View Change Protocols and Reconfiguration

COS 418: Distributed Systems
Lecture 11
Kyle Jamieson

Today
1. More primary-backup replication
2. View changes
3. Reconfiguration

Review: primary-backup replication

- Nominate one replica primary
  - Clients send all requests to primary
  - Primary orders clients’ requests

From two to many

- Last time: Primary-Backup case study
- Today: State Machine Replication with many replicas
  - Survive more failures
**Introduction to Viewstamped Replication**

- **State Machine Replication** for any number of replicas
- **Replica group**: Group of \(2f + 1\) replicas
  - Protocol can tolerate \(f\) replica crashes

**Viewstamped Replication Assumptions:**

1. Handles **crash failures** only
   - Replicas fail only by **completely stopping**

2. **Unreliable network**: Messages might be lost, duplicated, delayed, or delivered out-of-order

**Normal operation**  
\(f = 1\)

- Client **Request**, **Prepare**, **PrepareOK**, **Reply**
- Time →

1. Primary adds request to end of its log
2. Replicas add requests to their logs in primary’s log order
3. Primary **waits for** \(f\) PrepareOKs → request is **committed**

**Replica state**

1. **configuration**: identities of all \(2f + 1\) replicas
2. In-memory **log** with clients’ requests in assigned order

\[
\begin{array}{cccc}
\text{op1, args1} & \text{op2, args2} & \text{op3, args3} & \text{op4, args4} \\
\end{array}
\]

**Normal operation: Key points**  
\(f = 1\)

- Client **Request**, **Prepare**, **PrepareOK**, **Reply**
- Time →

- Protocol guarantees **state machine replication**

- On **execute**, primary knows request in \(f + 1 = 2\) nodes’ logs
  - Even if \(f = 1\) then **crash**, \(\geq 1\) retains request in log
Where’s the commit message? (f = 1)

Client | Request | Prepare | PrepareOK | Reply
--- | --- | --- | --- | ---
A (Primary) | → | +Commit previous | → | →
B | → | | | →
C | | | | Time →

- Previous Request’s commit piggybacked on current Prepare
- No client Request after a timeout period? – Primary sends Commit message to all backups

The need for a view change

- So far: Works for f failed backup replicas
- But what if the f failures include a failed primary? – All clients’ requests go to the failed primary – System halts despite merely f failures

Today

1. More primary-backup replication
2. View changes
   - With Viewstamped Replication
   - Using a View Server
   - Failure detection
3. Reconfiguration

Views

- Let different replicas assume role of primary over time
- System moves through a sequence of views
  - View = (view number, primary id, backup id, ...)

Views

- View #1
- View #2
- View #3
View change protocol
• Backup replicas monitor primary

• If primary seems faulty (no Prepare/Commit):
  – Backups execute the view change protocol to select new primary
    • View changes execute automatically, rapidly

• Need to keep clients and replicas in sync: same local notion of who is primary
  • Same local notion at clients
  • Same local notion at replicas

Making the view change correct
• View changes happen locally at each replica

• Old primary executes requests in the old view, new primary executes requests in the new view
  – Want to ensure state machine replication

• Correctness condition: Executed requests
  1. Survive in the new view
  2. Retain the same order in the new view

Replica state (for view change)
1. configuration: sorted identities of all $2f + 1$ replicas
2. In-memory log with clients’ requests in assigned order
3. view-number: identifies primary in configuration list
4. status: normal or in a view-change

View change protocol ($f = 1$)

1. B notices A has failed, sends Start-View-Change
2. C replies Do-View-Change to new primary, with its log
3. B waits for $f$ replies (one), then sends Start-View
4. On receipt of Start-View, C replays log, accepts new ops
Old primary A must have received one or two PrepareOK replies for that request (why?)

Request is in B’s or C’s log (or both): so it will survive into new view

- Any group of \( f + 1 \) replicas is called a quorum
- Quorum intersection property: Two quorums in \( 2f + 1 \) replicas must intersect at least one replica

Applying the quorum principle

- Quorum that processes one request: \( Q_1 \)
  - ...and 2\(^{nd}\) request: \( Q_2 \)
- \( Q_1 \cap Q_2 \) has at least one replica →
  - Second request reads first request’s effects
- Quorum processes previous (committed) request: \( Q_1 \)
  - ...and that processes Start-View-Change: \( Q_2 \)
- \( Q_1 \cap Q_2 \) has at least one replica →
  - View Change contains committed request

Applying the quorum principle

Normal Operation:

