Distributed Systems

To survive failures you need a raft...
Agenda Today

- Understanding Passive Replication Usage Scenario
- Lazy replication
- CAP Theorem
Raft Replica Manager

S1 S2 S3 S4

S1 S2 S4

State for the State machine

Log Entries!!!
When should you use Raft-style Replication?

Server A

New replica server

Sequence of events

Server Replication?

Log replication

Metadata Management
When should you use Raft-style Replication?

Server Replication?

New replica server

Sequence of events

Server A

Log replication

Metadata Management
Ideal Usage scenario

- Small data with infrequent updates within the same data center
  - Small data → all data is stored in memory, so must fit in memory
  - Infrequent updates → updates require coordination and replication which is expensive
  - Same data center --> coordination-speed is a function on network latencies, server processing. Going over the internet means HUGE network latencies
Lazy Replication

VECTOR CLOCKS

HAVE A SPECIFIC SET OF SKILLS ... VERY SPECIFIC.
How do emails and forums work?

https://www.name.com/blog/fun-stuff/2012/05/office-email-chain-dog-policy/
Lazy Replication

- Supported operations
  - Get: returns value
  - Set: alters a value
    - Must get before setting

- Modifications:
  - K/V store includes ‘last modified time’

[Diagram showing interactions between Fast Edge (FE) and servers A, B, and C with timestamps and data updates]
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
Lazy Replication: Query

Set(k, v, ts)

FE → Server A

k_4 v_4 ts_4
k_5 v_5 ts_5
Lazy Replication: Update

- FE creates ID
  - ID allows for duplicate suppression

- Submits \((k, v, VC_{\text{req}}, ID)\) to server

- Server returns new time (i.e., \(VC_{\text{U}}\))
  - For future requests, FE will use \(FC_{\text{U}}\) as \(VC_{\text{FE}}\)

- Create Req-Info and store in "update-log"

Steps for creating new \(ts\):
1. Find max of index \(I\)
2. Increment \(i\) by 1

\[
\begin{align*}
VC_{\text{A}} &\left[ i', i', i', i', i'\right] \\
VC_{\text{FE}} &\left[ i, i, i, i, i\right] \\
VC_{\text{U}} &\left[ i, i, \text{max}(i, i') + 1, i, i\right]
\end{align*}
\]
Lazy Replication: All Put Together

Set(k, v, ID, VC_{req})

Steps for creating new ts:
Step 1: Find max of index i
Step 2: Increment i by 1

VC_A[ '#', '#', '#', '#', '#']

VC_{FE} [ #, #, #, #, #]

VC_U [ #, #, max(#, #') + 1, #, #]

Req_info === (K, V, ID, VC_{req}, VC_U)

Old timestamp
Request ID
New timestamp
Lazy Replication: All Put Together

- **Set** \((k,v, ID, VC_{req})\)

- **VC_U**

Steps for creating new ts:
- Step 1: Find max of index \(I\)
- Step 2: Increment \(i\) by 1

- \(VC_{C}\) \([#', '#', '#', '#', '#']\)
- \(VC_{FE}\) \([#, #, #, #, #]\)
- \(VC_U\) \([#, #, max(#, #') + 1, #, #]\)

- **VC_{K, V, VC_{K}}**
- **Local K/V store**
- **Update log**
- **Executed log**

**Req_info**

- \((K, V, ID, VC_{req}, VC_U)\)
- Old timestamp
- Request ID
- New timestamp
- Request

**Diagram:**
- FE
- Set(k,v, ID, VC_{req})
- VC_U
- Steps for creating new ts:
  - Step 1: Find max of index \(I\)
  - Step 2: Increment \(i\) by 1

**Update log**

- \(..., VC_X, .. VC_N\)
- \(K, V, VC_{K}\)
- \(K, V, VC_{K}\)
- \(K, V, VC_{K}\)
- \(req\_info\)
- \(req\_info\)
- \(req\_info\)
- \(req\_info\)

**Executed log**

- \(K, V, VC_{K}\)
- \(K, V, VC_{K}\)
- \(K, V, VC_{K}\)
- \(req\_info\)
- \(req\_info\)
- \(req\_info\)
Lazy Replication: Update

Set(k,v, ID, VC_{req})

- Compare key’s time \((VC_k)\) with req’s time \((VC_{req})\)
  - Only update value if \(VC_k \geq VC_{req}\)

- If value is updated store req-info in “executed log”

- Else keep req-info in “updated-log”
Lazy Replication: All Put Together

Set(k,v, ID, VC_{req})

VC_U

FE

K, V, VC_K
K, V, VC_K
K, V, VC_K

Local K/V store

Update log

..., VC_X, .. VC_N

req_info
req_info
req_info
req_info

Executed log

VC_K[#, #, #, #] >= VC_{req}[#, #', #', #']
Lazy Replication: All Put Together

Set(k, v, ID, VC_{req})

VC_{U}

---

VC_{K} [ #, #, #, #] >= VC_{req} [ #', #', #', #', #']
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 $VC_{req} >= VC_i$
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 $VC_{req} \geq VC_i$
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
Lazy Replication: Challenges

• Log message can grow to an infinite size

• Messages exchanged can be quite large

• May be unable to order clocks
Approaches to Replication

Active Replication

Passive Replication

Server A (leader)

Server B (follower)

Server C (follower)

Lazy Replication

Server A

Server B

Server C
• 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

- **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
Where Does Raft lie?
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