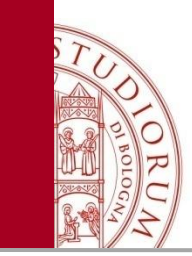
Distributed Systems 2023/24 Project:

Hash chain replication with RAFT Consensus

By

Mert Akpinar &

Yago Illing Arizti

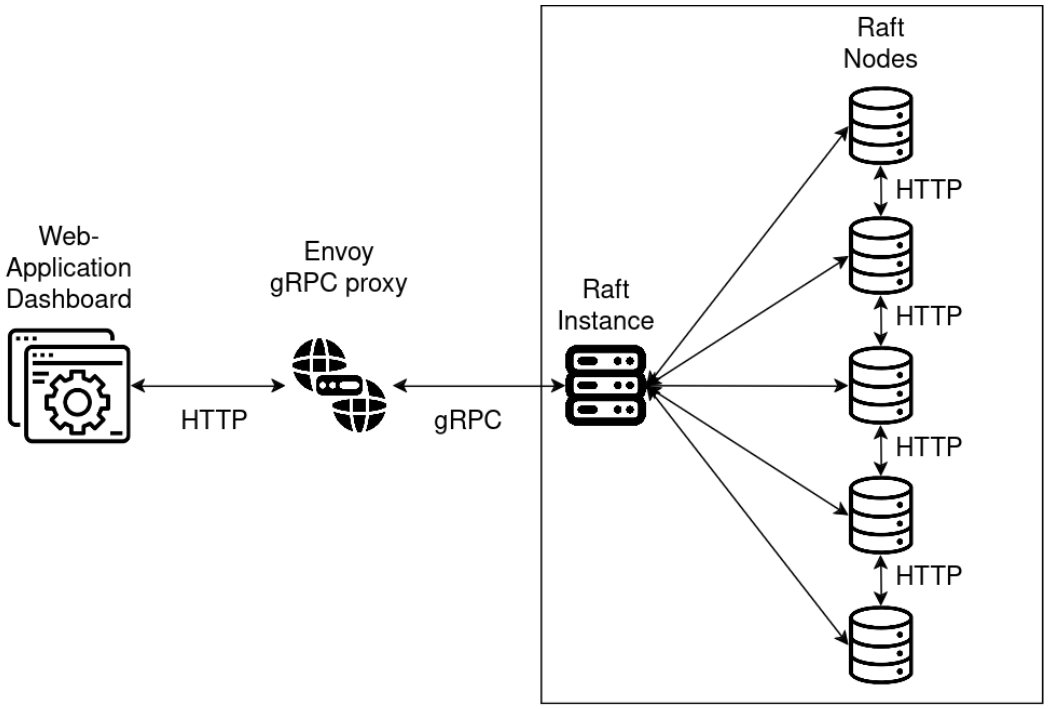


Project goal

- Develop a distributed application which consists of replicated state machines (nodes)
- Each node manages its own hash chain
- Use the RAFT consensus algorithm to guarantee same state of the hash chain across nodes
- Develop a web application to monitor/manage state of the node cluster and to append data (text) to the hash chain

Project structure

- Components:
  1. Raft Instance (Kotlin - Ktor)
  2. gRPC proxy (envoy)
  3. Web-Application Dashboard (React.js)

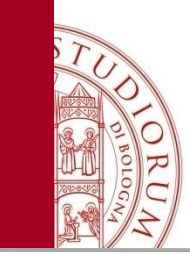


Project structure diagram

RAFT consensus

Raft terminology:

- **Term**: time frame of arbitrary length which starts with a new election
- **Log**: list of data entries received from client/leader which are going to be applied to the hash chain
- **Commit index**: index of the log, which is replicated on the majority of nodes and can be applied to the hash chain



RAFT consensus

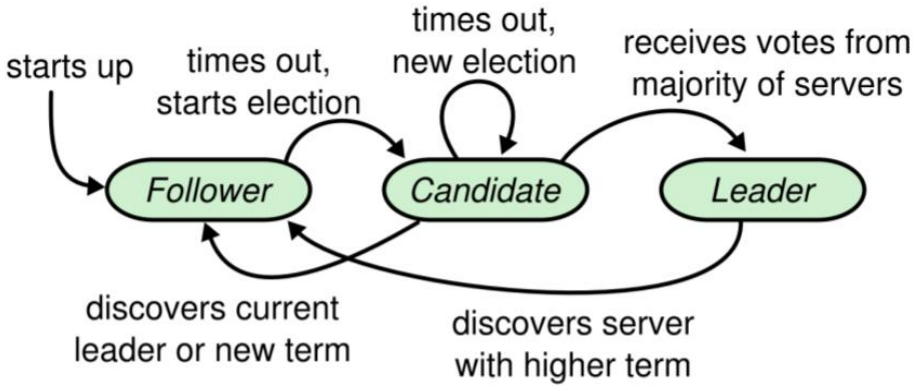
Raft states:

- **Leader**: node can receive entries from client and sends them out periodically to
- **Candidate**: node is requesting to be the leader for the current term
- **Follower**: node is requesting to be the leader for the current term

RAFT consensus

Raft state changes:

