Replication and Consistency

COS 518: Advanced Computer Systems
Lecture 3
Michael Freedman

**Time and distributed systems**

- With multiple events, what happens first?

  - A shoots B
  - A dies

  - B shoots A
  - B dies

**Correct consistency model?**

- Let’s say A and B send an op.
- All readers see A → B?
- All readers see B → A?
- Some see A → B and others B → A?

**Just use time stamps?**

- Clients ask time server for time and adjust local clock, based on response
- How to correct for the network latency?
  
  \[ \text{RTT} = \text{Time}_{\text{received}} - \text{Time}_{\text{sent}} \]
  
  \[ \text{Time}_{\text{local new}} = \text{Time}_{\text{server}} + (\text{RTT} / 2) \]
Is this sufficient?

- Server latency due to load?
  - If can measure: \( \text{Time}_{\text{local}}_{\text{new}} = \text{Time}_{\text{server}} + (\text{RTT} / 2 + \text{lag}) \)

- But what about asymmetric latency?
  - RTT / 2 not sufficient!

- What do we need to measure RTT?
  - Requires no clock drift!

- What about “almost” concurrent events?
  - Clocks have micro/milli-second precision

Order by logical events, not by wall clock time

Correct consistency model?

- Let’s say A and B send an op.
- All readers see A → B ?
- All readers see B → A ?
- Some see A → B and others B → A ?

“Lazy replication”

- Acknowledge writes immediately
- Lazily replicate elsewhere (push or pull)
- Eventual consistency: Bayou, Dynamo, …
“Eager replication”

- On a write, immediately replicate elsewhere
- Wait until write committed to sufficient # of nodes before acknowledging

Strong consistency

- Provide behavior of a single copy of object:
  - Read should return the most recent write
  - Subsequent reads should return same value, until next write

- Telephone intuition:
  1. Alice updates Facebook post
  2. Alice calls Bob on phone: “Check my Facebook post!”
  3. Bob reads Alice’s wall, sees her post

Strong Consistency?

Phone call: Ensures *happens-before* relationship, even through “out-of-band” communication
Strong Consistency?

One cool trick: Delay responding to writes/ops until properly committed

Strong Consistency? This is buggy!

- Isn’t sufficient to return value of third node: It doesn’t know precisely when op is “globally” committed
- Instead: Need to actually order read operation

Strong Consistency!

Order all operations via (1) leader, (2) consensus

Strong consistency = linearizability

- Linearizability (Herlihy and Wang 1991)
  1. All servers execute all ops in some identical sequential order
  2. Global ordering preserves each client’s own local ordering
  3. Global ordering preserves real-time guarantee
     - All ops receive global time-stamp using a sync’d clock
     - If $t_{op1}(x) < t_{op2}(y)$, $OP1(x)$ precedes $OP2(y)$ in sequence
  - Once write completes, all later reads (by wall-clock start time) should return value of that write or value of later write.
  - Once read returns particular value, all later reads should return that value or value of later write.
Intuition: Real-time ordering

- Once write completes, all later reads (by wall-clock start time) should return value of that write or value of later write.
- Once read returns particular value, all later reads should return that value or value of later write.

Sequential Consistency

In example, system orders read(A) before write(A,1)

Valid Sequential Consistency?

P1: W(x)a
P2: W(x)b
P3: R(x)b R(x)a
P4: R(x)b R(x)a

- Why? Because P3 and P4 don’t agree on order of ops. Doesn’t matter when events took place on diff machine, as long as proc’s AGREE on order.

- What if P1 did both W(x)a and W(x)b?
  - Neither valid, as (a) doesn’t preserve local ordering

Sequential Consistency

- Sequential = Linearizability – real-time ordering
  1. All servers execute all ops in some identical sequential order
  2. Global ordering preserves each client’s own local ordering

- With concurrent ops, “reordering” of ops (w.r.t. real-time ordering) acceptable, but all servers must see same order
  - e.g., linearizability cares about time
  - sequential consistency cares about program order

Weaker: Sequential consistency

- Sequential Consistency
- Write(A,1) succeeds
- committed
- 1
- read(A)

- Once write completes, all later reads (by wall-clock start time) should return value of that write or value of later write.
- Once read returns particular value, all later reads should return that value or value of later write.

