Ira A. Fulton Schools of Engineering
Arizona State University

CSE 531: Distributed and Multiprocessor Operating Systems

gRPC Written Report

Problem Statement

The project focuses on building a distributed banking system that allows multiple customers to deposit or withdraw money at various branches. Each branch maintains a replica of the overall account balance, and ensuring these replicas are synchronized is essential to maintain consistency across the system. Despite each customer only interacting with their assigned branch, the updates made to one branch's balance must propagate to all other branches, ensuring that all replicas reflect the latest account state. This setup highlights challenges typical of distributed systems, such as consistency, communication efficiency, and state management.

Goal

The goal is to implement a robust, distributed system using gRPC that supports customer transactions and branch-to-branch synchronization of account balances. This requires building reliable communication channels between customers and branches, with additional inter-branch updates to maintain consistency across distributed replicas. Key goals include demonstrating gRPC's effectiveness in real-time, distributed settings and validating that all customer transactions are consistently reflected across branches.

Setup

The project's technology stack includes:

- **Python** for core development and process control,
- **gRPC** to handle remote procedure calls, enabling efficient inter-process communication,
- **Protocol Buffers** for defining service interfaces and serializing data in a compact, cross-language format.

Library dependencies such as **grpcio==1.64.1, grpcio-tools==1.64.1, and protobuf==5.27.2** are integral for setting up the gRPC environment in Python [1], [2]. Using Protocol Buffers allows developers to define service methods and message formats in .proto files, which are then compiled to generate Python code for server and client communication. This setup reduces message size and enhances the efficiency of data exchange, which is particularly valuable in distributed environments where bandwidth is limited.

Implementation Processes

- **Service Interface Definition**: The first step involves defining the service interfaces in Protocol Buffers. Key services include Deposit, Withdraw, and Query for customer-initiated transactions, along with Propagate_Deposit and Propagate_Withdraw for inter-branch communication. These services facilitate state consistency across branches by ensuring that each branch receives updates from other branches following any transaction. This method of defining services in .proto files enables easy code generation for both the client and server sides using grpcio-tools, which significantly simplifies the setup [1].
- **Customer and Branch Classes**: The system implements customer and branch processes as Python classes, which are then configured to interact using gRPC stubs. Each customer is associated with a specific branch and can only communicate with that branch, using the createStub and executeEvents functions to initiate requests. Branches, on the other hand, maintain a balance variable to record the account balance and a stubList to manage inter-branch communication. This setup allows each branch to manage local transactions independently, while also coordinating with other branches to ensure that the distributed balance remains accurate [2].
- **Transaction Propagation and Synchronization**: When a customer initiates a transaction, such as a deposit or withdrawal, the branch processes this request by updating its local balance. The branch then uses the Propagate_Deposit or Propagate_Withdraw service to inform other branches about the balance change. gRPC's support for different RPC types, including unary and streaming RPCs, enables efficient communication for both single transactions and continuous updates. This setup ensures that all branches consistently reflect the latest account state, preventing discrepancies in customer-visible balances. Techniques like bidirectional streaming allow each branch to send and receive balance updates continuously, ensuring smooth synchronization across the network [3].
- **Execution and Testing**: The main program loads an input file that specifies each customer's event sequence and initial balances for each branch. Customer events, such as deposits and queries, are processed sequentially, with slight delays to allow each branch's updates to propagate before the next transaction. After execution, the program outputs the results in JSON format, which contains the list of successful transaction responses each customer received from their assigned branch. This format allows for easy validation of balance consistency across branches and verifies that each transaction was processed accurately.

Results

The implemented system successfully manages deposit, withdrawal, and query operations across distributed branches, ensuring that all branches remain synchronized. For instance, when a customer deposits funds at one branch, that branch updates its balance and propagates the change to all other branches, achieving global consistency. The output JSON file, which lists each customer's successful transactions and the resulting balances, confirms that updates are accurately synchronized across branches.

This result validates the project's objectives by showcasing gRPC's capability to support efficient communication and state consistency in distributed systems. Protocol Buffers further optimize the setup by reducing message sizes, which minimizes bandwidth usage—a critical factor in distributed applications [3]. The success of this implementation underscores the effectiveness of gRPC for real-time data consistency in multi-node networks and highlights Protocol Buffers' benefits for efficient data serialization in complex distributed environments [4].

References

[1] "Quick start | Python | gRPC," grpc.io. Available: https://grpc.io/docs/languages/python/quickstart/. Accessed: Nov. 2024.
[2] "Basics tutorial | Python | gRPC," grpc.io. Available: https://grpc.io/docs/languages/python/basics/. Accessed: Nov. 2024.
[3] N. Manchanda, "Implementing gRPC In Python: A Step-by-step Guide," Velotio Technologies, 2022. Available: https://www.velotio.com. Accessed: Nov. 2024.
[4] "Protocol Buffers | Google Developers," Google, 2024. Available: https://developers.google.com/protocol-buffers. Accessed: Nov. 2024.