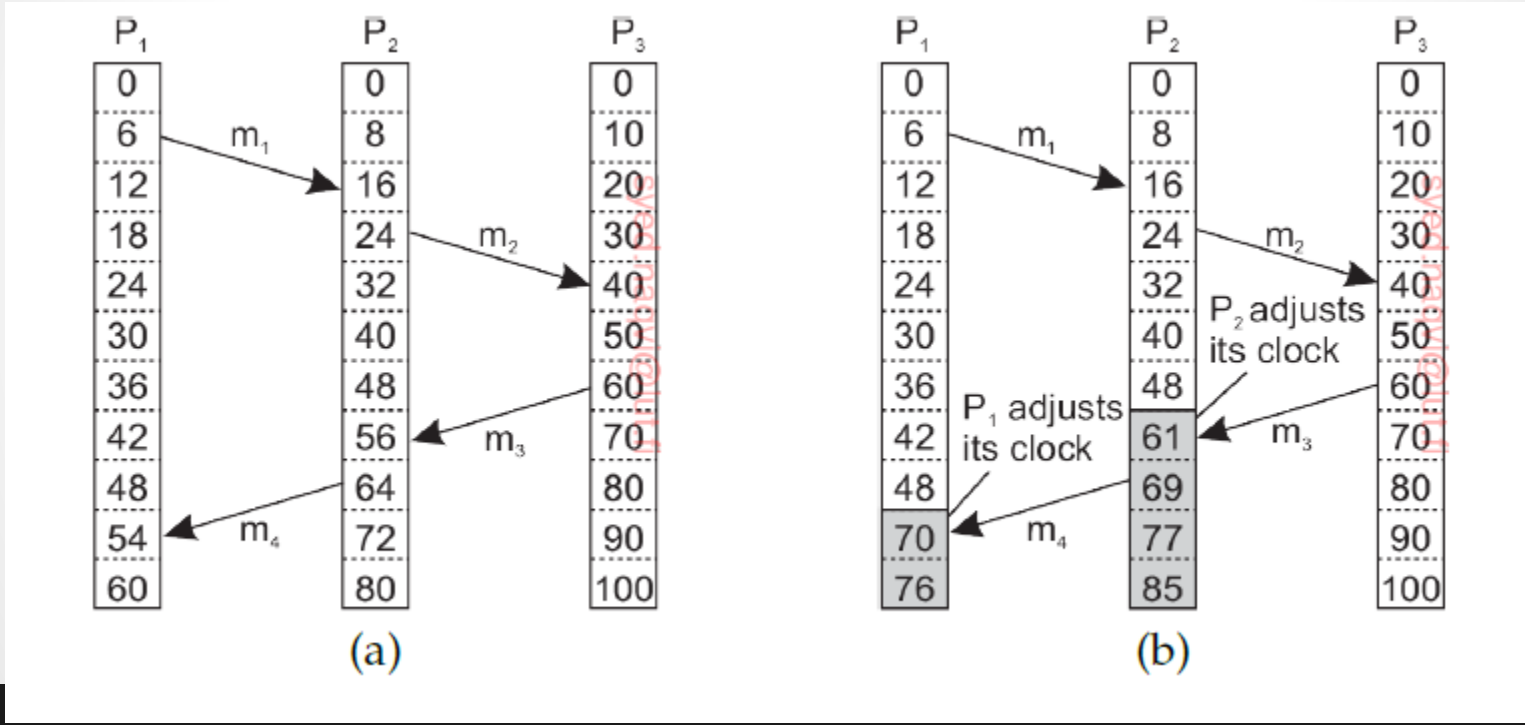
- Rather than sharing and adjusting time, computers keep track of each other's events
- A logical clock is a monotonically increasing software counter
- Each process keeps its own, L, and uses it to timestamp events
  - L++ before each event
  - Each message sent contains current L (as t)
  - Each message received sets L = max(L,t)+1

Lamport's logical clocks

- Based on a relation called happens-before
- Now if event e1 happens-before e2,

  $L(e1) < L(e2)$
- Each message carries the sending time according to the sender's clock
- When a message arrives and the receiver's clock shows a value prior to the time the message was sent, the receiver fast forwards its clock to be one more than the sending time

Lamport's logical clocks



Vector clocks

- A vector clock in a system with n processes is an array of n integers
- Each process keeps its own time
- Messages between processes contain the vector clock of the sender as a timestamp
- Each clock starts with all integers 0

Vector clocks

- Events in process i increment the i'th element in its vector clock
- When process i receives a timestamp, t, in a message it resets each element in its clock

  $V[j] = \max(V[j], t[j])$

  $j = 1 \ldots n$
- This operation is referred to as a *merge*