EECS 491
Introduction to Distributed Systems

Fall 2019

Harsha V. Madhyastha
Replicated State Machine

- How to enable fault-tolerance of a service?
  - Replicate state
  - Model every replica as deterministic state machine
  - Apply updates in same order \(\rightarrow\) Same final state

- How to implement RSM?
  - Order updates based on time of receipt
  - Challenge: Syncing clocks given unbounded network delay
Let’s revisit notion of time...

- What is the utility of associating every event with a timestamp?

- What property do we expect from our clocks?

- If event a “happened before” event b, then \( \text{time}(a) < \text{time}(b) \)
Idea: *Logical clocks*

- Landmark 1978 paper by Leslie Lamport

- **Insight**: Disregard precise clock time
  - Only relationships between events matter

- Associate every event with “logical time”
  - Preserve “happens before” relationships
  - a “happens before” b represented as $a \rightarrow b$
What rules can we follow to identify which events happened before a particular event?
Defining “happens-before”

1. In same process, if $a$ occurs before $b$, then $a \rightarrow b$
2. If $c$ is a message receipt of $b$, then $b \rightarrow c$
3. If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$
Back to RSMs ...

- How do we use “happens before” to construct a logical clock?

- Use logical clock to ensure consistent ordering of updates in an RSM
Lamport clock: Objective

- Associate any event $a$ with a *clock time* $C(a)$

- **Clock property:** If $a \rightarrow b$, then $C(a) < C(b)$
The Lamport Clock algorithm

- Each process $P_i$ maintains a local clock $C_i$
- Define clock’s rules to respect “happens before”
  - If $a \rightarrow b$, then it must be the case that $C(a) < C(b)$

Physical time ↓

September 10, 2019

EECS 491 – Lecture 3
1. Before executing an event, $C_i \leftarrow C_i + 1$
The Lamport Clock algorithm

1. Before executing an event, $C_i \leftarrow C_i + 1$
The Lamport Clock algorithm

1. Before executing an event, $C_i \leftarrow C_i + 1$

2. Send the local clock in the message $m$

Physical time ↓

P1
$C_1 = 2$

P2
$C_2 = 0$

P3
$C_3 = 0$

$C(a) = 1$

$C(b) = 2$

$C(m) = 2$
3. On process $P_j$ receiving a message $m$:

- What time to set for receive event? $C(m) + 1$?

---

The Lamport Clock algorithm
The Lamport Clock algorithm

3. On process $P_j$ receiving a message $m$:

- What time to set for receive event? $C(m) + 1$?
3. On process \( P_j \) receiving a message \( m \):

- Set \( C_j \) and time of receive event to \( 1 + \max\{ C_j, C(m) \} \)
Lamport clock

- Associate any event $a$ with a clock time $C(a)$

- Clock condition: If $a \rightarrow b$, then $C(a) < C(b)$

- If $a$ causally leads to $b$, then $C(a) < C(b)$
Example use of Lamport clock

- Imagine actions on Facebook after internship
  - A: Remove manager as friend
  - B: Post that you hated your internship

- Server that serves news feed to manager has executed B, but not yet A
  - Oops!

- Lamport clock to the rescue!
  - Guarantees that clock(A) < clock(B)
  - All servers will do A then B if they follow clock order
Concurrent events

- Not all events are related by “happens before”

- a, d not related by $\rightarrow$ so concurrent, written as $a \parallel d$
Ordering all events

- How to ensure total ordering at all replicas?
- Can two events at same process have same clock value?
Ordering all events

- Break ties by appending process number to each event:

  1. Process \( P_i \) timestamps event \( e \) with \( C_i(e).i \)
  2. \( C(a).i < C(b).j \) when:

     » \( C(a) < C(b) \), or \( C(a) = C(b) \) and \( i < j \)

- Now, for two events \( a \) and \( b \), \( C(a) < C(b) \) or \( C(b) < C(a) \)
  - Total ordering of events

20 September 10, 2019 EECS 491 – Lecture 3
Announcements

● **Project 1**
  - Run non-deterministic test cases repeatedly
  - Synchronize concurrent access of shared state

● **Friday’s discussion section:**
  - Homework questions will be posted on web page

● **Reminder:** Group declaration due next Monday
Making concurrent updates consistent

- Recall multi-site database replication:
  - San Francisco (P1) deposited $100:
  - New York (P2) paid 1% interest:
- Applying updates in different order made replicas inconsistent
- All replicas apply updates in Lamport clock order
RSMs with Lamport Clocks
Consistent Ordering of Updates

- **Key idea:** Place events into a *local queue*
  - Sorted by increasing $C(x)$

- How to tell if update at head of queue can be applied?
  - Ping other nodes and check if they have earlier updates

---

![Diagram of Consistent Ordering of Updates](image-url)
RSMs with Lamport Clocks

P1

P2

3.1

8.2

2.1

7.2

3.1

8.2

9.1

9.2

$\text{P1}$

$\text{P2}$

3.1

8.2

2.1

7.2

3.1

8.2

9.1

9.2

$\text{P1}$

$\text{P2}$
Reducing Waiting

- How to reduce time between receiving an update and applying it?

- Periodically ping all other nodes to sync clock

- Tolerate temporary inconsistency
  - Apply update immediately upon receipt
  - But, maintain log of updates in order to rollback