CISC 434: Synchronization – I

- Synchronizations – how processes in distributed systems synchronize
  - Clock Synchronization – no notion of globally shared clock
  - Logical Clocks – to reach agreement among the processes about correct ordering of events
  - Mutual Exclusion – an synchronization algorithm to limit concurrent access
  - Election Algorithms – selection of a coordinator process for synchronization
Clock Synchronization

- When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

There is no global agreement in time.

Is it possible to synchronize all the clocks in a distributed system?
UTC (Universal Coordinated Time)

- Based on constant International Atomic Time (TAI) seconds
  - but stays in phase with the motion of the sun
    - TAI is maintained in Paris by counting the transitions of the cesium 133 atom (cesium 133 clocks)
- Replaced GMT (astronomical time)
- NIST (National Institute of Standard Time) operates a short wave radio station called WWV in Colorado, which provides a short pulse at the start of every UTC second

Leap seconds introduced into UTC to get it in synch with TAI
UTC (Universal Coordinated Time)

- Time Services in Canada
  - National Research Council provides the official time for Canada – used by many agencies such as radio and television networks
  - Institute for National Measurement Standards – responsible for maintaining the preciseness in time
  - The NRC clocks are synchronized with UTC – “the reference for the official time used by all countries.”

More information:
http://www.nrc-cnrc.gc.ca/eng/services/inms/time-services.html
Clock Synchronization Algorithms

- The relation between clock time and UTC when clocks tick at different rates

- Synchronization Algorithms - Cristain’s algorithm and The Berkley algorithm
Synchronization Algorithms - Cristian’s Algorithm

- Getting the current time from a time server (B)
Cristian’s Algorithm

- The server is passive
- Time must never run backward – if the client’s (A) clock is fast, A may be slowed down gradually
- There is a nondeterministic delay in getting the server’s reply to the client – current value depends on that delay (assumed that T2-T1 = T4-T3)
Synchronization Algorithms - The Berkeley Algorithm

- The time daemon asks all the other machines for their clock values
- The machines answer
- The time daemon tells everyone how to adjust their clock

Disadvantage: Server needs active polling to ask every m/c what time is there
Logical Clocks

- It is sufficient that all the interacting machines have agreements on current time – it is not essential that this time also agrees with the real-time clock.
- Therefore, for many distributed algorithms, synchronization is achieved based on a logical clock.
Logical Clocks

- Lamport’s algorithm for synchronizing logical clocks
  - Happens before relation: If they are of the events of the same process then if \( a \) happens before \( b \) then \( a \rightarrow b \) is true
  - If \( a \) is a sending event in one process and \( b \) is the corresponding receiving event in another process then \( a \rightarrow b \) is true
  - Happens before relationship is transitive. If \( a \rightarrow b \) and \( b \rightarrow c \) then \( a \rightarrow c \)
  - If two events happen in different processes that do not communicate, nothing can be stated about their relative ordering (concurrent events)
Lamport Timestamps

a) Three processes, each with its own clock. The clocks run at different rates

b) Lamport’s algorithm corrects the clocks
Each process $P_i$ maintains a counter $C_i$. Updating of counter takes place as follows:

1. Before executing an event, $P_i$ executes $C_i \leftarrow C_i + 1$.
2. When process $P_i$ sends a message $m$ to $P_j$, it sets $m$'s timestamp $ts(m)$ equal to $C_i$ after having executed the previous step.
3. Upon the receipt of a message $m$, process $P_j$ first adjusts its own local counter as $C_j \leftarrow \max\{C_j, ts(m)\}$ and then delivers the message to the application.
Lamport’s Logical Clocks – contd.

The positioning of Lamport’s logical clocks in distributed systems.

![Diagram](image-url)
Totally-Ordered Multicasting

- Two update operations should have been performed in the same order at each copy - otherwise, they may be inconsistent.
- Solution to this problem: totally ordered multicast where a multicast operation is performed by which all messages are delivered in the same order to each receiver – Lamport’s timestamps can be used for this.
Lamport’s Clocks – Lack of Causality

- Concurrent message transmission using logical clocks.
- \( T_{snd}(mi) < T_{rcv}(mi) \)
- \( T_{rcv}(mi) < T_{snd}(mj) \)? E.g., \( T_{rcv}(m1) < T_{snd}(m2) \)?
Capturing Causality - Vector Clocks

Vector clocks are constructed by letting each process $P_i$ maintain a vector $VC_i$ with the following two properties:

1. $VC_i[i]$ is the number of events that have occurred so far at $P_i$ - local logical clock at process $P_i$.

2. If $VC_i[j] = k$ then $P_i$ knows that $k$ events have occurred at $P_j$ - $P_i$'s knowledge of the local time at $P_j$.
Vector Clocks – contd.

Steps to know the events of the other processes

1. Before executing an event, \( P_i \) executes
   \[ VC_i[i] \leftarrow VC_i[i] + 1. \]

2. When process \( P_i \) sends a message \( m \) to \( P_j \), it sets \( m \)'s (vector) timestamp \( ts(m) \) equal to \( VC_i \) after having executed the previous step.

3. Upon the receipt of a message \( m \), process \( P_j \) adjusts its own vector by setting
   \[ VC_j[k] \leftarrow \max\{VC_j[k], ts(m)[k]\} \]
   for each \( k \), after which it executes the first step and delivers the message to the application.
Enforcing Causal Communication

- Enforcing causal communication – When process $P_i$ receives a message $m$ from $P_j$, with (vector) timestamp $ts(m)$, the delivery depends on the following two conditions.
  1. $ts(m)[i] = VC_j[i] + 1$
  2. $ts(m)[i] \leq VC_j[k]$ for all $k \neq i$
Global State

- Global State - local state of each process, together with the messages in transit (sent but not received yet)
  
  a) consistent global state - if there is a receipt message in a global state, there should be a corresponding sending event (if a message has been received, it must have been sent before)
  
  b) inconsistent global state

---

(a) Consistent cut

(b) Inconsistent cut

Sender of m2 cannot be identified with this cut
Summary

- Clock Synchronization
  - Why synchronization?
  - Cristian’s algorithm for synchronization
  - The Berkley algorithm synchronization
- Logical Clocks
- Global State