Ira A. Fulton Schools of Engineering
Arizona State University

CSE 531: Distributed and Multiprocessor Operating Systems

Client-Centric Consistency Written Report

Name: Varad Deshmukh **ASU ID: 1225369184**

Problem Statement

This project showcases a distributed system with gRPC implementation, focusing on Client-Centric consistency via the Read-your-Writes model. It utilizes an illustrative diagram from the Project 3 documentation and extends the consistency model built in Project 1. Following are the tasks for this project:

1. Tracking read and write events across different branch processes by the same customer.
2. Implementing Read-your-Writes consistency among branch processes.

Goal

The primary objective of this project revolves around comprehending and executing consistency and replication mechanisms. Client-centric consistency stands as a pivotal aspect to afford clients an uninterrupted view of the most recent data, regardless of its storage location. As the project involves a single customer traversing multiple branches, it becomes imperative to maintain a uniform and consistent balance across all these branches for the said customer. Read-your-write consistency is a crucial implementation target in this endeavor as it ensures seamless synchronization between branches, guaranteeing that any updates made by the customer are immediately reflected across all relevant branches.

Setup

[What are the relevant technologies for the setup and their versions?]

1. Setup Ubuntu on Windows with WSL 2
   - **Ubuntu 22.04.2**

2. Install Python
   - **Python 3.10.6**

3. Install Python libraries:
   - **grpcio==1.59.0**
   - **grpcio-tools==1.59.0**
   - **protobuf==4.24.3**

4. Download and Unzip the submitted zip file.
   - **CSE531_Project_3_Varad.zip**

5. Get into the folder
   - **cd CSE_531_Project_3_Varad**

6. Optional – Build proto file
   - **python3 -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. Relay.proto**

7. Run main.py file with input_big.json
   - **python3 main.py input_big.json**

8. After execution is complete the following output files will be created in the folder
   - **output.json**

Implementation Processes

1. Relay.proto
   - Relay.proto is a proto which contains two rpc services, Cast and Deliver. Cast is branch-to-branch communication and Deliver is customer-to-branch communication. Once the proto file is built, two files are created Relay_pb2 and Relay_pb2_grpc. These files help in using grpc functions like Relay_pb2_grpc.relayServicer and Relay_pb2_grpc.relayStub.
2. Customer.py
   - executeEvents:

   The "executeEvents" function serves the purpose of establishing communication between customers and branches. It utilizes the stub to send each customer event via the "Deliver" function defined in the branch file.
   - Output:

   The “output” function serves the purpose of returning all event execution output to the customer and is called in the main file to save in json format.
3. Branch.py
   - createStubs:

The “createStubs” function is used to create a list of stubs of all branches. These stubs are later used for branch-to-branch communication using gRPC tools.

- Deliver:
  The “Deliver” is a function from Relay_pb2_grpc defined in branch for customer-to-branch communication. It receives the event request from the customer, and it verifies the write request using the function “Check_WriteSet” function. And depending upon the interface it calls either Withdraw or Deposit functions and finally returns a response to customer with updated writeset.
- Cast:
  The “Cast” function serves the purpose of doing branch-to-branch communication. It sends a request from one branch, receives it in destination branch and verifies the writeset using “Check_WriteSet”. Based on the interface it calls either Withdraw or Deposit function to update the balance.
- Deposit,Withdraw:
  Both these functions are used to update the branch balance based on the interface and these functions have Lock associated to them to avoid interference from other requests.
- Propagate_Depsoit, Propagate_Withdraw:
  Both these functions are function are used to propagate the changes in branch to the other branches. They for each stub from stubList and use the “Cast” function to perform branch-to-branch communication.
- Check_WriteSet:
  The basic purpose of the “Check_WriteSet” function is verifying the writes from request with writeset in each branch. If the branch is not up to date, then the returned value will be False. This is to ensure the branch complying with the consistency and has all the previous writes performed.

