View Change Protocols and Reconfiguration

COS 418: Distributed Systems
Lecture 9

Wyatt Lloyd
Housekeeping

• Midterm finally scheduled
  • 10/24, 7-9pm, Computer Science 104
  • Talk to me after if you have a conflict

• Final also scheduled
  • 1/23, 730pm, Friend Center 101

• Assignment 2 due Thursday

• Where I was last week
  • Global tables in DynamoDB!
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 Replicas

• Last time: Primary-Backup case study

• Today: State Machine Replication with many replicas
  • Survive more failures
Intro 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
Replica State

1. **configuration**: identities of all $2f + 1$ replicas

2. In-memory **log** with clients’ requests in assigned order

$$\langle \text{op1, args1} \rangle \langle \text{op2, args2} \rangle \langle \text{op3, args3} \rangle \langle \text{op4, args4} \rangle \ldots$$
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**
Normal Operation: Key Points

(f = 1)

• Protocol provides 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
Piggybacked Commits

(f = 1)

- 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

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, ...)

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 state of the current view
  • Same current view at replicas
  • Same current view at clients
Correctly Changing Views

- 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

- So 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
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, then sends **Start-View**

4. On receipt of Start-View, C replays log, accepts new ops
**View Change Protocol: Correctness**

(f = 1)

- 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
Principle: Quorums

(f = 1)

- Any group of f + 1 replicas is called a quorum
- Quorum intersection property: Two quorums in 2f + 1 replicas must intersect in at least one replica.
Applying the Quorum Principle

Normal Operation:

• Quorum that processes one request: $Q_1$
  • ...and 2$^{\text{nd}}$ 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

• $Q_1 \cap Q_2$ has at least one replica
  • View Change contains committed request
• 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

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 View Server:
  • Only **one primary** at a time for correct state machine replication
For now, assume View Server never fails

Each replica periodically pings the View Server
  • VS declares replica dead if missed N pings in a row
  • VS 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

View Server

(1, S₁, S₂)
(2, S₂, −)
One Possibility: $S_2$ in Old View

Client

View Server

$(1, S_1, S_2)$

$(2, S_2, -)$

$(1, S_1, S_2)$

$(2, S_2, -)$

$(2, S_2, -)$
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 (Viewstamped Replication)
  • With centralized control (View Server)

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

• 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)

- Primary immediately stops accepting new requests
- No up-call to RSM for **executing** this request
Reconfiguration (3)

(f = 1)

- Primary sends Commit messages to old replicas
- Primary sends **StartEpoch** message to new replica(s)

Reconfiguration

Client

new-config

A (Primary)

B

C (remove)

D (add)

Prepare, PrepareOK

Reply

Commit

StartEpoch

Time →

• Primary sends Commit messages to old replicas

• Primary sends **StartEpoch** message to new replica(s)
1. Update state with new epoch-number
2. Fetch state from old replicas, update log
3. Send *EpochStarted* msgs to replicas being removed
1. Respond to state transfer requests from others
   • Waits until it receives $f' + 1$ EpochStarteds, $f'$ is fault tolerance of new epoch

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 and decommissions old nodes, may cause $> f$ failures in old group
  • Can’t shut down replicas on receiving Reply at client

• Must ensure committed requests survive reconfiguration!

• Fix: A new type of request CheckEpoch reports the current epoch
  • Goes thru normal request processing (again no upcall)
  • Return indicates reconfiguration is complete
  • Q: Why not have reconfigure wait for this to complete?
Conclusion: What’s Useful When

• Backups fail or has network connectivity problems?
• Minority partitioned from primary?
  → Quorums allow primary to continue

• 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