REPORT

Introduction

The Raft algorithm provides a mechanism for achieving consensus among a cluster of nodes, ensuring that they agree on the state of the system even in the presence of failures.

The system consists of multiple Raft nodes, each representing a server in the distributed system. These nodes communicate with each other using Remote Procedure Calls (RPCs) to coordinate their actions and maintain consistency.

Key Components

RaftNode Class

Represents a Raft node with functionalities related to state management, RPC handling, election process, log replication, committing entries, and client interaction.
Implements methods for handling RPCs such as serving client requests and managing appendEntries and requestVote RPCs.
Timeout Class

Manages election and heartbeat timeouts for initiating elections and sending heartbeats respectively.
Client Interaction

Defines RPC methods for serving client requests such as setting and retrieving key-value pairs from the database.
Clients interact with the Raft cluster through these RPCs, ensuring consistency and fault tolerance.
Storage and Database Operations

Handles storage and database operations by persisting logs and metadata to disk.
Stores logs for write operations and no-op entries along with metadata like the current term, voted-for candidate, and commit index.
Standard Raft RPCs

Implements standard Raft RPCs (AppendEntry and RequestVote) for communication between nodes to maintain consensus and elect leaders.
Election Functionalities

Implements the leader election process, including initiating elections, receiving vote requests, transitioning to leader state upon receiving majority votes, and managing leader leases.
Log Replication Functionalities

Manages log replication by sending periodic heartbeats from the leader to followers, replicating logs upon receiving client SET requests, and committing entries when conditions are met.
Committing Entries

Ensures that entries are committed only when conditions such as majority acknowledgment and leader term consistency are satisfied.
Metadata Storage

Persists metadata such as term, voted-for candidate, and commit index along with logs to ensure continuity even after node restarts.
Error Handling

Provides error handling for RPC communications and log consistency checks to ensure the integrity of the distributed system.

Conclusion

In conclusion, the implementation of the Raft consensus algorithm provides a robust mechanism for maintaining fault-tolerant replication and consistency of the distributed key-value database across multiple nodes. By following the Raft algorithm's principles and design, the system ensures reliability and resilience in the face of failures, making it suitable for distributed systems requiring strong consistency guarantees.

This concludes the report on the implementation of the Raft consensus algorithm.

How to run the Code?

On the respective VMs, run the server node code using the command:

```
python raft_server_node0.py --node_id 0 --node_ip localhost --node_port 60000
```

To run the client code:

```
python raft_client.py
```