Results

Below are the result in figure after running main.py with Input_big.json and the running the checker.py with output.json.

```
varad@DESKTOP-F765JFB:~/CSE_531_Project_3_Varad$ python3 main.py input_big.json

NOTE: Wait till all processes are terminated.

Starting Branch Processes

Server started. Listening on port : 50001
Server started. Listening on port : 50002
Server started. Listening on port : 50004
Server started. Listening on port : 50006
Server started. Listening on port : 50005
Server started. Listening on port : 50003
Server started. Listening on port : 50007
Server started. Listening on port : 50008
Server started. Listening on port : 50009
Server started. Listening on port : 50010

Starting Customer Processes

{'id': 1, 'recv': {'interface': 'query', 'branch': 1, 'balance': 0}}
{'id': 1, 'recv': {'interface': 'deposit', 'branch': 1, 'result': 'Success'}}
{'id': 1, 'recv': {'interface': 'query', 'branch': 1, 'balance': 10}}
{'id': 1, 'recv': {'interface': 'query', 'branch': 2, 'balance': 10}}
{'id': 1, 'recv': {'interface': 'deposit', 'branch': 2, 'result': 'Success'}}
{'id': 1, 'recv': {'interface': 'query', 'branch': 2, 'balance': 20}}
{'id': 1, 'recv': {'interface': 'query', 'branch': 3, 'balance': 20}}
{'id': 1, 'recv': {'interface': 'deposit', 'branch': 3, 'result': 'Success'}}
{'id': 1, 'recv': {'interface': 'query', 'branch': 3, 'balance': 30}}
{'id': 1, 'recv': {'interface': 'query', 'branch': 4, 'balance': 30}}
{'id': 1, 'recv': {'interface': 'deposit', 'branch': 4, 'result': 'Success'}}
{'id': 1, 'recv': {'interface': 'query', 'branch': 4, 'balance': 40}}
{'id': 1, 'recv': {'interface': 'query', 'branch': 5, 'balance': 40}}
{'id': 1, 'recv': {'interface': 'deposit', 'branch': 5, 'result': 'Success'}}
{'id': 1, 'recv': {'interface': 'query', 'branch': 5, 'balance': 50}}
{'id': 1, 'recv': {'interface': 'query', 'branch': 6, 'balance': 50}}
{'id': 1, 'recv': {'interface': 'deposit', 'branch': 6, 'result': 'Success'}}
{'id': 1, 'recv': {'interface': 'query', 'branch': 6, 'balance': 60}}
```

The figure is of main.py with input_big.json

```
varad@DESKTOP-F765JFB:~/CSE_531_Project_3_Varad$ python3 checker.py output.json
Consistent balance between branch 1 and branch 2. Balance=10
Consistent balance between branch 2 and branch 3. Balance=20
Consistent balance between branch 3 and branch 4. Balance=30
Consistent balance between branch 4 and branch 5. Balance=40
Consistent balance between branch 5 and branch 6. Balance=50
Consistent balance between branch 6 and branch 7. Balance=60
Consistent balance between branch 7 and branch 8. Balance=70
Consistent balance between branch 8 and branch 9. Balance=80
Consistent balance between branch 9 and branch 10. Balance=90
Consistent balance between branch 10 and branch 1. Balance=100
Consistent balance between branch 1 and branch 2. Balance=90
Consistent balance between branch 2 and branch 3. Balance=80
Consistent balance between branch 3 and branch 4. Balance=70
Consistent balance between branch 4 and branch 5. Balance=60
Consistent balance between branch 5 and branch 6. Balance=50
Consistent balance between branch 6 and branch 7. Balance=40
Consistent balance between branch 7 and branch 8. Balance=30
Consistent balance between branch 8 and branch 9. Balance=20
Consistent balance between branch 9 and branch 10. Balance=10
19 out of 19 cross-branch query events are correct.
varad@DESKTOP-F765JFB:~/CSE_531_Project_3_Varad$
```

The figure is of checker.py with output.json