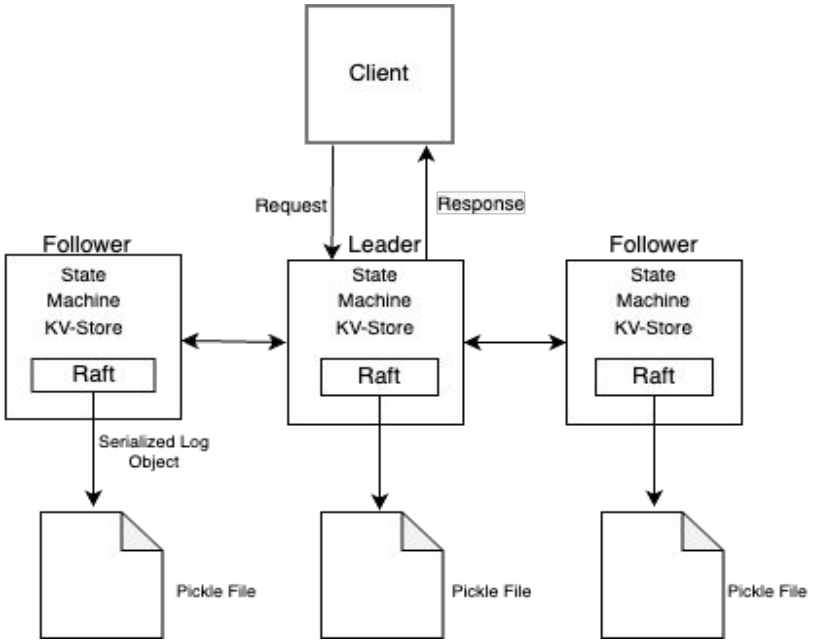
# Replicated Database with Raft Protocol

Mia Weaver
Ruthvika Reddy Loka
Mahitha Pillodi

# Supported APIs

- **GET** - The GET operation fetches the current value for the given key. It returns an error message in case of a non-existent key.
- **PUT** - The PUT operation re-writes the value for the given key. In case of a non-existent key, the PUT operation simply adds the key and value pair to the in-memory database. Only the leader can service PUT operations.

# Overall Design



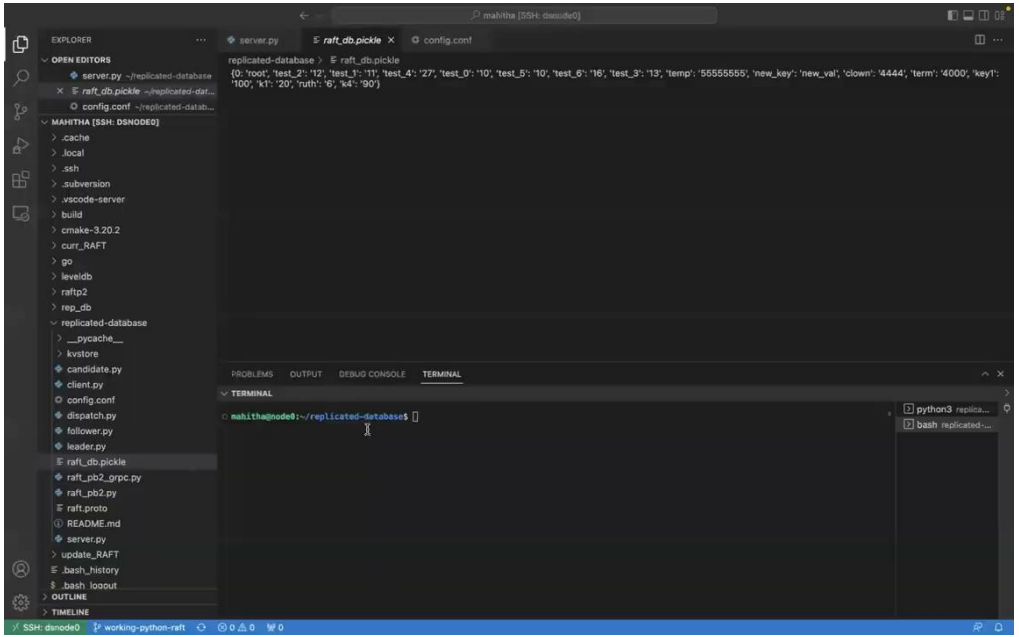
# Raft Functionality

- **Consensus** - Raft for establishing consensus, through leader election and log replication among the servers.
- **Replication** - Client operations are logged as a LogEntry and replicated on all servers through the AppendEntries operation.
- **Durability** - The Log Entry (containing the Operation information) is stored durably as a byte array in a file on disk. However, we persisted the data in the Pickle file.
- **Crash Recovery** - The replicated log is used to populate the server state machine after it comes up post crash

