Distributed Systems

Day 12: Consistency

“Got to Catch Them All...”
Today

• Raft Recap

• Consensus

• CAP Theorem
Raft

• Proposed by Ongaro and Ousterhout in 2014

• Five components
  • Leader election
  • Log replication
  • Safety
  • Client protocol
  • Membership changes

• Assumes crash failures (so no byzantine failures)
• No dependency on time for safety
  • But depends on time for availability
• Tolerates (N-1)/2 failures
Raft Properties

• **Safety**: at most one leader
  - Each follower votes for at most one candidate
  - A candidate needs a majority to be leader

• **Liveness**: eventually there will be a leader
  - Challenge: if multiple try to call for election → split vote
  - Timeout+randomness: randomness helps to ensure that one server detects faster than the others

• **Log Safety**: if leader commits, then data is in all future leaders
  - Election Modifications: followers only vote for client with higher term/index
  - Commit Modifications: New Leader does not commit until entries in current term have been agreed on by followers
How do you Change Cluster Size?

Goal: add two new servers to cluster

Challenge: consistently get all servers to agree on new cluster size

Case: During cluster update, leader fails AND different servers have different notions of size → multiple leaders
How do you Change Cluster Size?

Goal: add two new servers to cluster

Challenge: consistently get all servers to agree on new cluster size

Case: During cluster update, leader fails AND different servers have different notions of size → multiple leaders

Solution: Need a protocol to consistently update the cluster
Configuration Changes

- Cannot switch directly from one configuration to another: conflicting majorities could arise

- Switching from N=3 to N=5

See the paper for details

Server 1
Server 2
Server 3
Server 4
Server 5

Majority of $C_{\text{old}}$
Majority of $C_{\text{new}}$
Today

• Raft Recap

• Consensus

• CAP Theorem
Why you should pick strong consistency, whenever possible

Mike Curtiss
Software Engineer, Cloud Spanner
January 11, 2018

Do you like complex application logic? We don’t either. One of the things we’ve learned here at Google is that application code is simpler and development schedules are shorter when developers can rely on underlying data stores to handle complex transaction processing and keeping data ordered. To quote the original Spanner paper, “we believe it is better to have application programmers deal with performance problems due to overuse of transactions as bottlenecks arise, rather than always coding around the lack of transactions.”

Try GCP
Why you should pick strong consistency, whenever possible

Eventual Consistency in Cloud Datastore

The correct API must be selected when a strongly consistent view of data is required. The different varieties of Cloud Datastore query APIs and their corresponding consistency models are shown in Table 1.

<table>
<thead>
<tr>
<th>Cloud Datastore API</th>
<th>Read of entity value</th>
<th>Read of index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Query</td>
<td>Eventual consistency</td>
<td>Eventual consistency</td>
</tr>
<tr>
<td>Keys-only Global Query</td>
<td>N/A</td>
<td>Eventual consistency</td>
</tr>
<tr>
<td>Ancestor Query</td>
<td>Strong consistency</td>
<td>Strong consistency</td>
</tr>
<tr>
<td>Lookup by key (get())</td>
<td>Strong consistency</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Approaches to Replication

Active Replication
- FIFO ordering
- Tolerates byzantine failures

Passive Replication
- Total ordering
- Protocols: Zookeeper, Paxos, Chubby

Lazy Replication
- Causal ordering
- Protocols: Gossip, DynamoDB, CassandraDB, VoldemortDB, MongoDB

Diagram:

- Active Replication (Server A, Server B, Server C)
- Passive Replication (Server A (leader), Server B (follower), Server C (follower))
- Lazy Replication (Server A, Server B, Server C)
Approaches to Replication

**Passive Replication**
- Total ordering
- Performance issues: slow and limits parallelism
  - All servers process the same request

**Lazy Replication**
- Causal ordering
- Performance: faster
  - Any server can process any request
  - More parallelism
Thinking About Consistency

• All replicas are one server

• If different clients write and read to this "one" server, what should we expect?

```
C1  Get(c)  set(c=5)
C2  Get(c)  set(c=7)
C3  Get(c)  Get(c)
C4  Get(c)  Get(c)
```
Consistency Spectrum

- **Strong Consistency**: Faster, harder to program.
- **Weak Consistency**: Slower, easier to program.

