Distributed Systems

Day 8: Time and Snapshots [Part 2]
Review of Time and Ordering

• **Time = a way to order events**
  - Total ordering: events ordered in same way by all servers
  - Partial ordering: only a subset of events are ordered in the same way by all servers
  - Causal ordering: event that ‘cause’ other events can be ordered in same way by all servers
    - Causal ordering: is an example of partial ordering

<table>
<thead>
<tr>
<th>New Post: 1</th>
<th>Modifiers to one account are causally related</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment on Post</td>
<td>Add $100</td>
</tr>
<tr>
<td>Like post</td>
<td>Add $100</td>
</tr>
<tr>
<td>Add $100</td>
<td>Withdraw $100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>New Post: 2</th>
<th>Modifications to across accounts may not be related</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment on Post</td>
<td>Add $100</td>
</tr>
<tr>
<td>Like post</td>
<td>Add $100</td>
</tr>
<tr>
<td>Add $100</td>
<td>Withdraw $100</td>
</tr>
</tbody>
</table>
Review of Time and Ordering

• Time = a way to order events
  • Total ordering: events ordered in same way by all servers
  • Causal ordering: event that ‘cause’ other events can be ordered in same way by all servers

• Total ordering: very nice but very expensive
  • Required global sequencer or global clock

• Causal ordering: easier to achieve and extremely useful
  • Vector Clocks = Counter + message exchange
Vector Clocks ----- versus ----- Logical Clocks

Initial Values

- $[1,0,0,0]$ for $P_0$
- $[0,0,0,0]$ for $P_1$
- $[0,0,0,0]$ for $P_2$
- $[0,0,0,0]$ for $P_3$

- Compare $P_0$'s $[1,0,0,0]$ with $P_1$'s $[0,0,0,0]$
- Pick max $\rightarrow [1,0,0,0]$
- Increment $P_1[1]++ \rightarrow [1,1,0,0]$
Vector Clocks

versus

Logical Clocks

Initial Values

[1,0,0,0] P0

[1,1,0,0] P1

[0,0,1,0] P2

[0,0,0,0] P3

A [1,0,0,0]

B [2,0,1,0]

C [1,1,0,0]

D [0,0,1,0]

E [1,2,2,0]

F [1,0,2,0]

G [0,0,1,1]

H [1,0,1,2]

K [3,0,1,3]

M [1,0,1,3]

P0 P1 P2 P3

• Compare P0’s [1,0,0,0] with P2’ [0,0,1,0]
• Pick max ---> [1,0,1,0]
• Increment P0[0]++ → [2,0,1,0]
Vector Clocks  ---- versus ---- Logical Clocks

A [1,0,0,0]  C [1,1,0,0]  D [0,0,1,0]  G [0,0,1,1]
B [2,0,1,0]  E [1,2,2,0]  F [1,0,2,0]  H [1,0,1,2]
K [3,0,1,3]  P0  P1  P2  P3

M [1,0,1,3]
Vector Clocks  ----- versus ----- Logical Clocks

Vector Clock Causality:
If X >= M, then M -> X
M causes X.
Use-cases for Distributed Snapshots

Node A

send ProofWork

Node B

Receive ProofWork
Use-cases for Distributed Snapshots

Node A

Send MinerHello

Receive MinerHello

Node B

How do you Debug? You need to get consistent information across all nodes
Use-cases for Distributed Snapshots

- Debug a distributed program
  - Is distributed program deadlocked?
  - Is program working correctly?
  - Want to save system state to recover from failures?

- Want to capture consistent information across all nodes.
  - Consistent snapshot
Linux Snapshotting Tool: CRIU

- Captures Process’s snapshot
  - Working memory
  - List of openfile
- Stores the snapshot to file
- Given a snapshot, you can recover the process.

https://criu.org/Checkpoint/Restore
Use-cases for Distributed Snapshots

• Debug a distributed program
  • Is distributed program deadlocked?
  • Is program working correctly?
  • Want to save system state to recover from failures?

• Want to capture consistent information across all nodes.
  • Consistent snapshot

• Distributed Snapshot
  • Capture a snapshot from each node
Definition of Consistent Distributed Snapshot

• Recall: three types of events:
  • Msg sending
  • Msg receiving
  • Local events

• Snapshot is NOT consistent
  • If it includes msg receive but not msg sending
  • In terms of VC (vector clocks)
  • A global snapshot is consistent if
    • for every event in the snapshot---the snapshot includes the events that happens before it

- Diagram showing communication between Node A and Node B:
  - Send MinerHello
  - Receive MinerHello
  - Send MineReq
  - Receive MineReq
  - Send ProofWork
  - Receive ProofWork
But...

- Distributed Snapshots are easy with total ordering
  - However, total ordering is too expensive

Take snapshot before for events < 3
Identifying Distributed Snapshots

Distributed snapshot are easy when events are totally ordered

Node A

Node B

Snapshot before 3

Snapshot before 4

 Totally ordered with global sequencer

Node A

Node B

1. Send MinerHello

2. Receive MinerHello

3. Send MineReq

4. Receive MineReq

5. send ProofWork

6. Receive ProofWork
But...

- Distributed Snapshots are easy with total ordering
  - However, total ordering is too expensive

- Alternative 1: independent Snapshots
  - Each process *independently* creates snapshots
  - Select a snapshot across all processes that is globally consistent.