# Demos

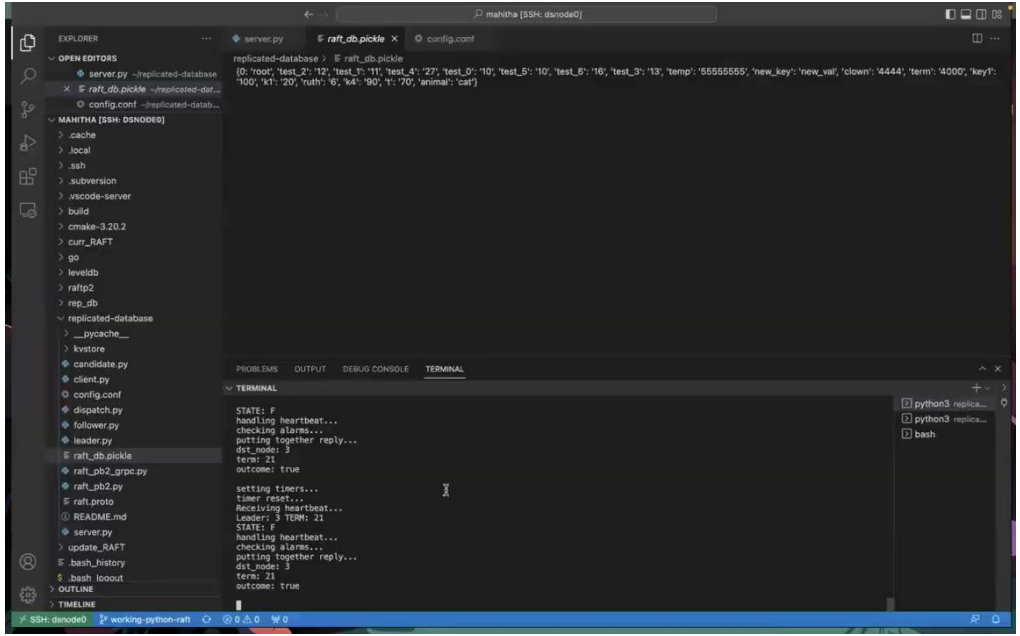
## Single Leader Election and Normal Operations Testing

- Tested single leader election when all nodes are running by spawning all servers that communicate with each other and printing the status of each server.
- Tested expected response of GET and PUT operations.



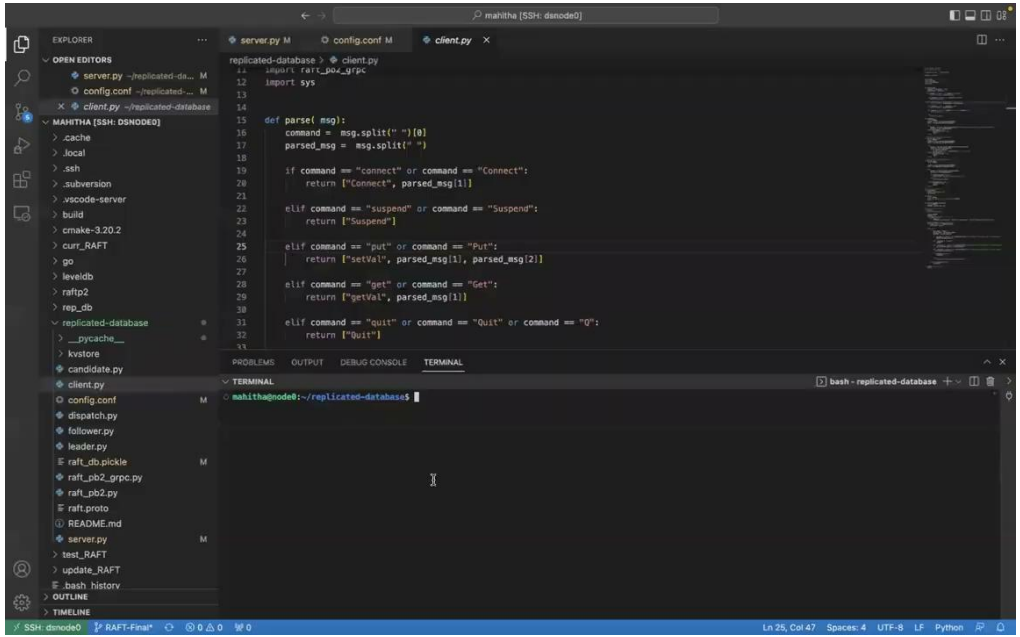
## New Leader Election upon Current Leader Crash and Quorum Testing

