Advanced Distributed Systems

Assignment 2

Assignment given on: 14th Sept 2020
Due Date: 19th Sept 2020

Assignment 1 – At-Most-Once Semantics

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

Assume you are developing a 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.

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 (l *Listener) GetBalance(args *Request, reply *Response) error
```

Logic to be implemented

- Whenever a request comes from client read the *TransNo* from the Request and check whether the transaction is already processed.
- If it is processed add a message “The transaction already processed.” to the Response(Response.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 *TransNo* into *TransIds[]* array and Trans_Processed.txt file also.

DepositeAmount()

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

Logic to be implemented

- Whenever a request comes from client read the *TransNo* from the Request and check whether the transaction is already processed.
- If it is processed add a message “The transaction already processed.” to the Response(Response.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 Response.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 TransNo into the request.
Request2: Wiat for a second and call the `GetBalance()`method again by adding different TransNo into the request.

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

2. Duplicate Entry:

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

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

3. Deposite the Amount

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

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

4. Server Crash

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

Introduce some delay in 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 now this time call the `DepositeAmount()`method again by adding same TransNo and Amount into the Request.
Request9: Call the `GetBalance()` method by adding new TransNo into the request.

Expected Output:
Response7: Client will not receive the response
Response8: The transaction already processed.
Response9: Print the balance amount on console.

Assumptions:

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

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

Resources:
client.go
server.go
Balance.txt
Trans_Processed.txt

Grading – 20 Marks
Client.go – 9 Marks
Server.go – 10 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 it.
2. Make a directory with name <Your_Roll_Number> and copy your all program (source code) and output file to that folder.
3. Implement the solution using GoLang.
4. Please take screen shot of output, name it with its question number and put it in a 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 20-09-2020 (23:55 PM)

References:
https://medium.com/rungo/building-rpc-remote-procedure-call-network-in-go-5bfebe90f7e9
https://ops.tips/gists/example-go-rpc-client-and-server/