Distributed Key-Value Store Using Raft Consensus Algorithm

Hamza Amin Khokhar, Mohammad Faizan, Azeem Waqar

March 2025

Contents

1 Introduction

This report examines the implementation of a distributed key-value store utilizing the Raft consensus algorithm, as developed in the repository available at `https://github.com/hak2979/Raft-Algorithm`. The primary objective of this project is to ensure consistency and fault tolerance in distributed systems through effective leader election and log replication mechanisms.

2 Project Overview

The project encompasses the following core features:

- **Put(Key, Value)**: Stores a key-value pair in the system.
- **Append(Key, Value)**: Appends a value to an existing key.
- **Get(Key)**: Retrieves the value associated with a key.

To achieve fault tolerance, the system implements:

- Automatic leader election upon leader failure.
- Log synchronization for worker nodes upon reconnection.
- Timeout and retry mechanisms for network failures and dropped messages.

3 Raft Consensus Algorithm

The Raft consensus algorithm is designed to manage a replicated log in distributed systems, ensuring consistency across nodes. It divides the consensus problem into two primary components:

1. **Leader Election**: Ensures that only one leader is active at a time, managing all client interactions and log replication.
2. **Log Replication**: Guarantees that all nodes maintain an identical log by replicating entries from the leader to the followers.

Raft achieves consensus via an elected leader. A server in a Raft cluster is either a leader or a follower, and can become a candidate during an election if the leader is unavailable. The leader is responsible for log replication to the followers and regularly informs them of its existence by sending heartbeat messages. Each follower has a timeout period in which it expects the heartbeat from the leader. If no heartbeat is received, the follower transitions to a candidate state and initiates a leader election.

4 Code Structure and Explanation

The project's codebase comprises the following key files:

- `main.go`: Initializes the server and manages Raft consensus operations.
- `client.go`: Handles client interactions and forwards requests to the leader.
- `go.mod`: Manages module dependencies.
- `README.md`: Provides an overview of the project and its features.

4.1 `main.go`

This file is the entry point of the application. It initializes the server, sets up the Raft consensus mechanism, and listens for incoming client requests. The server operates in different roles (Leader, Follower, Candidate) as defined by the Raft algorithm.

4.2 `client.go`

This file manages client interactions. It sends client requests to the leader node and handles responses. If the leader is unavailable, it implements retry mechanisms to ensure the request is eventually processed.

4.3 `go.mod`

This file specifies the module path and manages the project's dependencies, ensuring that all required packages are correctly versioned and available.

4.4 `README.md`

This file provides a comprehensive overview of the project, detailing its features, the implementation of the Raft consensus algorithm, and fault tolerance mechanisms.

5 Fault Tolerance Mechanisms

The system incorporates several fault tolerance mechanisms:

- **Leader Failure Handling**: Upon detecting a leader failure, the system initiates a new leader election to ensure continued operation.
- **Worker Node Recovery**: When a worker node reconnects after a failure, it synchronizes its log with the leader to maintain consistency.
- **Network Failure Management**: The system employs timeout and retry mechanisms to handle network failures and dropped messages, ensuring reliable communication between nodes.

6 Conclusion

The implementation of the Raft consensus algorithm in this distributed key-value store ensures consistency and fault tolerance across the system. By effectively managing leader elections and log replication, the system can handle network failures and node crashes, maintaining reliable operation in a distributed environment.