Distributed Systems

Day 10: Replication [Part 2 – the lazy replication day]
Lazy Replication
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
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

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

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

![Diagram with arrows indicating data flow and operations on servers and clients. Time is last accessed time not current time.]
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

• **Update:** only update data if local timestamp is higher than client's times stamp

• **Server** May need to wait for replication before responding to an FE
Lazy Replication: Query

Set(k,v, ts)

FE

Server A

<table>
<thead>
<tr>
<th>k_4</th>
<th>v_4</th>
<th>ts_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>k_5</td>
<td>v_5</td>
<td>ts_5</td>
</tr>
</tbody>
</table>
Lazy Replication: Update

- FE creates ID
  - ID allows for duplicate suppression

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

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

- Create Req-Info and store in ``update-log''

\[
\text{Steps for creating new ts:} \\
\text{Step 1: Find max of index } I \\
\text{Step 2: Increment } i \text{ by 1}
\]

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

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

Set(k,v, ID, VC req)

FE

VC_U

...

,..., VC_X, .., VC_N

K, V, VC_K

K, V, VC_K

K, V, VC_K

Local K/V store

Update log

req info

req info

req info

req info

Executed log

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

VC_FE[#,#,#,#,#]

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

Old timestamp

Request

Request ID

New timestamp

Req_info === (K, V, ID, VC req, VC_U)

FE
Lazy Replication: All Put Together

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

- Compare key’s time ($\text{VC}_K$) with req’s time ($\text{VC}_{\text{req}}$)
  - Only update value if $\text{VC}_K \geq \text{VC}_{\text{req}}$

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

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

FE

VC_U

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

,..., VC_X, .., VC_N

req_info
req_info
req_info
req_info

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

Local K/V store

Update log

Executed log

VC_K[#, #, #, #] \geq VC_{req}[#, #, #, #, #]
Lazy Replication: All Put Together

\[ \text{Set}(k,v,ID, VC_{req}) \]

\[ VC_K[\#,\#,\#,\#] \geq VC_{req} [\#,\#,\#,\#,\#] \]
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 $V C_{req} \geq V C_i$
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
• 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
Approaches to Replication

**Active Replication**
- FE communicates to all replicas
- Requires total ordered multicast
- FIFO ordering
- Tolerates byzantine failures

**Passive Replication**
- FE communicates with one replica (leader)
  - Leader synchs with followers before responding
- Total ordering
- Protocols: Zookeeper, Paxos, Chubby

**Lazy Replication**
- FE communicates with a random replica
  - In background, replica exchange information to other replicas.
- Causal ordering
- Protocols: Gossip, DynamoDB, CassandraDB, VoldemortDB, MongoDB