University of Science and Technology of Hanoi

Distributed Systems
Midterm Report

Distributed Database: Database Replication

Quang Vo Hong [quangvh.22bi13386@usth.edu.vn]
Tam Nguyen Duc [tamnd.22bi13400@usth.edu.vn]
Quang Le Anh [quangla.22bi13380@usth.edu.vn]
Nguyen Vu The Khoi [nguyenvtk.22bi13344@usth.edu.vn]
Quan Hong Chu [quanch.22bi13367@usth.edu.vn]

Hanoi, December 2024

Contents

1 Introduction

1.1 Context

A Distributed database is a database shared by multiple servers or computers, instead of limited to one system. Inside a Distributed database system, each component contains its own database connected with other databases. A Distributed database has higher benefits than a centralized database system as it provides faster data processing between sites. It also ensures that the system can still execute if one or more sites fail to operate.

One of the most important parts of a Distributed database is Replication. Replication is a method used for storing data. In the Replication approach, systems maintain copies of data stored in multiple sites instead of only one main copy in the main site. With Replication, data can be accessed at different sites in parallel. The system is also ensured to continue to operate even if a server fails as there are copies of the same data on different other servers.

Following the Replication method, MySQL Replication is a process that can automatically copy data from one MySQL server to other replica servers. The replica servers are proposed to be updated from the source's data.

The efficient synergy between MySQL and the Replication method is widely used for numerous purposes:

- Balancing Loads: Queries can be redirected to replica servers in order to reduce the load on the source server.
- System Availability: MySQL Replication can ensure the source system can still operate in case one or more servers fail to operate.
- Backup Solution: Replica servers can be used for backups without interaction with the main database.
- Disaster Recovery: A replica server can be quickly promoted to the primary server in the event of catastrophic failures.

1.2 Objectives

Our main objective is building a MySQL Master-Slave Replication system. The system can copy and update data from a primary database (Master database) to one or more secondary databases (Slave databases). The Master database will receive all write operations including inserts, updates, deletions, etc... After the Master database is modified, a copy of it will be kept in a Slave database. The copy process will be executed automatically whenever the master database is altered.

1.3 Outcomes

A MySQL Master-Slave Replication system can automatically copy the Master database to a Slave database every time the Master database is modified.

2 Methodology

2.1 System Architecture

Our system consists of 5 components: Master Database, Master Server, Message Bus, Slave Server and Slave Database. User will interact with the Master Database to perform write operations. The modifications will be forwarded to Message Bus and a Slave Database will be created for storing a copy of the Master Database.

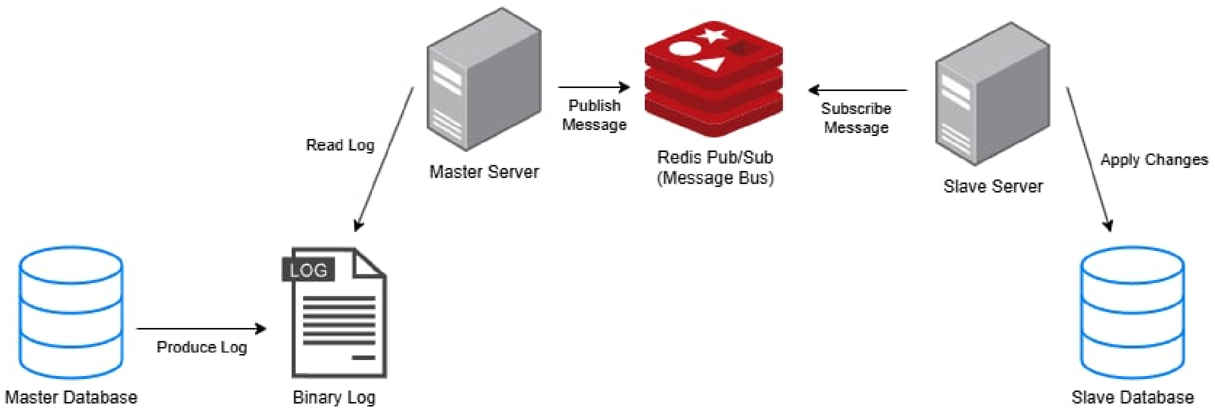

Figure 1: System Architecture

2.2 Role Assignments

Each component of the system plays an important role in the system:

Master Database

The Master Database is the system source database. It will handle all write operations and create a binary log whenever there is a modification to the database.

