CSE 531: gRPC Written Report

Shachi Shah
School of Computing Augmented Intelligence
Arizona State University
Azusa, California, United States of America
spshah22@asu.edu

I. PROBLEM STATEMENT

The problem at hand focuses on building a distributed banking system where multiple customers interact with various branches of a bank, each performing operations such as deposits, withdrawals, and balance inquiries. In a real-world scenario, these interactions need to happen in a distributed environment, where customers are not directly interacting with the central bank system but rather through intermediary branches. These branches act as a bridge between the customers and the bank's core system, maintaining up-to-date information on the customers' accounts. The challenge lies in ensuring the consistency and accuracy of the account balances across multiple customers and branches, especially when operations are happening concurrently. The problem becomes more complex when scaled to handle dozens or even hundreds of customers and branches operating in parallel. To address this challenge, the project utilizes the gRPC (Google Remote Procedure Call) framework, which facilitates communication between distributed services over a network.

At its core, the problem demands the implementation of an efficient and reliable communication structure between clients (representing customers) and servers (representing branches) using gRPC. Each customer sends requests to a designated branch for transactions like depositing money, withdrawing funds, or checking their account balance. The system must be designed to handle these requests simultaneously, ensuring that the correct balance is maintained and updated without errors. Additionally, the solution must be robust enough to handle a large number of transactions while maintaining consistency across the distributed system. The project also introduces potential issues like race conditions and conflicts in balance updates due to concurrent requests, which the system must mitigate. The problem, therefore, encompasses designing and implementing a distributed banking system that ensures correctness, efficiency, and fault tolerance in handling client requests.

II. GOAL

The primary goal of this project is to design and implement a distributed banking system using gRPC that can handle the transactions of multiple customers across several branches. The system must simulate a real-world banking scenario where each customer interacts with a specific branch to perform deposits, withdrawals, and balance inquiries. The implementation should ensure that all transactions are accurately processed and that the account balances remain consistent, even when multiple customers are interacting with the same branch concurrently. A key objective is to demonstrate the effective use of gRPC to manage remote procedure calls in a distributed system and handle the challenges of concurrency in a multi-client environment.

Another important aspect of the project is the correct mapping of customers to branches. Since each branch can only handle a fixed number of customers (50) which was one of the parameters I had the most difficulty accommodating for, the system must efficiently assign customers to the appropriate branches using logic such as modulo operations to handle customer IDs. Additionally, the system needs to ensure that branch balances are updated correctly in response to customer transactions, and it must return the appropriate results for balance queries after each transaction. The project aims to pass a series of automated tests (through an autograder) that evaluate the accuracy, efficiency, and concurrency management of the system. Ultimately, the goal is to build a robust, scalable distributed system that can handle the complexities of real-world banking transactions in a simulated environment.

III. SETUP

After reading the documentation for what gRPC is, the set-up of the project required building a gRPC-based communication structure between the client (which simulates the customers) and the server (which manages the branches). The following explains what I initiated and completed into my setup process:

1. Install Dependencies: Install gRPC and Protocol Buffers using pip for gRPC communication and Protocol Buffers compilation.
2. Design Protobuf File: Define the service (BankService) and message structures in banks.proto for deposits, withdrawals, and queries.
3. Compile Protobuf File: Use the command python -m grpc_tools.protoc -I./protos --python_out=. --grpc_python_out=. protos/banks.proto to generate the Python gRPC files (banks_pb2.py and banks_pb2_grpc.py).
4. Implement Server: Create a server (branch.py) that listens for gRPC requests from clients, processes transactions, and updates branch balances.
5. Implement Client: Implement the client (client.py) to read input data (customers and their events), execute events, and interact with the branches via gRPC.
6. Run the Server and Client: First, start the server using python server.py, then run the client with the input file using python client.py input.json.

IV. IMPLEMENTATION PROCESSES

The implementation process began with defining the .proto file, which specified the available services (like Deposit, Withdraw, and QueryBalance) and their corresponding request and response messages. After compiling the proto file, these services were implemented in the server. For example, the Deposit service would receive a deposit request from the client, update the branch's balance accordingly, and send a success response back to the client. Similarly, the Withdraw service would reduce the balance if sufficient funds were available and respond to the client with the result.

1. Designing the Protobuf File (banks.proto): The first step involves defining the communication between clients (customers) and the server (branches) using Protocol Buffers. This file specifies the structure for deposit, withdrawal, and query requests, along with the service interface (BankService) that branches will use to handle these operations.
2. Compiling the Protobuf File: After designing the Protobuf file, it is compiled into Python files using the gRPC compiler tool. This process generates the necessary files (banks_pb2.py and banks_pb2_grpc.py) for gRPC communication, which are used by both the server and the client.
3. Implementing the Server (branch.py): The server implementation creates the Branch class, which processes customer transactions like deposits, withdrawals, and balance queries. Each branch maintains its own balance, and the server responds to customer requests by adjusting balances or returning current balances. The server uses gRPC to receive requests and send responses.
4. Client-Side Logic (client.py): On the client side, each customer runs a series of events (deposit, withdraw, or query). Customers send these requests to branches and receive responses. The client uses the generated gRPC stubs from Protobuf to communicate with the server and log each transaction's result.
5. Mapping Customers to Branches: To distribute customers across branches, the customer ID is moduloed by 50 to assign them to one of the 50 branches. If the result is 0, the customer is assigned to Branch 50. This ensures an even distribution of

customers and prevents all customers from interacting with the same branch.

Implementation of all of these components along with the prior setup greatly helped me in getting some initial results, even if they were not accurate. Because understanding how each file was used to communicate to one another was more important than getting correct results right away.

V. RESULTS

The results of the project were reflected in the correct simulation of customer-branch interactions, where deposits and withdrawals were accurately processed, and the balances were updated accordingly. The server was able to handle multiple clients concurrently, demonstrating the efficacy of the gRPC framework in facilitating communication in a distributed system. Initial test runs revealed discrepancies in the balance calculations, which were traced back to issues with customer ID handling. By implementing the modulo operation, these errors were corrected, and subsequent test runs produced accurate results.

The results of the project are evaluated based on how accurately the system handles customer transactions and the correctness of the final balances. Some examples of the results obtained include:

1. Successful Deposits and Queries: For each customer, the system correctly processes deposits and queries the updated balance. For instance:
    a. Customer 1 deposits \$10, and a subsequent query shows a balance of \$410.
    b. Customer 2 deposits \$10, and the query response shows a balance of \$420, indicating that the deposit was processed correctly.
2. Handling Withdrawals: For customers who perform withdrawals, the system correctly reduces the balance. For example:
    a. Customer 51 withdraws \$10 from an initial balance of \$400, and the query shows a balance of \$390, indicating the withdrawal was processed successfully.
    b. Customer 52 withdraws \$10, and the balance is updated to \$380, demonstrating correct handling of withdrawals.

REFERENCES

[1] gRPC Quick Start, gRPC, https://grpc.io/docs/languages/cpp/quickstart/. Accessed 25 October 2024.

[2] Sadri, M. 2024. *CSE 531_gRPC Project Overview Document*, Ira A. Fulton Schools of Engineering, Arizona State University. Accessed 25 October 2024.