Data Processing - URL shortener project

1. Build and run

Requires Go to be installed

```
chmod u+x startRaftCluster.sh
./startRaftCluster.sh
```

2. Describe the design of your system

The system consists of the following main components:

- **Raft library** (`/pkg/raft`; closely following the Raft paper)
- **URL shortener service** (`/internal/urlShortener`)
  - **HTTP server** (`/internal/urlShortener/server.go`; handles client requests)
  - **Finite-state machine** (`/internal/urlShortener/store.go`; listens to Raft commits, applies them to the state machine, and responds to read-only queries)
- **Main executable** (`/cmd/joinNode/main.go`; starts a new Raft node and the URL shortener service)

The URL shortener service can simply be swapped out with another service that implements the `ListenToNewCommits(commitChan chan raft.Commit)` method. This method indefinitely listens to new commits from the Raft library and applies them to the state machine. The state machine is responsible for responding to read-only queries by the server.

If the server receives a write request, it forwards it to the Raft node via the `raftNode.Submit(req ClientRequest)` method. This method appends a new entry with the FSM command to the log if the method was called on the leader node. If it was not called on the leader node, the client receives an error and can call the `raftNode.GetLeaderId()` method to query the current leader. Additionally, the `Submit` method not only appends a new log entry but also adds the `ClientRequest` to a map with the log index as key. That way, the Raft node is able to notify the client after it has committed and applied the request.

3. How does your cluster handle leader crashes?

The Raft library handles leader crashes by electing a new leader. The leader election is triggered by some node whose election timer times out. The node then starts a new election and sends out `RequestVote` RPCs to all other nodes. If a node receives a `RequestVote` RPC, it checks if the candidate is eligible to become the new leader. If it is, it votes for the candidate and sends a `RequestVoteResponse` back. If the candidate receives a majority of votes, it becomes the new leader. If not, the election is aborted and all nodes wait for their election timer to time out to start the process again.

- How long does it take to elect a new leader?

That depends on the election timeout and the time it takes for network round trips. If the election timeout is set to 150ms and the network round trip time is 50ms, it takes at most 200ms to elect a new leader.

- Measure the impact of election timeouts. Investigate what happens when it gets too short / too long.

If there is not enough randomization in the election timeouts between nodes, then it might happen that elections are triggered at the same time on multiple nodes. That means that multiple nodes become a candidate (the so-called split vote problem). In that case, the election is aborted and the nodes wait for their election timer to time out to start the process again.

4. Analyze the load of your nodes:

- How many resources do your nodes use?

Depends on the number of client requests. The number of RPC calls stays constant, though the network traffic depends on the number of new log entries and increases linearly with the number of client requests.

- Where do inserts create most load?

On the leader node.

- Do lookups distribute the load evenly among replicas?

That is the task of the client.

5. How many nodes should you use for this system? What are their roles?

At least 3 nodes are required to establish a quorum.

6. Measure the latency to generate a new short URL

On average, a write request takes `E[w] = heartbeatInterval/2 + networkRoundtripQuorum + networkRoundtrip` to finish (neglecting the processing time) since the client must wait for the leader to commit and apply the FSM command. The log entries are only sent with every heartbeat, so the client must wait for the next heartbeat to receive a response.

On the other hand, a read request takes only a few milliseconds `E[r] = processingTime + networkRoundtrip` since the client can directly query the state machine.

- Analyze where your system spends time during this operation

Waiting for the leader to commit and apply the FSM command.

7. Measure the lookup latency to get the URL from a short id

Of course, this depends on the machine the server runs on. But on my machine, it takes about 2-5ms to resolve a short URL with Postman (with “automatically follow redirects” being disabled).

8. How does your system scale?

- Measure the latency with increased data inserted, e.g., in 10% increments of inserted short URLs

It does not change much - as expected. With higher load the response time increases but just slightly compared to the expected write latency. When we increase the rps to 1000 we see more outliers and noise in the chart. This is due to the fact that the leader has to process many clients at the same time now and it might happen that one client can not be processed until the end of the current heartbeat cycle and it has to wait for the next cycle. See presentation slides and the benchmark folder for real numbers.

During a write-only benchmark test on the leader, the replica nodes have almost zero load. This is as expected since they only get a single request from the leader each heartbeat interval and must only append the entries to their log.

- Measure the system performance with more nodes

No significant changes between 3, 5, 11 nodes.