EECS 491
Introduction to Distributed Systems

Fall 2019

Harsha V. Madhyastha
Replicated State Machine

- High-level strategy for fault-tolerance:
  - Replicate state
  - Apply updates in same order at all replicas

- Attempt 1:
  - Timestamp updates upon receipt
  - Order by timestamp, but clocks out of sync

- Attempt 2:
  - Establish ordering of events
  - Construct logical clock to respect ordering
1. Before executing an event at $P_i$, $C_i \leftarrow C_i + 1$
2. Send the local clock value in any message $m$
3. If $P_j$ gets message $m$, set $C_j$ to $1 + \max\{C_j, C(m)\}$

The Lamport Clock algorithm

<table>
<thead>
<tr>
<th>$P_1$</th>
<th>$C_1=2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_2$</td>
<td>$C_2=4$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$C_3=2$</td>
</tr>
</tbody>
</table>

- $C(a) = 1$
- $C(b) = 2$
- $C(m) = 2$
- $C(c) = 4$
- $C(f) = 3$
- $C(d) = 1$
- $C(e) = 2$

Physical time ↓

September 12, 2019
RSM with Lamport Clock

Diagram showing P1 and P2 with labels 2.1, 3.1, 9.1, 3.1, 10.1, 10.2, 7.2, 8.2, 9.2, and 11.2.
Causal ordering

- Ordering all updates may be unnecessary
- Example: Replication of Facebook posts
  - Causal ordering suffices
  - Show comments only if post visible to user

- With Lamport clock, $a \rightarrow b$ implies $C(a) < C(b)$
- But, converse is not necessarily true
  - $C(a) < C(b)$ does not imply $a \rightarrow b$ (possibly, $a \parallel b$)
  - Same problem with physical clock
The Lamport Clock algorithm

P1

\[ C_1 = 2 \]

\[ C(a) = 1 \]
\[ C(b) = 2 \]

P2

\[ C_2 = 3 \]

\[ C(f) = 3 \]

P3

\[ C_3 = 2 \]

\[ C(d) = 1 \]
\[ C(e) = 2 \]

C(c) = 4
Vector clock (VC)

- Label each event $e$ with a vector
  - $V(e) = [c_1, c_2, \ldots, c_n]$
  - No. of components = No. of processes

- Semantic interpretation:
  - $c_i$ is a count of events in process $i$ that causally precede $e$
Updating vector clocks

- Initially, all vectors are \([0, 0, \ldots, 0]\)
- Two rules for updating

1. For each local event on process \(i\), increment local entry \(c_i\)
2. If process \(j\) receives message with vector \([d_1, d_2, \ldots, d_n]\):
   - Set each local entry \(c_k = \max\{c_k, d_k\}\)
   - Increment local entry \(c_j\)
Vector clock: Example

- All counters start at [0, 0, 0]
- Applying local update rule
- Applying message rule
  - For each component, compute max of local clock and that in message
  - Increment component for local process
Vector clocks can establish causality

- Rule for comparing vector clocks:
  - \( V(a) = V(b) \) when \( a_k = b_k \) for all \( k \)
  - \( V(a) < V(b) \) when \( a_k \leq b_k \) for all \( k \) and \( V(a) \neq V(b) \)

- How to tell if two events are concurrent?
  - \( a || b \) if \( a_i < b_i \) and \( a_j > b_j \), some \( i, j \)

- \( V(a) < V(z) \) if and only if there is a chain of events linked by \( \rightarrow \) between \( a \) and \( z \)
Vector clock: Example

Physical time ↓

P1

P2

P3

a [1,0,0] [2,0,0] [2,1,0] [2,2,0] [2,2,2] [0,0,1]

b [2,0,0] [2,2,0]

c [2,1,0]

d [2,2,0] [2,2,0]

e [2,2,0] [2,2,2]

f [2,2,2]

September 12, 2019
EECS 491 – Lecture 4
RSMs with Logical Clocks

- Are we done implementing RSMs?

- Any replica can execute an update only after confirming clock is higher on all other replicas

- Implication:
  - If one replica down, all other replicas cannot progress
Primary Backup Replication

Client → Primary → Backup

Primary → Backup

Primary → Backup

Primary → Backup
Primary Backup Replication

- How to handle primary failure?
  - Promote one of the backups as the primary

- How to handle backup failure?
  - Add another machine as a backup
Primary Backup Sync

- When should primary sync up with backups?
- What should be transferred when syncing?
Example: MapReduce Master

```java
RegisterServer() {
    while (1) {
        receive msg and parse addr
        pick task to assign
        mark task as assigned
        respond with task assignment
    }
}
```
Example: MapReduce Master

RegisterServer() {
    while (1) {
        receive msg and parse addr
        pick task to assign
        mark task as assigned
        respond with task assignment
    }
}
Takeaways

- Cannot tolerate failure of out-of-sync primary after update to its state is externally visible

- Corollary: Okay for primary to be out of sync with backup until change is externally visible
  - External consistency

- So, when should primary sync with backups?
Primary-Backup Sync

C1

C2

P

B
Reads vs. Updates

- For operations that do not update state, primary need not consult backups
  - When is this true?

- If backup is externally consistent with primary
  - If backup takes over as primary, it will generate identical response as primary may have
What to transfer in sync?

- Snapshot of primary’s state
  - Slow
  - When is this necessary?
  - Necessary when bootstrapping new backup

- Every operation
  - Why is this okay?
  - Leverage determinism of state machine
Announcements

● Reminders:
  ◆ Attempt questions for tomorrow’s discussion
  ◆ Declare your project group by Monday
  ◆ Project 1 due in a week

● How to differentiate between network failure and worker failure?
  ◆ Cannot tell
  ◆ Both result in RPC failure

● Don’t make any assumptions about failures
What does a worker need to know to register itself?

Needs to know primary
Primary Backup Replication

- Client
- Primary
- Backup
- Backup
- Backup
Client perspective

- What does a worker need to know?
- Which machine is primary
- Can primary be hard coded into client code?
- No, primary gets replaced when it fails
- How does client discover current primary?
Primary Backup Replication

What if view service is down or unreachable?