Distributed Distributed Systems

Lazy Replication and Transactions
Today!

- Lazy Replication
- CAP Theorem
- ACID Semantics
- Distributed Transactions
Overview of Lazy Replication

• Goal: give client data newer than time stamp
  • Not the most recent data just newer than FE timestamp

• Query: return value only if local timestamp is higher than client’s time stamp
  • Client must wait for condition to hold

• Update: only update data if local timestamp is higher than client's times stamp
  • Server return OK but waits internally for condition to hold

• Server May need to wait for replication before responding to an FE
What is Gossip?
What is Gossip?

- Each node selects a random subset
  - Replicates information to this subset

- Eventually data is replicated across all nodes
  - The same data may be sent to a node multiple times
  - Replication can take a while

Replication = (Frequency of Gossiping (freq. of data replication), Policy for node selection (how to select nodes to gossip to), Network Partitions)
Lazy Replication: Sending Gossip

- Gossip: sending update log to nodes
  - Randomly select gossip group
  - Gossip group == nodes to gossip with
  - For each node, \( N_i \), in gossip group
    - Gossip subset of update-log with \( \text{VC}_{\text{req}} \geq \text{VC}_i \)

Update log
- Local K/V store
- Executed log

Server A
- \( \text{VC}_A, \text{VC}_B, \text{VC}_C \)
- \( K, V, \text{VC}_K \) (request info)

Server B
- \( \text{VC}_A, \text{VC}_B, \text{VC}_C \)
- \( K, V, \text{VC}_K \) (request info)

Server C
- \( \text{VC}_A, \text{VC}_B, \text{VC}_C \)
- \( K, V, \text{VC}_K \) (request info)
Lazy Replication: Receiving Gossip

• On receive gossip message
  • Merge with ``update-log''
  • Only keep messages newer than node’s timestamp

• Process ``update-log’’
  • Order ``update-log’’ according to timestamps
  • Use requests to update K,V store
Lazy Replication: Receiving Gossip

- On receive gossip message
  - Merge with ``update-log``
  - Only keep messages newer than node’s timestamp

- Process ``update-log``
  - Order ``update-log`` according to timestamps
  - Use requests to update K,V store
Lazy Replication: All Put Together

- Gossip messages
- Update log
- Timestamp table
- Executed log
- Stable updates
- Other replica managers

FE

K, V, VC

K, V, VC

K, V, VC

ID

VC

VC

K, V

Lazy Replication: Open Challenges

• Log message can grow to an infinite size

• Messages exchanged can be quite large

• May be unable to order clocks (due to conflicts)
Lazy Replication: Sending Gossip

- Gossip: sending update log to nodes
  - Randomly select gossip group
    - Gossip group == nodes to gossip with
  - For each node, $N_i$, in gossip group
    - Gossip subset of update-log with $VC_{req} \geq VC_i$
Lazy Replication: Open Challenges

• Log message can grow to an infinite size
  • Use VC and Client-ACKs to limit size
  • Client-ACKs $\rightarrow$ limits executed log

• Messages exchanged can be quite large
  • Use VC to limit size
  • VC limits size of log in gossip messages

• May be unable to order clocks (due to conflicts)
  • Discuss in the future
Email Services

• Sending/receiving emails

• Add/remove users

• Should both use causal consistency? i.e., lazy replication?
• Services want a combination of Passive and Lazy replication
  • Some operations require passive replications (e.g., sending money)
  • Others work with lazy replication (e.g., posting)

• How do you merge passive and lazy replication?
  • Approach 1: two different systems
  • Approach 2: same system which can switch modes
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

- **Partition tolerance**: Return data during network failures
- **Consistency**: Return the most recent write else and error. Consistency == (linearizability)
- **Availability**: Return the data ASAP.

https://medium.com/system-design-blog/cap-theorem-1455ce5fc0a0
Approaches to Replication

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
Where Does Raft lie?
Where Does Raft lie?

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

No availability
Where Does Lazy Replication lie?

- Set(k, v, ts)
  - Server A
    - k4 v4 ts4
    - k5 v5 ts5
  - Server B
    - k6 v6 ts6
  - Server C
    - k4 v4 ts4
    - k5 v5 ts5

Consistency
All clients see the same data at the same time

Availability
The system continues to operate even in the presence of node failures

Partition-Tolerance
The system continues to operate in spite of network failures

FE
Server B
Server A
Server C
Where Does Lazy Replication lie?

• **Query:** return value only if local timestamp is higher than client's time stamp
  - Client must wait for condition to hold

• **Update:** only update data if local timestamp is higher than client's times stamp
  - Server return OK but waits internally for condition to hold
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