

IT559 – Distributed Systems Project

Distributed Banking System

Team ID: 20

Team Members:

GALCHAR RAHULKUMAR – 202201100

CHAUDHARI SAHIL – 202201171

YADAV ALPESH – 202201264

1. Problem Statement

To develop a distributed banking system that supports core banking functionalities (account creation, deposit, withdrawal, transfer, balance check, and statement) using a distributed architecture with concurrency support and persistent logging through text files.

2. System Overview

This version of the banking system uses Python and XML-RPC for communication between components and persists all transaction and balance data using plain text files. It includes a centralized Master Server and two distributed worker servers (S1 and S2), all running concurrently and serving multiple clients.

3. System Architecture and Components

- **Master Server (M1):** Stores all account balances and logs in `master_data/`. All operations are processed here.
- **Server 1 and Server 2:** Threaded servers handling client requests and forwarding them to the master server. Maintains history logs per session.
- **Clients (C1):** Provide a menu-driven interface for users to perform banking operations.

4. Distributed System Concepts Applied

- **Remote Procedure Call (RPC):** All communication between clients, servers, and the master is done via XML-RPC.

- **Threading:** Worker servers are implemented using `ThreadedXMLRPCServer` to handle multiple clients.
- **Mutual Exclusion:** `threading.Lock` is used to prevent race conditions during file updates.
- **Data Consistency:** Only the master server modifies account balances to ensure global consistency.

5. Implementation Details

- **Language:** Python 3
- **Technologies:** `xmlrpc.server`, `threading`, file I/O
- **Persistence:** Text files named `data_<account>.txt` stored in `master_data/`
- **Multithreading:** Threaded XML-RPC server ensures simultaneous clients are handled properly

6. Challenges Faced and Solutions

- **Inconsistent view between servers:** Modified `account_exists()` to check with master server in real-time.
- **Concurrency issues:** Solved using locks in both master and worker servers.
- **File corruption during parallel access:** Implemented append-only logging and atomic writes with locks.

7. Results and Performance Analysis

- System worked successfully for all core banking operations.
- Multi-client support was validated through simultaneous client terminals.
- File-based logging proved to be simple and efficient for small-scale testing.

8. Future Improvements and References

Future Improvements

- Move to a database for better scalability and durability.
- Add authentication for secure access.
- GUI-based or web-based client interfaces.
- Replication and crash recovery support.

References

- Python XML-RPC: <https://docs.python.org/3/library/xmlrpc.server.html>
- Python Threading: <https://docs.python.org/3/library/threading.html>
- Tanenbaum, A.S. – Distributed Systems: Principles and Paradigms