Distributed Database System with Master-Slave Replication

A Go-Based Implementation

Prepared by: Esmail Mohamed Esmail Elkot

Date: May 18, 2025

Submitted as part of the Distributed Systems Project

1 Introduction

This report presents a distributed database system implemented in Go, utilizing a master-slave replication architecture. The system comprises one master node (port 8083) and two slave nodes (ports 8084 and 8085), communicating via HTTP. The master node manages primary database operations, while slaves replicate these operations to ensure data consistency. The system integrates with MySQL and provides interactive terminal dashboards for user interaction.

The primary objectives are to demonstrate master-slave replication, ensure fault tolerance through health monitoring, and provide a user-friendly interface for database management. This report details the system's architecture, implementation, usage, and potential improvements.

2 System Architecture

The system employs a master-slave architecture, where the master node is the single point of truth for write operations (CREATE, DROP, INSERT, UPDATE, DELETE). Slaves handle read operations and replicate write operations received from the master. The architecture is illustrated below using a textual description due to the absence of graphical figures.

- Master Node: Runs on port 8083, connects to MySQL, and exposes HTTP endpoints for database operations. It maintains a replication queue and monitors slave health.
- Slave Nodes: Run on ports 8084 and 8085, connect to MySQL, and register with the master. They handle replication requests and forward write operations to the master.
- MySQL Database: Serves as the storage backend, accessible by all nodes using the credentials root:k7l15981.
- HTTP Communication: Uses RESTful APIs for node communication, with CORS enabled for flexibility.

Replication is asynchronous, with the master queuing tasks (e.g., ReplicationTask) and sending them to slaves via HTTP requests. Slaves execute these tasks on their local MySQL instances.

3 Implementation Details

The system is implemented in Go, leveraging packages like net/http, database/sql, and go-sql-driver/mysql. The codebase is split into three files: master.go, slave1.go, and slave2.go.

Listing 1: Master Node Replication Queue

```
type ReplicationTask struct {
    Operation string
    Data      map[string]interface{}
```

```
}
var replicationQueue = make(chan ReplicationTask, 1000)
func replicationWorker() {
    for task := range replicationQueue {
        slaveConnections.Range(func(key, value interface{}) bool {
            addr := key.(string)
            status := value.(bool)
            if status {
                go replicateToSlave(addr, task)
            }
            return true;
        })
    }
}
```

The master node implements endpoints like /createdb, /insert, and /select, while slaves implement /replicate/* endpoints. Dashboards use terminal-based interfaces, clearing the screen with ANSI codes or Windows commands. Health checks are performed every 5 seconds for slaves and 10 seconds for the master.

Table 1: Master Node HTTP Endpoints

Endpoint	Description
/ping	Checks node availability
/createdb	Creates a new database
/insert	Inserts a record into a table
/select	Retrieves records from a table

4 Usage and Testing

To use the system, users must run the master and slave nodes after setting up MySQL and Go. The master dashboard allows creating databases, tables, and records, while slaves display replicated data. Write operations from slaves are forwarded to the master.

For example, to create a database and table:

1. On the master dashboard, select option 1 and enter mydb.
2. Select option 3, enter mydb, users, and id INT, name VARCHAR(255).
3. Verify replication by checking the slave dashboards (option 3).

Testing involved creating databases, inserting records, and verifying replication across nodes. The system handles basic error cases (e.g., missing parameters) but requires enhancements for robustness.

5 Conclusion and Future Work

This project successfully demonstrates a distributed database system with master-slave replication. It provides a practical implementation of distributed systems concepts, including replication, health monitoring, and HTTP-based communication.

Future improvements include:

- Implementing a robust leader election algorithm (e.g., Raft).
- Adding transaction support for atomic operations.
- Enhancing error handling and logging.
- Supporting distributed deployments beyond localhost.