Scalable Causal Consistency

COS 418: Distributed Systems
Lecture 14

Wyatt Lloyd
Consistency Hierarchy

- Linearizability
  - Sequential Consistency
    - Causal+ Consistency
      - Eventual Consistency

Examples:
- Linearizability: e.g., RAFT
- Sequential Consistency: e.g., Bayou
- Causal+ Consistency: e.g., Bayou
- Eventual Consistency: e.g., Dynamo

PRAM 1988
(Princeton)
Consistency Hierarchy

Linearizability

Sequential Consistency

Part 1: More on consistency

Causal+ Consistency

Eventual Consistency

---

e.g., RAFT

e.g., Bayou

CAP

PRAM 1988 (Princeton)

P2: Scalable Causal Consistency
Last Time’s Causal+ Consistency

• Partially orders all operations, does not totally order them
  • Does not look like a single machine

• Guarantees
  • For each process, ∃ an order of all writes + that process’s reads
  • Order respects the happens-before (⇒) ordering of operations
  • + replicas converge to the same state
    • Skip details, makes it stronger than eventual consistency
Causal Consistency

1. Writes that are potentially causally related must be seen by all processes in same order.

2. Concurrent writes may be seen in a different order on different processes.
   - Concurrent: Ops not causally related
Causal Consistency

1. Writes that are potentially causally related must be seen by all processes in same order.

2. Concurrent writes may be seen in a different order on different processes.
   - Concurrent: Ops not causally related
Causal Consistency

<table>
<thead>
<tr>
<th>Operations</th>
<th>Concurrent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td>N</td>
</tr>
<tr>
<td>b, f</td>
<td>Y</td>
</tr>
<tr>
<td>c, f</td>
<td>Y</td>
</tr>
<tr>
<td>e, f</td>
<td>Y</td>
</tr>
<tr>
<td>e, g</td>
<td>N</td>
</tr>
<tr>
<td>a, c</td>
<td>N</td>
</tr>
<tr>
<td>a, e</td>
<td>N</td>
</tr>
</tbody>
</table>

Physical time ↓

Diagram: Operations a, b, f, c, f, e, f, e, g, a, c, a, e.
Causal Consistency

<table>
<thead>
<tr>
<th>Operations</th>
<th>Concurrent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td>N</td>
</tr>
<tr>
<td>b, f</td>
<td>Y</td>
</tr>
<tr>
<td>c, f</td>
<td>Y</td>
</tr>
<tr>
<td>e, f</td>
<td>Y</td>
</tr>
<tr>
<td>e, g</td>
<td>N</td>
</tr>
<tr>
<td>a, c</td>
<td>Y</td>
</tr>
<tr>
<td>a, e</td>
<td>N</td>
</tr>
</tbody>
</table>

Physical time ↓
Causal Consistency: Quiz

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

- Valid under causal consistency

- **Why?** $W(x)b$ and $W(x)c$ are concurrent
  - So all processes don’t (need to) see them in same order

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

**Invalid under sequential consistency**

**Why?** P3 and P4 see b and c in different order

**But fine for causal consistency**
  - B and C are not causally related

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

A: Violation: $W(x)b$ happens after $W(x)a$

B: Correct. P2 doesn’t read value of $a$ before $W$
Causal consistency within replicated systems
Implications of laziness on consistency

- Linearizability / sequential: Eager replication
- Trades off low-latency for consistency
Implications of laziness on consistency

- Causal consistency: Lazy replication
- Trades off consistency for low-latency
- Maintain local ordering when replicating
- Operations may be lost if failure before replication
Consistency Hierarchy

- Linearizability
  - Sequential Consistency
    - Causal+ Consistency
      - Eventual Consistency
        - Part 1: More on consistency
          - CAP
            - PRAM 1988 (Princeton)
          - P2: Scalable Causal Consistency
            - e.g., Bayou
            - e.g., RAFT
**Consistency vs Scalability**

Scalability: Adding more machines allows more data to be stored and more operations to be handled!

