Advanced Distributed Systems
Assignment 1
Assignment given on: 16th Aug 2021
Due Date: 29th Aug 2021

Assignment 1 – gRPC and failure handling for At-Most-Once Semantics

The object of this assignment is to understand and implement At-Most-Once Semantics scenarios using gRPC and golang.

Assume you are developing software for a bank ABC using Golang. You need to implement the following functionalities. The client and server templates are attached. Comments are added in the files, please make use of it. Also hello.proto file is included for your references.

Server side:
Implement the logic to read the Transactions id from Trans_Processed.txt file, add into the *TransIds[ ]* array, during the starting of the server.

GetBalance()

```
func (*server) GetBalance(ctx context.Context, request
*transactionpb.TransactionRequest) (*transactionpb.TransactionResponse,
error) {
```

Logic to be implemented

- Whenever a request comes from the client read the *TransId* from the Request and check whether the transaction is already processed.
- If it is processed add a message “The transaction has already been processed.” to the TransactionResponse.Data and return from the client.
- Otherwise (If transaction is not processed) read the balance amount from the *Balance.txt* and add the amount in Response. Add the *TransId* into *TransIds[]* array and Trans_Processed.txt file also.

DepositeAmount()

func (l *Listener) DepositeAmount(args *Request, reply *Reply) error

```
func (*server) DepositeAmount(ctx context.Context, request
*transactionpb.TransactionRequest) (*transactionpb.TransactionResponse,
error) {
```

Logic to be implemented

- Whenever a request comes from the client read the *TransId* from the Request and check whether the transaction is already processed.
- If it is processed add a message “The transaction has already processed.” to the

TransactionResponse.Data and return from the client.

- Otherwise (If transaction is not processed) read the balance amount from the Balance.txt amount. Read the amount from Request and add this amount with the balance and delete the balance amount from the file and add the new balance amount into Balance.txt file.
- Add a message into the TransactionResponse.Data “Your Amount is deposited into the account successfully.”.

Client:

Write a client program to connect the client into the server and do the following

1. No Duplicate Entry

Request1: Call the GetBalance() method by adding TransId into the request.
Request2: Wait for a second and call the GetBalance()method again by adding a different TransId into the request.

Expected Output:
Response1: Print the balance amount on the console.
Response2: Print the balance amount on the console.

2. Duplicate Entry:

Request3: Call the GetBalance() method by adding TransId into the request.
Request4: Wait for a second and call the GetBalance()method again by adding the same TransId into the request.

Expected Output:
Response3: Print the balance amount on the console.
Response4: The transaction has already processed.

3. Deposit the Amount

Request5: Call the DepositeAmount() method by adding TransId and Amount into the Request.
Request6: Call the GetBalance() method by adding a new TransId into the request.

Expected Output:
Response5: Your Amount is deposited into the account successfully.
Response6: Print the balance amount on the console.

4. Server Crash

Request7: Call the DepositeAmount() method by adding TransId and Amount into the Request.

Introduce some delay in the server after updating the balance and before sending the response back to the client. Then stop the server. Start the server again

Request8: Stop the client and start again, this time call the DepositeAmount()method again by adding the same TransId and Amount into the Request.
Request9: Call the GetBalance() method by adding a new TransId into the request.

Expected Output:
Response7: Client will not receive the response

Response8: The transaction has already processed.
Response9: Print the balance amount on the console.

Assumptions:

- At a given time only one client will get connected to the server
- No Authentication is required

Note: You can make any other suitable assumptions while implementing and the same needs to be added in the readme file.

Resources:

client.go
server.go
Balance.txt
Trans_Processed.txt
transaction.proto

Grading – 20 Marks

Client.go – 6 Marks
Server.go – 10 Marks
transaction.proto - 3 Marks
Readme.txt – 1 Marks

Note:

1. Penalty of 10% per day will be issued if the deadline is not met
2. If found copied, you will fetch 0 score.

Submission Details:

1. Please read the questions carefully and complete them.
2. Make a directory with the name <Your_Roll_Number> and copy your all program (source code) and output file to that folder.
3. Implement the solution using GoLang and grpc.
4. Please take a screenshot of the output, name it with its question number and put it in the same folder.
5. Test well before submission. Follow some coding style uniformly. Provide proper comments in your code.
6. Submit only through moodle and well in advance. Any hiccups in the moodle/Internet at the last minute is never acceptable as an excuse for late submission. Submissions through email will be ignored.
7. Please attach a readme.txt file
8. Please zip your folder and submit to moodle within 29-Aug-2021 (23:55 PM)

References:

http://golang.org/doc/
http://www.golang-book.com/books/intro
https://grpc.io/docs/languages/go/basics/
https://towardsdatascience.com/grpc-in-golang-bb40396eb8b1