- Follower starts new election when connection to leader times out
- Leader is elected by majority vote from other nodes (guarantees one leader per term)
- Leader/Candidate steps down when higher term is detected



Raft State changes

RAFT consensus

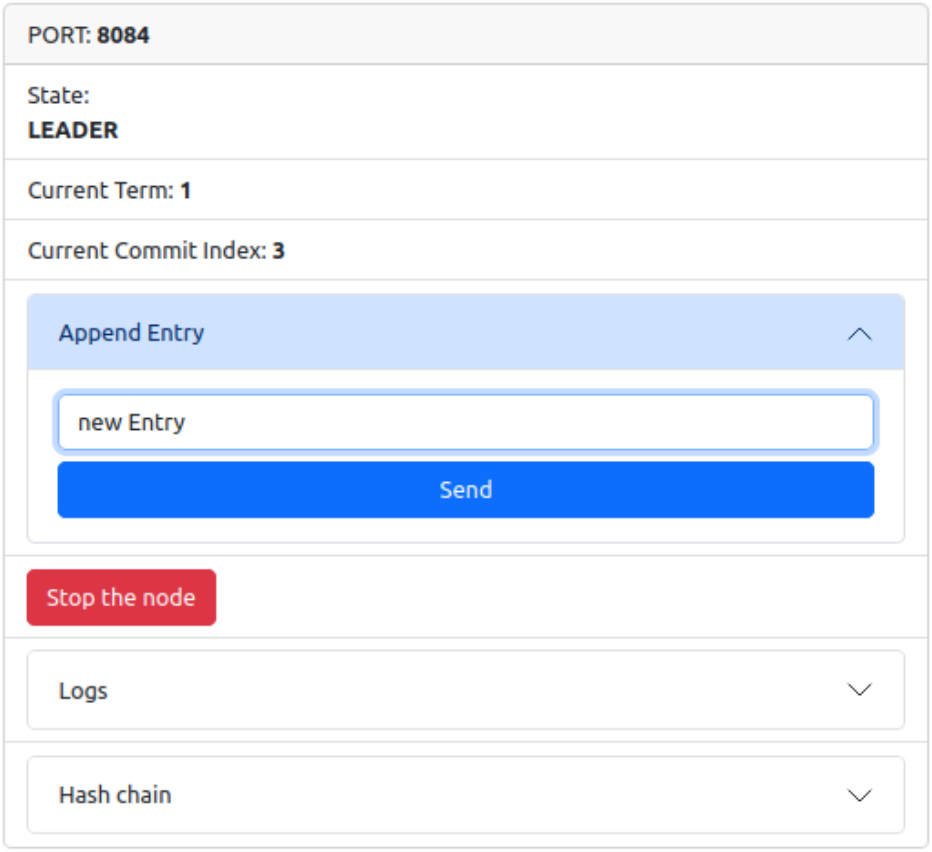
Log replication:

- Leader periodically sends out requests to append new log entries to followers (heartbeat)

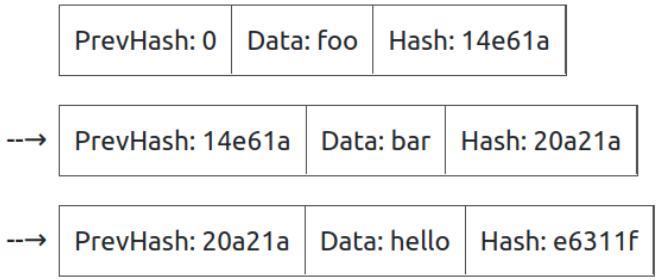
- Follower appends (or overwrites existing log entry) received entry under these conditions:
  1. Term of leader matches with term of follower
  2. Term and index of the previous log entry to the one sent by the leader matches with the follower's log

- If the follower rejects the new entry, then the leader decreases the index of the entry to be sent until the operation is a success

Web-application dashboard



Status widget for Leader node



Hash chain visualization

Demo

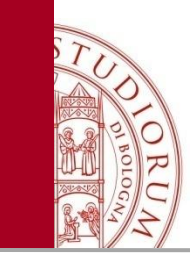
Scenario 1 "Log entry replication":

Action:

1. String data is sent to the Leader node

Expected Behavior:

1. Leader node creates new raft log entry
2. Leader node broadcasts new entry
3. Follower nodes append new log entry to log and return success
4. Leader node increases commit index and creates hash chain
5. Follower nodes receive increased commit index and create hash chain



Demo

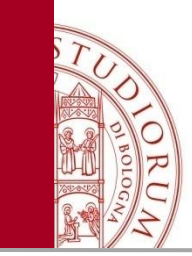
Scenario 2 "New leader election and log catch up":

Action:

1. Stop Leader Node
2. Append Log Entry to new Leader
3. Restart old Leader Node

Expected Behavior:

1. New leader is elected
2. Old leader node becomes a follower node when discovering the new term
3. Old leader node catches up to current state of log



Demo

Scenario 3 "Invalid log entry overwrite":

Action:

1. Add entries to Leader node and stop it before entries are broadcasted
2. Add entries to new Leader node

Expected Behavior:

1. Uncommitted entries of old leader node are overwritten by new leader by going back in the log

Thank you for your attention!