- Quorum that processes one request: \( Q_1 \)
  - ...and 2\(^{nd}\) request: \( Q_2 \)
- \( Q_1 \cap Q_2 \) has at least one replica →
  - Second request reads first request’s effects
Applying the quorum principle

View Change:

- Quorum processes previous (committed) request: Q1
  - ...and that processes Start-View-Change: Q2

- Q1 ∩ Q2 has at least one replica →
  - View Change contains committed request

Split Brain (not all protocol messages shown)

<table>
<thead>
<tr>
<th>Client 1</th>
<th>Request</th>
<th>Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (Primary)</td>
<td>Execute</td>
<td>Execute</td>
</tr>
</tbody>
</table>

Network partition

<table>
<thead>
<tr>
<th>Client 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>B (New Primary)</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

- What’s undesirable about this sequence of events?
- Why won’t this ever happen? What happens instead?

Today

1. More primary-backup replication

2. View changes
   - With Viewstamped Replication
   - Using a View Server
   - Failure detection

3. Reconfiguration

Would centralization simplify design?

- A single View Server could decide who is primary
  - Clients and servers depend on view server
    - Don’t decide on their own (might not agree)

- Goal in designing the VS:
  - Only want one primary at a time for correct state machine replication
View Server protocol operation

- For now, assume VS never fails

- Each replica now periodically **pings** the VS
  - VS declares replica **dead** if missed $N$ pings in a row
  - Considers replica **alive** after a single ping received

- **Problem:** Replica can be alive but because of network connectivity, be declared “dead”

---

View Server: Split Brain

One possibility: $S_2$ in old view

Also possible: $S_2$ in new view
Split Brain and view changes

Take-away points:

• Split Brain problem can be avoided both:
  – In a decentralized design (VR)
  – With centralized control (VS)

• But protocol must be designed carefully so that replica state does not diverge

Today

1. More primary-backup replication

2. View changes
   – With Viewstamped Replication
   – Using a View Server
   – Failure detection

3. Reconfiguration

Failure detection

• Both crashes and network failures are frequent: the “common case”

• Q: How does one replica estimate whether another has crashed, or is still alive?

• A: Failure detection algorithm
  – So far, we’ve seen Viewstamped Replication e.g.:
    • Replicas listen for Prepare or Commit messages from the Primary
    • Declare primary failed when hear none for some period of time

Failure detection: Goals

• Completeness: Each failure is detected

• Accuracy: There is no mistaken detection

• Speed: Time to first detection of a failure

• Scale (if significant in system context):
  – Equal processing load on each node
  – Equal network message load
Centralized versus Gossip

Centralized
- C thinks X is dead

Gossip
- Overcomes failure

X is alive.

Today

1. More primary-backup replication
2. View changes
3. Reconfiguration

The need for reconfiguration

- What if we want to replace a faulty replica with a different machine?
  - For example, one of the backups may fail

- What if we want to change the replica group size?
  - Decommission a replica
  - Add another replica (increase f, possibly)

- Protocol that handles these possibilities is called the reconfiguration protocol

Replica state (for reconfiguration)

1. configuration: sorted identities of all 2f + 1 replicas
2. In-memory log with clients’ requests in assigned order
3. view-number: identifies primary in configuration list
4. status: normal or in a view-change
5. epoch-number: indexes configurations
Reconfiguration (1)  \( (f = 1) \)

- Primary immediately stops accepting new requests

Reconfiguration (2)  \( (f = 1) \)

- Primary immediately stops accepting new requests
- No up-call executing this request

Reconfiguration (3)  \( (f = 1) \)

- Primary sends Commit messages to old replicas
- Primary sends \textit{StartEpoch} message to new replica(s)

Reconfiguration in new group \{A, B, D\}

1. Update state with new \textit{epoch-number}
2. Fetch state from old replicas, update log
3. Send \textit{EpochStarted} msgs to replicas being removed
Reconfiguration at replaced replicas {C}

1. Respond to state transfer requests from others
2. Send StartEpoch messages to new replicas if they don’t hear EpochStarted (not shown above)

Shutting down old replicas

- If admin doesn’t wait for reconfiguration to complete, may cause > f failures in old group
- Can’t shut down replicas on receiving Reply at client
- Fix: A new type of request CheckEpoch to report the current epoch, goes thru normal request processing

Conclusion: What’s useful when

- Primary fails or has network connectivity problems?
- Majority partitioned from primary?
  → Rapidly execute view change

- Replica permanently fails or is removed?
- Replica added?
  → Administrator initiates reconfiguration protocol

Monday topic: Consensus and Paxos