Models of Distributed Computation and Clocks

Pradipta De
pradipta.de@sunykorea.ac.kr
Inherent Limitations of Distributed System

• Lack of global shared memory
  – No one place where the global system state can be accessed at any point

• Lack of global clock
  – Events cannot be started at the same time
  – Events cannot be ordered in time easily
Distributed Program

- Distributed Program consists of $N$ processes denoted by $\{P_1, P_2, \ldots, P_n\}$

- Each process generates events
  - Execution of an instruction or a process
  - Send a message
  - Receive a message
Why do we need global clocks?

- For causally ordering events in a distributed system
  - Example:
    - Transaction T transfers $100 from S1 to S2
    - Consider the situation when:
      - State of S1 is recorded after the deduction and state of S2 is recorded before the addition
      - State of S1 is recorded before the deduction and state of S2 is recorded after the addition
Causal Order?

• Relation among the events of an execution, or a schedule of events
  – If a occurs before b on the same process, then a is causally ordered before b (a < b)
  – If b computes m, and c delivers m, then b < c
  – If c delivers m, and d receives m, then c < d
  – Transitive property: If a < b, and b < c then a < c

• Two events a and b are concurrent if (not a < b) and (not b < a) \( \Rightarrow \) a || b
How to find the causal order among these events?
Using a Physical Clock

• Can be used to timestamp an event
• Timestamps of two events can be used to order them
• But what if they are timestamped by two different clocks?
  – Ordering can go wrong

• Need some notion of making the clocks the same
  – Synchronize then with each other
  – Synchronize them with some reference clock
Remember the limitations !!

• But perfect synchronization is impossible
• May still be ok for some applications as long as clock difference is bounded
  – The acceptable bound depends on application
• Many clock synchronization algorithms exist
  – NTP (Network Time Protocol) – standard time synchronization protocol in the Internet
• But do we really need a physical clock to order events?
Logical Clock

• Keep a local counter at each process
• Timestamp events by the counter when they occur
• What we want:
  – Given any two events in the system (from potentially different processes), their timestamps (from potentially different local counters) should tell which occurred earlier
• Critical: When and how do we update the counter?
Lamport’s Logical Clock

Happened Before relationship:

- For two events $a$ and $b$, $a \rightarrow b$ (a happened before $b$) if and only if
  - $a$ and $b$ are events in the same process and $a$ occurred before $b$
  - $a$ is a send event of a message $m$ and $b$ is the corresponding receive event at the destination process
  - $a \rightarrow c$ and $c \rightarrow b$ for some event $c$
Lamport’s Logical Clock

• $a \rightarrow b$ implies $a$ is a *potential* cause of $b$

• Causal ordering: *potential* dependencies

• Happened Before relationship causally orders events
  – If $a \rightarrow b$, then $a$ causally precedes $b$
  – If $a \rightarrow b$ and $b \rightarrow a$, then $a$ and $b$ are concurrent
    ▪ ($a \|\| b$)

• Logical clock should capture this relationship
Causally Related versus Concurrent

A space-time diagram

e11 and e21 are concurrent

e14 and e23 are concurrent

e22 causally affects e14
### Implementation Rules

- Each process $i$ keeps a clock $C_i$
- Each event $a$ in $i$ is timestamped $C(a)$, the value of $C_i$ when $a$ occurred
- $C_i$ is incremented by 1 for each event in $i$
- In addition, if $a$ is a send of message $m$ from process $i$ to $j$, then on receive of $m$,
  \[ C_j = \max(C_j + 1, C(a) + 1) \]
How Lamport’s clocks advance

We get a partial order from this space-time diagram
Total Ordering

• What if two events happen at the same time?
• Attach a process number to an event to break the tie
• Timestamps are denoted as $C_i(e).i$
• Then, $C_i(a).i$ happens before $C_j(b).j$ iff:
  1. $C_i(a) < C_j(b)$, or
  2. $C_i(a) == C_j(b)$ and $i < j$
Limitation

\[ a \rightarrow b \text{ implies } C(a) < C(b) \]

BUT

\[ C(a) < C(b) \text{ doesn’t imply } a \rightarrow b !! \]

So not a true clock!

Why does this happen?
Limitation of Logical Clock

• Clocks lose track of the timestamp of the event on which they are dependent on
  – A single integer is used to store the logical time