Strong Consistency & CAP Theorem

COS 418: Distributed Systems
Lecture 15
Michael Freedman

Consistency in Paxos/Raft

- Fault-tolerance / durability: Don’t lose operations
- Consistency: Ordering between (visible) operations

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?
Paxos/RAFT has 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 read’s Alice’s wall, sees her post

Strong Consistency?

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

Strong Consistency? This is buggy!

- One cool trick: Delay responding to writes/ops until properly committed
- 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
     - As if all ops receive global time-stamp using a sync'd clock
     - If \( t_{\text{op1}}(x) < t_{\text{op2}}(y) \), \( \text{OP1}(x) \) precedes \( \text{OP2}(y) \) 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.

**Weaker: 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
Sequential Consistency

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

Valid Sequential Consistency?

- 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

"CAP" Conjecture for Distributed Systems

- From keynote lecture by Eric Brewer (2000)
  - History: Eric started Inktomi, early Internet search site based around “commodity” clusters of computers
  - Using CAP to justify “BASE” model: Basically Available, Soft-state services with Eventual consistency
- Popular interpretation: 2-out-of-3
  - Consistency (Linearizability)
  - Availability
  - Partition Tolerance: Arbitrary crash/network failures
CAP Theorem: Proof


CAP Theorem: Proof

Not consistent


CAP Theorem: Proof

Not available

CAP Theorem: Proof

Not partition tolerant


CAP Theorem: AP or CP

Criticism: It's not 2-out-of-3
• Can't "choose" no partitions
• So: AP or CP

More tradeoffs L vs. C

- Low-latency: Speak to fewer than quorum of nodes?
  - 2PC: write N, read 1
  - RAFT: write ⌊N/2⌋ + 1, read ⌊N/2⌋ + 1
  - General: |W| + |R| > N

- L and C are fundamentally at odds
  - “C” = linearizability, sequential, serializability (more later)

PACELC

- If there is a partition (P):
  - How does system tradeoff A and C?
- Else (no partition)
  - How does system tradeoff L and C?

- Is there a useful system that switches?
  - Dynamo: PA/EL
  - “ACID” dbs: PC/EC


More linearizable replication algorithms

Chain replication

- Writes to head, which orders all writes
- When write reaches tail, implicitly committed rest of chain
- Reads to tail, which orders reads w.r.t. committed writes
Chain replication for read-heavy (CRAQ)

• Goal: If all replicas have same version, read from any one
• Challenge: They need to know they have correct version

Chain replication for read-heavy (CRAQ)

• Replicas maintain multiple versions of objects while “dirty”, i.e., contain uncommitted writes
• Commitment sent “up” chain after reaches tail

Chain replication for read-heavy (CRAQ)

• Read to dirty object must check with tail for proper version
• This orders read with respect to global order, regardless of replica that handles

Performance: CR vs. CRAQ

Next Monday lecture

Causal Consistency