- Tested the election of a new leader when the current leader crashes and the majority of nodes are still alive.
- Tested by killing the current leader in a five node cluster and checking if one of the other nodes becomes a leader by printing the status of each server.



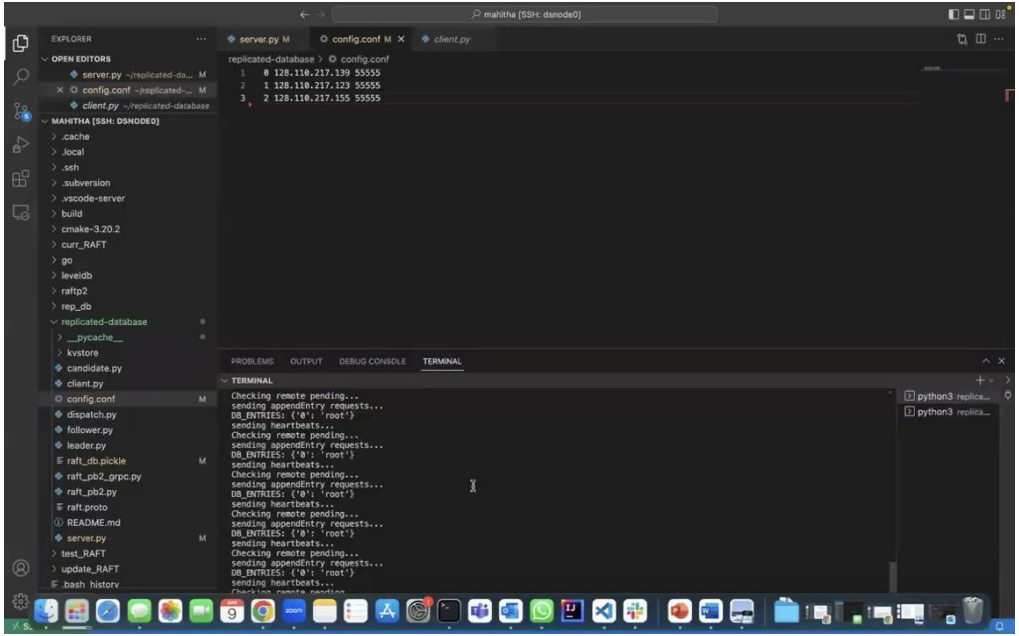
## No Leader Election upon Majority Node Crash Testing

- Tested that no leader gets elected when the majority of nodes are down.
- Tested by crashing two nodes in a three node cluster and checking that the third node continues to be a candidate by printing its status.



