IT559 – Distributed Systems Project

Distributed Banking System (MySQL-Based)

Team ID: 20

Team Members:
GALCHAR RAHULKUMAR – 202201100
CHAUDHARI SAHIL – 202201171
YADAV ALPESH – 202201264

1. Problem Statement

To build a distributed banking system using a centralized MySQL database to store account and transaction data, with support for concurrency, thread safety, and data persistence, accessible by multiple servers and clients via XML-RPC.

2. System Overview

The system is composed of a master server connected to a MySQL database, and multiple threaded servers (S1, S2) that handle client requests. The client can perform banking operations like account creation, deposits, withdrawals, transfers, and statement generation.

3. System Architecture and Components

- **Master Server (M1):** Handles all banking operations and stores data in MySQL using transactions.
- **Server 1 (S1) and Server 2 (S2):** Threaded XML-RPC servers that connect to clients and forward requests to the master.
- **Clients (C1):** Terminal-based menu-driven interface for performing banking operations.

4. Distributed System Concepts Applied

- **Remote Procedure Call (RPC):** Communication among client, servers, and master is implemented using XML-RPC.

- **Threading and Concurrency:** Servers use `ThreadingMixIn` to handle multiple clients.
- **Race Condition Handling:** Master server uses `SELECT ... FOR UPDATE` to lock account rows during critical operations.
- **Consistency and Durability:** MySQL transactions ensure ACID properties are maintained.

5. Implementation Details

- **Language:** Python 3
- **Technologies:** `xmlrpc.server`, `mysql-connector-python`, `threading`
- **Database:** MySQL 8.x or later
- **Persistence:**
  - **TXT Version:** File-based logs in `master_data/`
  - **MySQL Version:** Central database with `accounts` and `transactions` tables

6. Results and Performance Analysis

- The system successfully supports concurrent clients performing transactions on shared accounts.
- Race conditions are handled at the database level with row-level locks.
- Operations reflect immediately across servers due to centralized MySQL backend.
- The MySQL version scales better and ensures consistency across restarts and concurrent sessions.

7. Challenges Faced and Solutions

- **Race conditions:** Solved using MySQL's row-level locking and transactions.
- **Concurrent access:** Managed using Python's multithreaded XML-RPC servers.
- **Data loss on failure:** Handled by storing data persistently in MySQL.
- **Maintaining consistency across servers:** Ensured by querying data from the master.

8. Future Improvements and References

Future Improvements

- Add a web-based user interface.
- Use microservices with RESTful APIs.
- Enable replication and high availability features in MySQL.
- Implement logging and auditing features for admin use.

References

- MySQL Docs: `https://dev.mysql.com/doc/`
- Python MySQL Connector: `https://pypi.org/project/mysql-connector-python/`
- Python XML-RPC: `https://docs.python.org/3/library/xmlrpc.server.html`
- Tanenbaum, A.S. – Distributed Systems: Principles and Paradigms