Distributed Systems

CS425/ECE428

Feb 5 2021

Instructor: Radhika Mittal
Something to think while we wait...

• What are the practical usecases of clocks and timestamps?

• Do we necessarily need synchronization to reason about ordering of events across processes?
Logistics Related

• VM clusters were assigned on Wednesday.

• HW1 has been released today.
  • Four questions in total (each with a few subparts).
  • You should be able to solve the first three questions by the end of today’s class.
  • You might need to wait until Wednesday’s class for the last question.
Quick Recap: Clock Synchronization

• Synchronization in synchronous systems:
  • Synchronization bound = \((max – min)/2\)

• Synchronization in asynchronous systems:
  • Cristian Algorithm: Synchronization between a client and a server.
    • Synchronization bound = \((T\text{round} / 2) – min \leq T\text{round} / 2\)
  • Berkeley Algorithm: internal synchronization between clocks.
    • A central server picks the average time and disseminates offsets.
  • Network Time Protocol: Hierarchical time synchronization over the Internet.
    • Symmetric mode synchronization between lower strata servers for greater accuracy.
NTP Symmetric Mode

- $t$ and $t'$: actual transmission times for $m$ and $m'$ (unknown)
- $o$: true offset of clock at B relative to clock at A (unknown)
- $o_i$: estimate of actual offset between the two clocks
- $d_i$: estimate of accuracy of $o_i$; $d_i = t + t'$
- skew estimate $= d_i/2$

\[
\begin{align*}
T_{Br} &= T_{As} + t + o \\
T_{Ar} &= T_{Bs} + t' - o \\
o &= \left( (T_{Br} - T_{As}) - (T_{Ar} - T_{Bs}) + (t' - t) \right) / 2 \\
o_i &= \left( (T_{Br} - T_{As}) - (T_{Ar} - T_{Bs}) \right) / 2 \\
o &= o_i + (t' - t) / 2 \\
d_i &= t + t' = (T_{Br} - T_{As}) + (T_{Ar} - T_{Bs}) \\
(o_i - d_i / 2) &\leq o \leq (o_i + d_i / 2) \text{ given } t, t' \geq 0
\end{align*}
\]
Today’s agenda

• **Logical Clocks and Timestamps**
  • Chapter 14.4

• **Global State**
  • Chapter 14.5
Today’s agenda

• Logical Clocks and Timestamps
  • Chapter 14.4

• Global State
  • Chapter 14.5
Event Ordering

• A usecase of synchronized clocks:
  • Reasoning about order of events.

• Can we reason about order of events without synchronized clocks?
Process, state, events

• Consider a system with $n$ processes: $<p_1, p_2, p_3, \ldots, p_n>$

• Each process $p_i$ is described by its state $s_i$ that gets transformed over time.
  • State includes values of all local variables, affected files, etc.

• $s_i$ gets transformed when an event occurs.

• Three types of events:
  • Local computation.
  • Sending a message.
  • Receiving a message.
Event Ordering

• Easy to order events within a single process $p_i$, based on their time of occurrence.

• How do we reason about events across processes?
  • A message must be sent before it gets received at another process.

• These two notions help define happened-before (HB) relationship denoted by $\rightarrow$.
  • $e \rightarrow e'$ means $e$ happened before $e'$. 
Happened-Before Relationship

- Happened-before (HB) relationship denoted by $\rightarrow$.
  - $e \rightarrow e'$ means $e$ happened before $e'$.
  - $e \rightarrow_i e'$ means $e$ happened before $e'$, as observed by $p_i$.

- HB rules:
  - If $\exists p_i, e \rightarrow_i e'$ then $e \rightarrow e'$.
  - For any message $m$, $\text{send}(m) \rightarrow \text{receive}(m)$
  - If $e \rightarrow e'$ and $e' \rightarrow e''$ then $e \rightarrow e''$

- Also called “causal” or “potentially causal” ordering.
Event Ordering: Example

Which event happened first?

a → b and b → c and c → d and d → f

a → b and a → c and a → d and a → f
What can we say about $e$?