Sequential Consistency

- Write(A,1) succeeds
- committed
- 1
- read(A)

- Once write completes, all later reads (by wall-clock start time) should return value of that write or value of later write.
- Once read returns particular value, all later reads should return that value or value of later write.

Sequential Consistency

- Write(A,1) succeeds
- committed
- 1
- read(A)

- Once write completes, all later reads (by wall-clock start time) should return value of that write or value of later write.
- Once read returns particular value, all later reads should return that value or value of later write.

Sequential Consistency

- Write(A,1) succeeds
- committed
- 1
- read(A)

- Once write completes, all later reads (by wall-clock start time) should return value of that write or value of later write.
- Once read returns particular value, all later reads should return that value or value of later write.
Even Weaker: Causal consistency

- Potentially *causally related* operations?
  - R(x) then W(x)
  - R(x) then W(y), x ≠ y

- **Necessary condition**: Potentially causally-related writes must be seen by all processes in the same order
  - Concurrent writes may be seen in a different order on different machines

Causal consistency

<table>
<thead>
<tr>
<th>P1: W(x)a</th>
<th>W(x)c</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2: R(x)a</td>
<td>W(x)b</td>
</tr>
<tr>
<td>P3: R(x)a</td>
<td>R(x)c</td>
</tr>
<tr>
<td>P4: R(x)a</td>
<td>R(x)b</td>
</tr>
</tbody>
</table>

- Why not sequentially consistent?
  - P3 and P4 see W(x)b and W(x)c in different order.

- But fine for causal consistency
  - Writes W(x)b and W(x)c are **not causally dependent**
    - Write after write has no dependencies

Causal consistency

<table>
<thead>
<tr>
<th>P1: W(x)a</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2: R(x)a</td>
</tr>
<tr>
<td>P3: R(x)b</td>
</tr>
<tr>
<td>P4: R(x)a</td>
</tr>
</tbody>
</table>

- Allowed with causal consistency, but not with sequential
  - W(x)b and W(x)c are **concurrent**
    - So all processes don’t see them in the same order

- P3 and P4 read the values ‘a’ and ‘b’ in order as potentially causally related. No ‘causality’ for ‘c’.

Causal consistency

<table>
<thead>
<tr>
<th>P1: W(x)a</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2: W(x)b</td>
</tr>
<tr>
<td>P3: R(x)b</td>
</tr>
<tr>
<td>P4: R(x)a</td>
</tr>
</tbody>
</table>

- A: Violation: W(x)b potentially dependent on W(x)a
- B: Correct. P2 doesn’t read value of a before W
Causal consistency

• Requires keeping track of which processes have seen which writes
  – Needs a dependency graph of which op is dependent on which other ops
  – …or use vector timestamps!
  
  See COS 418: https://www.cs.princeton.edu/courses/archive/fall17/cos418/docs/L4-time.pptx

Recall “eager replication”

• On a write, immediately replicate elsewhere
• Wait until write committed to sufficient # of nodes before acknowledging
• What does this mean?

Two phase commit protocol

1. \text{C} \rightarrow \text{P}: “request write X”
2. \text{P} \rightarrow \text{A, B}: “prepare to write X”
3. \text{A, B} \rightarrow \text{P}: “prepared” or “error”
4. \text{P} \rightarrow \text{C}: “result write X” or “failed”
5. \text{P} \rightarrow \text{A, B}: “commit write X”
**State machine replication**

- Any server is essentially a **state machine**
  - Operations **transition** between states

- Need an op to be executed on all replicas, or none at all
  - *i.e.*, we need **distributed all-or-nothing atomicity**
  - If op is deterministic, replicas will end in same state

---

**Two phase commit protocol**

1. \( C \rightarrow P: \) “request \(<op>\)”
2. \( P \rightarrow A, B: \) “prepare \(<op>\)”
3. \( A, B \rightarrow P: \) “prepared” or “error”
4. \( P \rightarrow C: \) “result exec\(<op>\)” or “failed”
5. \( P \rightarrow A, B: \) “commit \(<op>\)”

*What if primary fails? Backup fails?*

---

“Okay” (i.e., op is stable) if written to > \( \frac{1}{2} \) backups

---

- Commit sets always overlap ≥ 1 node
- Any >\( \frac{1}{2} \) nodes guaranteed to see committed op
Wednesday class

Papers: Strong consistency

Lecture: Consensus, view change protocols