Levels:
- **Strict Serializability**: Faster, easier to program.
- **Linearizable**: Slower, harder to program.
- **Sequential**: Medium speed, easier to program.
- **Causal+**: Faster, harder to program.
- **Eventual**: fastest, hardest to program.
Linearizable

Initial c = 3

C1
- Get(c) set(c=5)

C2
- Get(c) set(c=7)

C3
- Get(c) Get(c)

C4
- Get(c) Get(c)

• Total order + FIFO + “Time”
• Recall: Total order – all servers operate in same order
• Linearizable is a subset of Total order
  • Ordering must be FIFO and based on time
Sequential

Initial c = 3

- Total order + FIFO
- Sequential is a subset of Total order
  - Ordering must be FIFO
  - Requests from different clients can be reshuffled
Causal+

Initial $c = 3$

- **Must respect Causality**
- **Need vector clocks to track and maintain causality**
- **Only causally related events need to be ordered**
- **NO TOTAL ORDERING!!!!!!**
Eventual

Initial $c = 3$

- Anything Can happen

- If no writes $\rightarrow$ eventually all servers return the same data
Consistency Spectrum

- Linearizable: total order + real time
- Sequential: total order + client order
- Causal+: causally ordered + eventually everyone agree
- Eventual: eventually everyone agrees

Strict Serializability  Sequential  Eventual

SLOWER BUT EASY TO PROGRAM

FAST and Parallel
BUT HARDER TO PROGRAM: need conflict resolution

STRONG CONSISTENCY  Linearizable  Causal+  WEAK CONSISTENCY
Today

- Raft Recap
- Consensus
- CAP Theorem
CAP Theorem

• Consistency Model: How does your system react during partition

• C: Consistency (linearizable)
  • All access is linearizable
  • Requires consensus

• A: Availability
  • All clients can make progress

• P: Partition tolerance
  • C and A can happen during a partition
**CAP Theorem**

- **Consistency Model**: How does your system react during partition
  - **C**: Consistency (linearizable)
    - All access is linearizable
    - Requires consensus
  - **A**: Availability
    - All clients can make progress
  - **P**: Partition tolerance
    - C and A can happen during a partition

- Can’t commit:
  - Not the leader
  - Cant reach leader

![Diagram]
CAP Theorem

- **Consistency Model:** How does your system react during partition

- **C:** Consistency (linearizable)
  - All access is linearizable
  - Requires consensus

- **A:** Availability
  - All clients can make progress

- **P:** Partition tolerance
  - C and A can happen during a partition

---

**I’ll commit: there WILL BE CONFLICTS**

**NO CONSISTENCY**
Raft(linearizable): Passive Replication:
• Strong consistency
• During partition ---
  • some clients will make no progress
  • Because leader is unavailable

Eventual Consistency:
• During partition ---
  • Some clients will make progress
  • Since clients can change same data
    • No consistency guarantees
• Given a “Partition”, you must pick between “Availability” and “Consistency”
  • Pick Consistently: Some clients (not all) can change “data consistently”
  • Pick Availability: All clients can change data but “inconsistently”

• C: Consistency (Linearizable)
• A: Availability
• P: Partition tolerance
Today

• Raft Recap
  • Challenges: how to change the size of the cluster

• Consensus: Consistency Models
  • Definitions of different consistency models
  • Differences between the models

• CAP Theorem: Given ‘P’, you can only have “A” and “P”.
  • When designing a system that must tolerate partitions, you must pick between “A” and “P”.