$e \rightarrow f$

$a \leftrightarrow e$ and $e \leftrightarrow a$

$a \parallel e$

$a$ and $e$ are concurrent.
Event Ordering: Example

What can we say about e and d?  
\( e \parallel d \)
Logical Timestamps: Example

What can we say about \( e \) and \( d \)?

\[ e \rightarrow d \]
Lamport’s Logical Clock

• Logical timestamp for each event that captures the happened-before relationship.

• Algorithm: Each process \( p_i \)
  1. initializes local clock \( L_i = 0 \).
  2. increments \( L_i \) before timestamping each event.
  3. piggybacks \( L_i \) when sending a message.
  4. upon receiving a message with clock value \( t \)
     • sets \( L_i = \max(t, L_i) \)
     • increments \( L_i \) before timestamping the receive event (as per step 2).
Logical Timestamps: Example

Logical time

Physical time

$p_1$

$p_2$

$p_3$

$p_1$

$p_2$

$p_3$

Physical time

Logical time
Logical Timestamps: Example

Logical time

Physical time

p1

p2

p3

e 0 1 2

a b m1 (2)

3 c

4 h 5
d

m2 (5)

f 6

(2)
Lamport’s Logical Clock

• Logical timestamp for each event that captures the happened-before relationship.

• If $e \rightarrow e'$ then $L(e) < L(e')$

• What if $L(e) < L(e')$?
  • We cannot say that $e \rightarrow e'$
  • We can say: $e' \nrightarrow e$
  • Either $e \rightarrow e'$ or $e \parallel e'$
Logical Timestamps: Example

\[ L(e) < L(d), \ e \parallel d \]

\[ L(e) < L(f), \ e \rightarrow f \]
Vector Clocks

- Each event associated with a vector timestamp.
- Each process $p_i$ maintains vector of clocks $V_i$
- The size of this vector is the same as the no. of processes.
  - $V_i[j]$ is the clock for process $p_j$ as maintained by $p_i$
- Algorithm: each process $p_i$:
  1. initializes local clock $L_i = 0$.
  2. increments $L_i$ before timestamping each event.
  3. piggybacks $L_i$ when sending a message.
  4. upon receiving a message with clock value $t$
     - sets $L_i = \max(t, L_i)$
     - increments $L_i$ before timestamping the receive event (as per step 2).
Vector Clocks

• Each event associated with a vector timestamp.
• Each process \( p_i \) maintains vector of clocks \( V_i \).
• The size of this vector is the same as the no. of processes.
  • \( V_i[j] \) is the clock for process \( p_j \) as maintained by \( p_i \).
• Algorithm: each process \( p_i \):
  1. initializes local clock \( V_i[j] = 0 \)
  2. increments \( L_i \) before timestamping each event.
  3. piggybacks \( L_i \) when sending a message.
  4. upon receiving a message with clock value \( t \)
     • sets \( L_i = \max(t, L_i) \)
     • increments \( L_i \) before timestamping the receive event (as per step 2).
Vector Clocks

- Each event associated with a vector timestamp.
- Each process \( p_i \) maintains vector of clocks \( V_i \).
- The size of this vector is the same as the no. of processes.
  - \( V_i[j] \) is the clock for process \( p_j \) as maintained by \( p_i \).

Algorithm: each process \( p_i \):
1. initializes local clock \( V_i[j] = 0 \)
2. increments \( V_i[i] \) before timestamping each event.
3. piggybacks \( L_i \) when sending a message.
4. upon receiving a message with clock value \( t \)
   - sets \( L_i = \max(t, L_i) \)
   - increments \( L_i \) before timestamping the receive event (as per step 2).
Vector Clocks