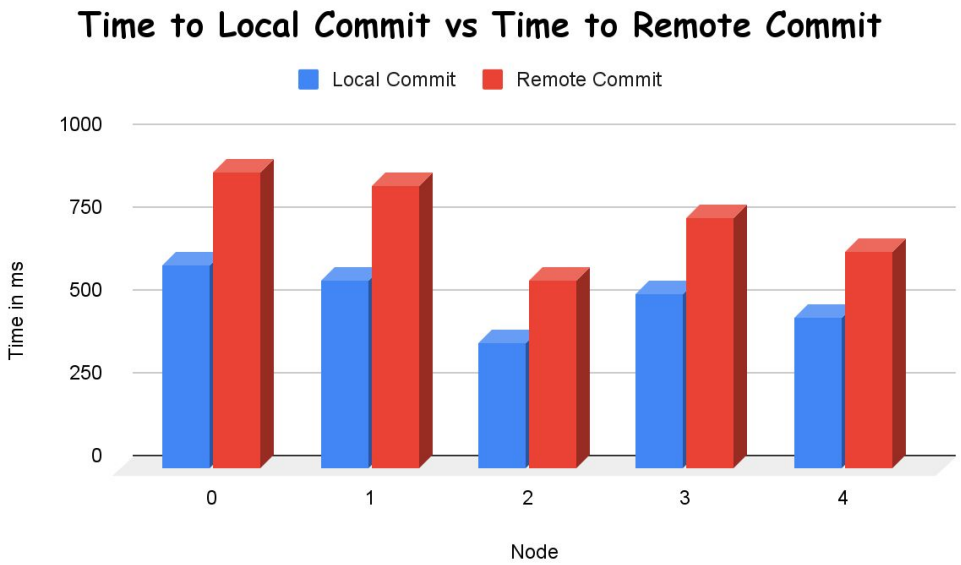
## Log Replication Testing

- Tested client response after log replication to all nodes by printing logs on each server and checking response on client side.
- Tested log replication on all nodes by printing logs on nodes after sending a request from the client.
- Tested synchronization of a crashed node with the majority of nodes alive after recovery by suspending a node from the cluster, sending a few requests to the other majority nodes, and checking if logs in the crashed node match the majority nodes after recovery.

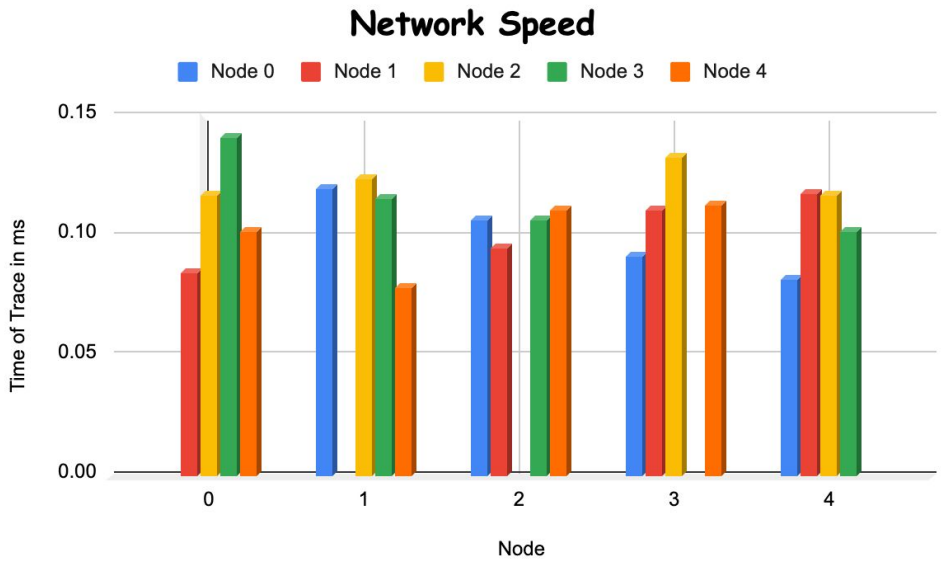


# Performance Tests

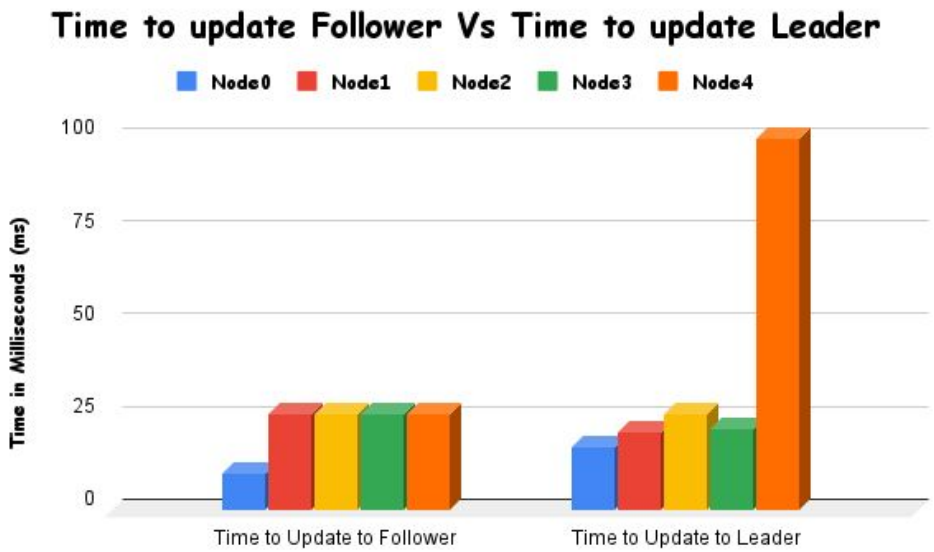
| Nodes | Time to local commit | Time to remote commit |
|---|---|---|
| 0 | 613ms | 895ms |
| 1 | 568ms | 856ms |
| 2 | 382ms | 568ms |
| 3 | 525ms | 759ms |
| 4 | 458ms | 656ms |



