EXPENSES TRACKER

Introduction

Safe Log Replication with leader election

Due to the lack of client requests, the lastLogIndex and lastLogTerm comparisons were trivial and not considered in phase 3. During this phase, we will see how these comparisons play a crucial role in determining who becomes a leader and who cannot.

Prerequisites of what each server should store

Persistent state on all servers:

currentTerm: Latest term server has seen

votedFor: candidateId that recevied vote in current term

log[]: log entries

Volatile state on all servers:

commitIndex: Index of highest log entry known to be committed.

Initialized to 0, increases monotonically.

lastApplied: Index of highest log entry applied to state machine.

Initialized to 0, increases monotonically.

Volatile state on leaders:

nextIndex[]: For each server, index of the next log entry to send to that server.

matchIndex[]: For each server, the index of the highest log entry known to be replicated on server.

Handling additional requests that will be sent out by the controller. STORE & RETRIEVE

STORE

We will be using the STORE cmd to make client requests.

RETRIEVE

The controller can send a request to any of the nodes to RETRIEVE all the committed entries at that node. However, only the Leader will respond with the committed entries.

entry = {

“Term”: Term in which the entry was received (may or may not be the current term)

“Key”: Key of the message (Could be some dummy value)

“Value”: Actual message (Could be some dummy value)

}

Design Overview

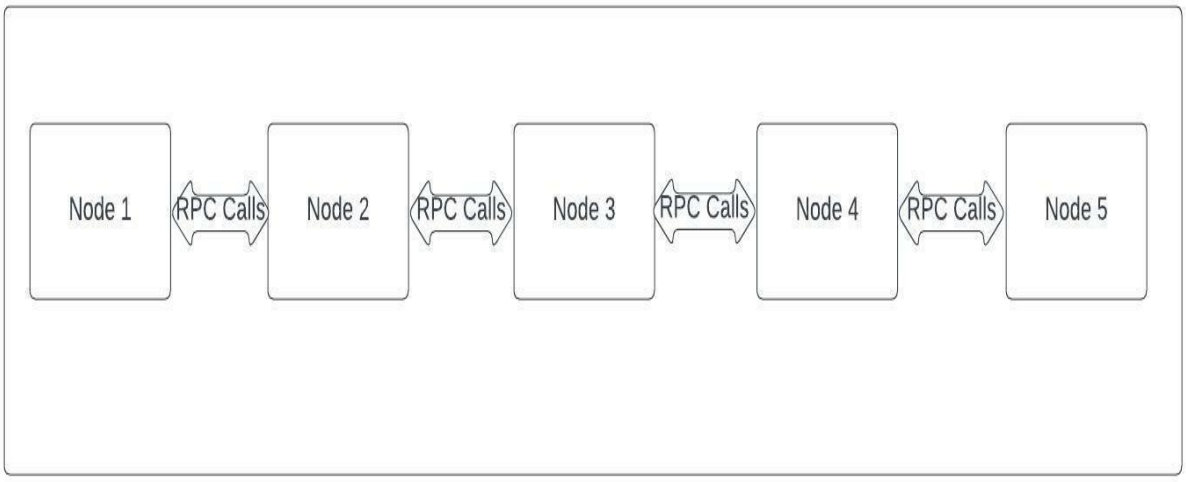


Fig: Architecture Diagram

Implementation

Handling additional requests that will be sent out by the controller. STORE & RETRIEVE

We will be using the STORE cmd to make client requests.

STORE CMD

{

“sender_name”:Controller,

“term”:null,

“request”:STORE,

“key”:K4,

“value”:Value4

}

Response :

From Non Leader Node :

{

“sender_name”: Node2,

“term”:null,

“request”:LEADER_INFO,

“key”: LEADER,

“value”:Node4

}

RETRIEVE

The controller can send a request to any of the nodes to RETRIEVE all the committed entries at that node. However, only the Leader will respond with the committed entries.

entry = {

“Term”: Term in which the entry was received (may or may not be the current term)

“Key”: Key of the message (Could be some dummy value)

“Value”: Actual message (Could be some dummy value)

}

Response :

From Non-Leader Node:

{

“sender_name”: Node2,

“term”:null,

“request”:LEADER_INFO,

“key”: LEADER,

“value”:Node4

}

Safe Log Replication

Any new entries will be appended in APPEND ENTRY RPC as part of the heartbeat.

The AppendEntry RPC is triggered at regular intervals as a heartbeat (which is why it is also functioning as a heartbeat). The STORE req appends an entry to the leader’s log and this new entry gets sent along to the followers in the subsequent heartbeat/AppendEntryRPC.

If a particular entry has been replicated on most of the followers, each node will apply that to their own logs for a final commit.

