Last Class: Clock Synchronization

- Physical clocks

- Clock synchronization algorithms
  - Cristian’s algorithm
  - Berkeley algorithm

Today: More Canonical Problems

- Logical clocks

- Causality
  - Vector timestamps

- Global state and termination detection
Global Positioning System

- Computing a position in a two-dimensional space.

Global Positioning System

- Real world facts that complicate GPS

1. It takes a while before data on a satellite’s position reaches the receiver.

2. The receiver’s clock is generally not in synch with that of a satellite.
Clock Synchronization in Wireless Networks

- Reference broadcast sync (RBS): receivers synchronize with one another using RB server
  - Mutual offset $= T_{i,s} - T_{j,s}$ (can average over multiple readings)

Network Time Protocol

- Widely used standard - based on Cristian’s algo
Logical Clocks

- For many problems, internal consistency of clocks is important
  - Absolute time is less important
  - Use *logical* clocks
- Key idea:
  - Clock synchronization need not be absolute
  - If two machines do not interact, no need to synchronize them
  - More importantly, processes need to agree on the *order* in which events occur rather than the *time* at which they occurred

Event Ordering

- *Problem*: define a total ordering of all events that occur in a system
- Events in a single processor machine are totally ordered
- In a distributed system:
  - No global clock, local clocks may be unsynchronized
  - Can not order events on different machines using local times
- Key idea [Lamport ]
  - Processes exchange messages
  - Message must be sent before received
  - Send/receive used to order events (and synchronize clocks)
Happened Before Relation

- If $A$ and $B$ are events in the same process and $A$ executed before $B$, then $A \rightarrow B$

- If $A$ represents sending of a message and $B$ is the receipt of this message, then $A \rightarrow B$

- Relation is transitive:
  - $A \rightarrow B$ and $B \rightarrow C \Rightarrow A \rightarrow C$

- Relation is undefined across processes that do not exchange messages
  - Partial ordering on events

Event Ordering Using $HB$

- Goal: define the notion of time of an event such that
  - If $A \rightarrow B$ then $C(A) < C(B)$
  - If $A$ and $B$ are concurrent, then $C(A) <, = \text{ or } > C(B)$

- Solution:
  - Each processor maintains a logical clock $LC_i$
  - Whenever an event occurs locally at $I$, $LC_i = LC_i + 1$
  - When $i$ sends message to $j$, piggyback $LC_i$
  - When $j$ receives message from $i$
    - If $LC_j < LC_i$ then $LC_j = LC_i + 1$ else do nothing
    - Claim: this algorithm meets the above goals
Lamport’s Logical Clocks

Example: Totally-Ordered Multicasting

\begin{itemize}
  \item Updating a replicated database and leaving it in an inconsistent state.
\end{itemize}
Causality

• Lamport’s logical clocks
  – If \( A \rightarrow B \) then \( C(A) < C(B) \)
  – Reverse is not true!!
    • Nothing can be said about events by comparing time-stamps!
    • If \( C(A) < C(B) \), then ??
• Need to maintain *causality*
  – If \( a \rightarrow b \) then \( a \) is casually related to \( b \)
  – *Causal delivery*: If \( \text{send}(m) \rightarrow \text{send}(n) \Rightarrow \text{deliver}(m) \rightarrow \text{deliver}(n) \)
  – Capture causal relationships between groups of processes
  – Need a time-stamping mechanism such that:
    • If \( T(A) < T(B) \) then \( A \) should have causally preceded \( B \)

Vector Clocks

• Each process \( i \) maintains a vector \( V_i \)
  – \( V_{i[i]} \): number of events that have occurred at \( i \)
  – \( V_{i[j]} \): number of events \( i \) knows have occurred at process \( j \)
• Update vector clocks as follows
  – Local event: increment \( V_i[I] \)
  – Send a message :piggyback entire vector \( V \)
  – Receipt of a message: \( V_{j[k]} = \max( V_{j[k]}, V_{i[k]} ) \)
    • Receiver is told about how many events the sender knows occurred at another process \( k \)
    • Also \( V_{j[i]} = V_{j[i]} + 1 \)
  • Exercise: prove that if \( V(A)<V(B) \), then \( A \) causally precedes \( B \) and the other way around.
Enforcing Causal Communication

- Figure 6-13. Enforcing causal communication.

Global State

- Global state of a distributed system
  - Local state of each process
  - Messages sent but not received (state of the queues)
- Many applications need to know the state of the system
  - Failure recovery, distributed deadlock detection
- Problem: how can you figure out the state of a distributed system?
  - Each process is independent
  - No global clock or synchronization
- Distributed snapshot: a consistent global state
### Global State (1)

**Diagram:**

- **Consistent cut:**
  - Process 1 (P1) sends a message m1 to Process 2 (P2).
  - P2 sends m2 to P3.
  - P3 sends m3 to P1.

- **Inconsistent cut:**
  - Process 1 (P1) sends a message m1 to Process 2 (P2).
  - P2 sends m2 to P3.
  - P3 sends m3 to P1.

**Notes:**

a) A consistent cut

b) An inconsistent cut

---

### Distributed Snapshot Algorithm

- **Assume each process communicates with another process using unidirectional point-to-point channels (e.g., TCP connections)**
- **Any process can initiate the algorithm**
  - Checkpoint local state
  - Send marker on every outgoing channel
- **On receiving a marker**
  - Checkpoint state if first marker and send marker on outgoing channels, save messages on all other channels until:
  - Subsequent marker on a channel: stop saving state for that channel

---

CS677: Distributed OS
Lecture 11, page 17
Distributed Snapshot

- A process finishes when
  - It receives a marker on each incoming channel and processes them all
  - State: local state plus state of all channels
  - Send state to initiator
- Any process can initiate snapshot
  - Multiple snapshots may be in progress
    - Each is separate, and each is distinguished by tagging the marker with the initiator ID (and sequence number)

Snapshot Algorithm Example

(a) Organization of a process and channels for a distributed snapshot
b) Process Q receives a marker for the first time and records its local state  
c) Q records all incoming messages  
d) Q receives a marker for its incoming channel and finishes recording the state of the incoming channel