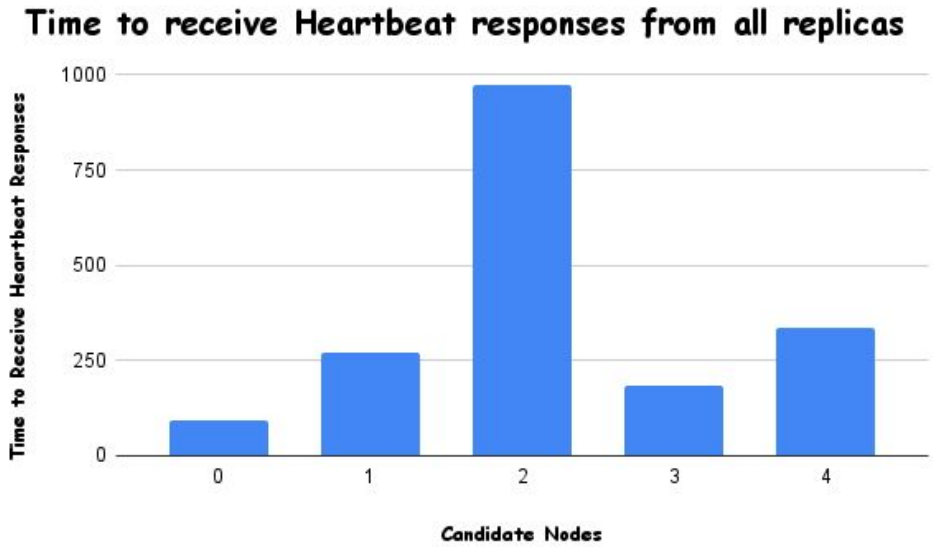
| Node | Time of trace to Node 0 (ms) | Time of trace to Node 1 (ms) | Time of trace to Node 2 (ms) | Time of trace to Node 3 (ms) | Time of trace to Node 4 (ms) |
|---|---|---|---|---|---|
| 0 | - | 0.085 | 0.117 | 0.141 | 0.102 |
| 1 | 0.120 | - | 0.124 | 0.116 | 0.079 |
| 2 | 0.107 | 0.095 | - | 0.107 | 0.111 |
| 3 | 0.092 | 0.111 | 0.133 | - | 0.113 |
| 4 | 0.082 | 0.118 | 0.117 | 0.102 | - |



| Node | Time to update to Follower (ms) | Time to update to Leader (ms) |
|---|---|---|
| 0 | 10 | 17 |
| 1 | 26 | 21 |
| 2 | 26 | 26 |
| 3 | 26 | 22 |
| 4 | 26 | 100 |



| Node | Time to receive heartbeat responses from all replicas (ms) |
|---|---|
| 0 | 91 |
| 1 | 269 |
| 2 | 974 |
| 3 | 184 |
| 4 | 337 |



# Inference

- Node 2's connection speed is equal to that of all other nodes, indicating that it may hold a central position in the network.
- This could explain why committing on all replicas is faster through Node 2 compared to other nodes.
- The time taken to commit changes to the local database is generally lower than the time taken to commit changes remotely, across all nodes.
- The network speeds are not symmetric, i.e., the speed from Node A to Node B may not be the same as the speed from Node B to Node A. For example, the speed from Node 0 to Node 1 is 0.085, while the speed from Node 1 to Node 0 is 0.12.

# Thanks