Examples of ways to take independent snapshots:
- Every 5 messages
- Every 5 minutes
- After N random messages
Independent Snapshots

The black lines are snapshot locations. Given these arbitrary snapshots select snapshot across both processes that is consistent.
Independent Snapshots

The black lines are snapshot locations. Given these arbitrary snapshots select snapshot across both processes that is consistent.
Is this a Consistent Snapshot?

- \( C_{1,0}, C_{2,0}, C_{3,0} \)
- \( C_{1,1}, C_{2,1}, C_{3,2} \)
- \( C_{1,2}, C_{2,1}, C_{3,1} \)
- \( C_{3,0}, C_{2,2}, C_{1,2} \)

Includes no events!

- P1: A
- P2: B, X
- P3: Z

- P1: A, G
- P2: B, X
- P3: Z

- P1: A, G
- P2: B, X, O, F
- P3: 

Diagram shows the timeline with events and checkpoints.
Is this a Consistent Snapshot?
(Now with Vector Clocks)

Includes no events!

P1: A
P2: B, X
P3: Z, M

P1: A, G
P2: B, X
P3: Z

P1: A, G
P2: B, X, O, F
P3: Z

C1,0, C2,0, C3,0
C1,1, C2,1, C3,2
C1,2, C2,1, C3,1
C3,0, C2,2, C1,2

P1: A
P2: B, X
P3: Z

P1: A, G
P2: B, X, O, F
P3: Z
Identifying Inconsistent Snapshots With Vectors Clocks

Snapshot is inconsistent

• If there exists two processes, Pi and Px,
  • Such that Pi[x] > Px[x]

• Restated Pi knows of an event at Px
  • But Px does not know of that event

• $C_{1,1}, C_{2,1}, C_{3,2}$ is not consistent
  • P3’s last event -- M = [1,3,2]
  • P2’s last event -- X=[1,2,0]
  • So P3 knows of event 3 at P2
  • But P2 only knows of event 2
Problem with Independent Snapshots

- Worst case scenario: Domino effect
  - No snapshots may be consistently aligned
  - The only consistent global snapshots may be at the beginning of time.

- Potential Solutions
  - More frequent snapshot → very costly
  - New Protocol to address snapshots
But...

• Distributed Snapshots are easy with total ordering
  • However, total ordering is too expensive

• Alternative 1: independent Snapshots
  • Each process independently creates snapshots
  • Select a snapshot across all processes that is globally consistent.

• Alternative 2: ChandyLamport Consistent Snapshot Algorithm
  • Intuition: Coordinate snapshots by sending a snapshot message
Chandy-Lamport Intuition: 2 processes

- Send a special message indicating snapshot
ChandyLamport Intuition: 2 processes

• Send a special message indicating snapshot
ChandyLamport Intuition: 2 processes

- Send a special message indicating snapshot
ChandyLamport Intuition: 2 processes

• Send a special message indicating snapshot
Chandy-Lamport Intuition: 2 processes

- Send a special message indicating snapshot
Chandy-Lamport Intuition with 2 processes

- Send a special message indicating snapshot
ChandyLamport Intuition with 2 processes

- Send a special message indicating snapshot
Chandy-Lamport Intuition with 2 processes

- Send a special message indicating snapshot

System Assumptions
- No server (node) failure.
  - Server eventually processes message
- Network is reliable:
  - Messages are delivered eventually
  - Messages delivered in FIFO order
Snapshot Terminology

- Channel -> connection between two processes
  - Each channel is unidirectional
  - One channel from P1 to P2 and a different one from P2 to p1
  - Channel deliver messages in FIFO order
  - Channel does not lose messages
  - Channel does not fail

- Snapshot are done one at a time
  - Initiated by a global and external entity
Snapshot Terminology

- Channel -> connection between two processes
  - Each channel is unidirectional
  - One channel from P1 to P2 and a different one from P2 to P1
  - Channel deliver messages in FIFO order
  - Channel does not lose messages
  - Channel does not fail

- Snapshot are done one at a time
  - Initiated by a global and external entity
ChandyLamport Formal Algorithm

- Any node can initiate a snapshot
  - By sending a marker to all nodes
  - Start watching all channels

- If $N_X$ does not have a snapshot and receives a marker from node $N_A$
  - $N_X$ creates a snapshot
  - $N_X$ records channel $C_A$ as empty
  - Start watching all channel except $C_A$
  - Send a marker on all channels (include $C_A$)

- If $N_X$ has a snapshot and receives marker from node $N_A$
  - $N_X$ records channel $C_A$ and adds to snapshot

System Assumptions

- No server (node) failure.
  - Server eventually processes message
- Network is reliable:
  - Messages are delivered eventually
  - Messages delivered in FIFO order