Master Server

The Master Server can access binary logs created by the Master Database. It will extract the modifications of the database from the binary logs. The Master Server publishes to a Redis Pub/Sub channel for sending modifications messages.

Message Bus

The Message Bus is used for connecting messages between the Master Server and the Slave Server. It act as a layer for multiple Slave Server interact with the

Master Server at the same time. We use Redis Pub/Sub for creating channels for transmitting data between Servers.

Slave Server

The Slave Server subscribe to a Redis Pub/Sub channel for receiving modifications messages. It will then update and apply to the Slave Database to create copies of the Master Database.

Slave Database

The Slave Database is a copy of the Master Database enabled with read operations. Each Slave Database contains a copy of the Master Database after each modification made to the source database.

3 Implementation

3.1 Configuration

In order to build the system, we have to enable MySQL binary logging to capture database changes. We need to configure MySQL:

Edit MySQL Configuration File

To edit the MySQL configuration file, use the following command:

```
sudo nano /etc/mysql/my.cnf
```

Enable Logging

Add the following configuration to enable binary logging in the ‘[mysqld]‘ section:

```
[mysqld]
log_bin = mysql-bin
server_id = 1
binlog_format = ROW
```

Limit Logs to a Specific Database

Add the following configuration to to limit logs a specific database (for example ‘master‘):

```
binlog-do-db=master
```

Add Options to Log Table Names

To log column names and metadata, add the following options to the ‘[mysqld]‘ section:

```
binlog_row_image = FULL
binlog_row_metadata = FULL
```

With these settings, MySQL can access the Binary Log. All data modifications will be logged, including column names and metadata.

3.2 Modules

We take advantages of the following modules for building and managing the MySQL Replication system:

SQLAlchemy

SQLAlchemy is a Python SQL toolkit used to handle SQL queries. We use this library for creating operations with the database using Python.

pymsqlreplication

pymysqlreplication is a Python module used for reading binary logs. It will extract database changes for creating Slave databases.

Redis Pub/Sub

Redis pub/sub acts as a message bus to publish and subscribe to events. The Master Server will publish to Redis and the Slave server will subscribe to it also.

4 Result

4.1 Functionality Tests

Based on the initial objectives specified, we conducted a detailed test on all functions of the system to evaluate usability and make sure they meet expectations.

For Data Definition Language Testing, these functionalities were tested:

4.1.1 Testing CREATE TABLE replication

We tested executing command on the Master Database:

```
"CREATE TABLE test_table"
```

Result: **Success**
A table 'test_table' was created accordingly in the Slave Database

4.1.2 Testing DROP TABLE replication

We tested executing on the Master Database:

```
"DROP TABLE test_table;"
```

Result: **Success**
The table 'test_table' was dropped respectively on the Slave Database.

For Data Modification Language Testing, these functionalities were tested:

4.1.3 Testing INSERT replication

We tested executing on the Master Database:

```
"INSERT INTO test_table (id, name) VALUES (1, 'Alice');"
```

Result: **Success**
The item with values (1, 'Alice') was inserted in the Slave Database.

4.1.4 Testing UPDATE replication

We tested executing on the Master Database:

```
"UPDATE test_table SET name = 'Bob' WHERE id = 1;"
```

Result: **Success**
The item with values (1, 'Alice') was updated into (1, 'Bob') in the Slave Database.

4.1.5 Testing DELETE replication

We tested executing on the Master Database:

```
"DELETE FROM test_table WHERE id = 1;"
```

Result: **Success**
The item with id = 1 was deleted in the Slave Database.

4.2 Conclusion

The implementation of the MySQL Master-Slave Replication system successfully achieved the objectives at the beginning of the project. The system is able to replicate data modifications, including Data Definition Language and Data Modification Language operations, from the Master Database to the Slave Database.