• Each event associated with a vector timestamp.
• Each process $p_i$ maintains vector of clocks $V_i$.
• The size of this vector is the same as the no. of processes.
  • $V_i[j]$ is the clock for process $p_j$ as maintained by $p_i$.
• Algorithm: each process $p_i$:
  1. initializes local clock $V_i[j] = 0$
  2. increments $V_i[i]$ before timestamping each event.
  3. piggybacks $V_i$ when sending a message.
  4. upon receiving a message with clock value $t$
    • sets $L_i = \max(t, L_i)$
    • increments $L_i$ before timestamping the receive event
      (as per step 2).
Vector Clocks

• Each event associated with a vector timestamp.
• Each process $p_i$ maintains vector of clocks $V_i$.
• The size of this vector is the same as the no. of processes.
  • $V_i[j]$ is the clock for process $p_j$ as maintained by $p_i$.
• Algorithm: each process $p_i$:
  1. initializes local clock $V_i[j] = 0$
  2. increments $V_i[i]$ before timestamping each event.
  3. piggybacks $V_i$ when sending a message.
  4. upon receiving a message with vector clock value $v$
     • sets $V_i[j] = \max(V_i[j], v[j])$ for all $j=1 \ldots n$.
     • increments $V_i[i]$ before timestamping receive event
       (as per step 2).
Vector Timestamps: Example

\[ [0,0,0] \quad [1,0,0] \quad [2,0,0] \]

\[ p_1 \quad a \quad b \quad m_1 \quad ([2,0,0]) \quad c \quad [2,1,0] \quad d \quad m_2 \quad ([2,2,0]) \quad f \quad [2,2,2] \]
Vector Timestamps: Example

Physical time

\[ \begin{align*}
[0,0,0] & \quad [1,0,0] & \quad [2,0,0] \\
\text{p}_1 & \quad a & \quad b & \quad m_1 \quad ([2,0,0]) \\
\text{p}_2 & \quad \text{c} & \quad \text{d} & \quad h \quad [2,2,2] \\
\text{p}_3 & \quad e & \quad g & \quad [0,0,2] & \quad f \quad [2,3,3] \\
\end{align*} \]
Comparing Vector Timestamps

• Let \( V(e) = V \) and \( V(e') = V' \)

• \( V = V' \), iff \( V[i] = V'[i] \), for all \( i = 1, \ldots, n \)

• \( V \leq V' \), iff \( V[i] \leq V'[i] \), for all \( i = 1, \ldots, n \)

• \( V < V' \), iff \( V \leq V' \) \& \( V \neq V' \)
  \quad \text{iff} \quad V \leq V' \& \exists j \text{ such that } (V[j] < V'[j])

• \( e \rightarrow e' \) iff \( V < V' \)
  \quad \bullet (V < V' \text{ implies } e \rightarrow e') \text{ and } (e \rightarrow e' \text{ implies } V < V')

• \( e \parallel e' \) iff \( V \not< V' \) and \( V' \not< V \)
What can we say about e & f based on their vector timestamps?
Vector Timestamps: Example

\[ V(e) < V(f), \ e \rightarrow f \]
What can we say about e & d based on their vector timestamps?
Vector Timestamps: Example

\[ V(e) \not\leq V(d) \text{ and } V(d) \not\leq V(e), \ e \parallel d \]
Vector Timestamps: Example

How about now?
Vector Timestamps: Example

\[ V(e) < V(f), \ e \rightarrow f \]
\[ V(e) < V(d), \ e \rightarrow d \]
Timestamps Summary

• Comparing timestamps across events is useful.
  • Reconciling updates made to an object in a distributed datastore.
  • Rollback recovery during failures:
    1. Checkpoint state of the system; 2. Log events (with timestamps);
    3. Rollback to checkpoint and replay events in order if system crashes.

• How to compare timestamps across different processes?
  • Physical timestamp: requires clock synchronization.
    • Google’s Spanner Distributed Database uses “TrueTime”.
  • Lamport’s timestamps: cannot fully differentiate between causal and concurrent ordering of events.
    • Oracle uses “System Change Numbers” based on Lamport’s clock.
  • Vector timestamps: larger message sizes.
    • Amazon’s DynamoDB uses vector clocks.