Append Entry Reply:

When a follower receives an AppendEntry RPC:

1. Reply false if term<currentTerm
2. Reply false if log doesn’t contain an entry at prevLogIndex whose term matches prevLogTerm
3. If both the above conditions are not met, the the follower accepts the logs, appends any new entries to its own logs and sends an Append_reply with success=True
4. update the follower node’s commitIndex based on the value sent by the leader and the length of the follower’s logs

{

“term”:1,

“matchIndex”:1,

“success”:”true/false”

}

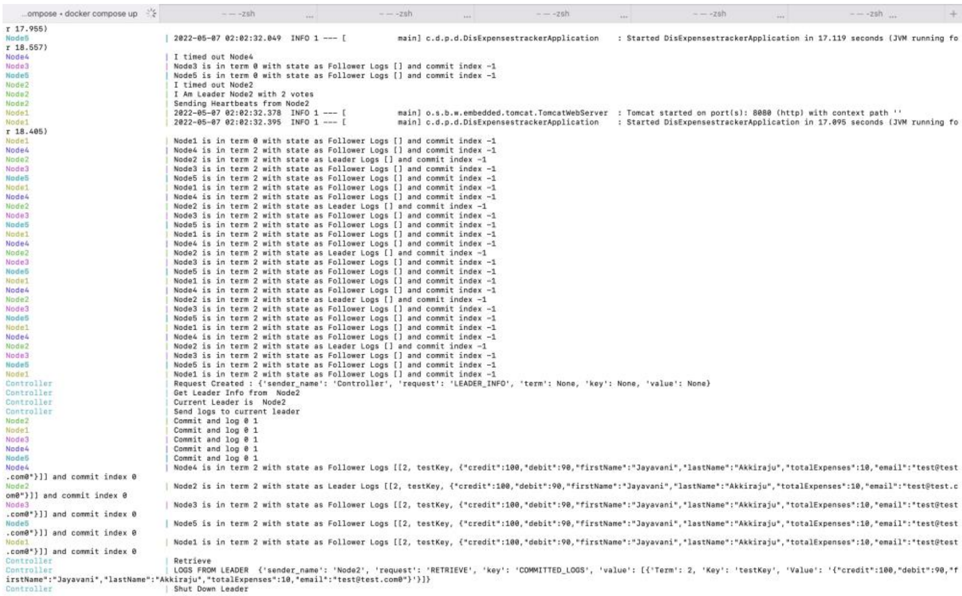
If the leader node receives false append entry reply from any of the follower nodes, the leader node decrements the nextIndex for the corresponding follower node until the leader’s and follower’s log indices match.

Client Integration:

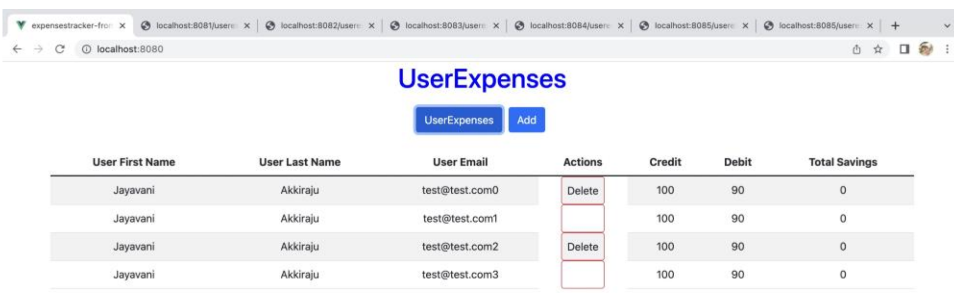
A request from a client is sent to any of the nodes in the system. If the receiving node is the leader node, the request is appended on the logs itself and sent as an entry in the next append entry RPC.

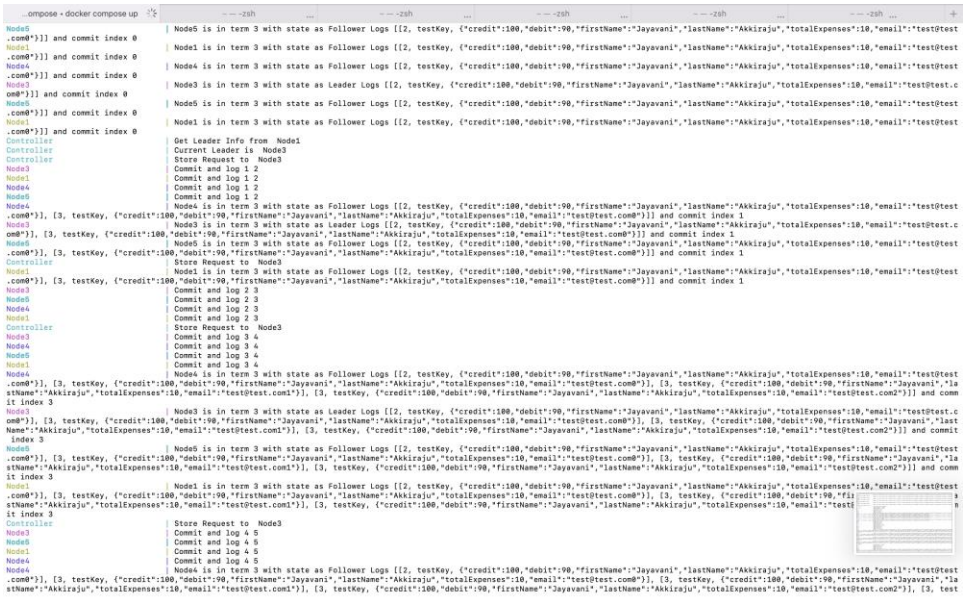
On receiving the corresponding entry on follower nodes, the entry is appended to the logs of the follower nodes. If the entry is replicated on most of the nodes, the leader commits the entry on its state machine. The same entry is later committed on the state machines of the follower nodes.

Validation



The requests from the client have been appended successfully to the leader as well as all other nodes.





References

https://raft.github.io/raft.pdf

http://thesecretlivesofdata.com/raft

https://www.youtube.com/watch?v=YbZ3zDzDnrw

https://docs.python.org/3/library/socket.html

https://docs.python.org/3/library/threading.html