<table>
<thead>
<tr>
<th>System</th>
<th>Consistency</th>
<th>Scalable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamo</td>
<td>Eventual</td>
<td>Yes</td>
</tr>
<tr>
<td>Bayou</td>
<td>Causal</td>
<td>No</td>
</tr>
<tr>
<td>Paxos/RAFT</td>
<td>Linearizable</td>
<td>No</td>
</tr>
</tbody>
</table>

It’s time to think about scalability!
## Consistency vs Scalability

**Scalability:** Adding more machines allows more data to be stored and more operations to be handled!

<table>
<thead>
<tr>
<th>System</th>
<th>Consistency</th>
<th>Scalable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamo</td>
<td>Eventual</td>
<td>Yes</td>
</tr>
<tr>
<td>Bayou</td>
<td>Causal</td>
<td>No</td>
</tr>
<tr>
<td>COPS</td>
<td>Causal</td>
<td>Yes</td>
</tr>
<tr>
<td>Paxos/RAFT</td>
<td>Linearizable</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Next Time!</td>
</tr>
</tbody>
</table>

---

Consistency vs Scalability

<table>
<thead>
<tr>
<th>System</th>
<th>Consistency</th>
<th>Scalable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamo</td>
<td>Eventual</td>
<td>Yes</td>
</tr>
<tr>
<td>Bayou</td>
<td>Causal</td>
<td>No</td>
</tr>
<tr>
<td>COPS</td>
<td>Causal</td>
<td>Yes</td>
</tr>
<tr>
<td>Paxos/RAFT</td>
<td>Linearizable</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Next Time!</td>
</tr>
</tbody>
</table>
Don't Settle for Eventual: Scalable Causal Consistency for [Geo-Replicated] Storage with COPS

W. Lloyd, M. Freedman, M. Kaminsky, D. Andersen
SOSP 2011
Geo-Replicated Storage: Serve User Requests Quickly
Trade-offs

- Consistency (Stronger)
- Partition Tolerance

VS.

- Availability
- Low Latency
- Partition Tolerance
- Scalability
Scalability through Sharding
Causality By Example

Remove boss from friends group

Post to friends: “Time for a new job!”

Friend reads post

Causality (→)
Same process
 Reads-From
 (message receipt)
Transitivity
Previous Causal Systems

• Bayou ‘94, TACT ‘00, PRACTI ‘06
  – Log-exchange based

• Log is single serialization point
  ✔ Implicitly captures & enforces causal order
  X Limits scalability
Scalability Key Idea

- Capture causality with explicit dependency metadata
  - after

- Enforce with distributed verifications
  - Delay exposing replicated writes until all dependencies are satisfied in the datacenter

![Diagram of Local Datacenter and Remote DC with numbered operations and arrows showing the flow and dependencies.](image)
COPS Architecture

All Ops Local = Available and Low Latency
Read

Client Library

read

A-F
G-L
M-R
S-Z
Write

write after = ordering metadata

Client Library

write

write + ordering metadata

write after

Replication

write after
Replicated Write

Exposing values after dep_checks return ensures causal

write_after(..., deps)

Locator Key

Unique Timestamp

deps

dep_check(A_{195})

dep check (L_{337})

A-F

G-L

M-R

S-Z

L_{337}

A_{195}
Basic Architecture Summary

• All ops local, replicate in background
  – Availability and low latency

• Shard data across many nodes
  – Scalability

• Control replication with dependencies
  – Causal consistency
Challenge: Many Dependencies

- Dependencies grow with client lifetime
Nearest Dependencies

- Transitively capture ordering constraints
Nearest Dependencies

- Transitivity capture ordering constraints
- Need extra server-side state to calculate
One-Hop Dependencies

• Small superset of nearest dependencies
• Simple to track:
  – Last write
  – Subsequent reads
One-Hop Dependencies

• Checking them suffices for causality
  – Competitive to eventually-consistent system

• Never store dependencies on the server
  
  ☐ Transitive Closure

• Simplifies client-side dep tracking
  – Clear on every write
Scalable Causal+

From fully distributed operation
Scalability

• Shard data for scalable storage

• New distributed protocol for scalably applying writes across shards

• Also need a new distributed protocol for consistently reading data across shards...
Reads Aren’t Enough
Asynchronous requests + distributed data = ??
Read-Only Transactions

• Consistent up-to-date view of data
  – Across many servers

More on transactions next time!
COPS Scaling Evaluation

More servers => More operations/sec
COPS

• Scalable causal consistency
  – Shard for scalable storage
  – Distributed protocols for coordinating writes and reads
    • Evaluation confirms scalability

• All operations handled in local datacenter
  – Availability
  – Low latency

• We’re thinking scalably now!
  – Next time: scalable strong consistency