**MP#3: Distributed transactions**

Zhicong Fan(zhicong2), Kehang Chang(kehang2)

05/06/2020

cluster number: g35

Git Revision Number:

Git URL: <https://gitlab.engr.illinois.edu/zhicong2/cs425mps_group35>

**Clients, Branches, and Accounts**

**Introduction:**

In this mp, we implemented client, coordinator and server interface to ensure ACI properties among different banks transactions.

**Running Instruction:**

**We implemented server, coordinator and client in one file so we only have to compile once.**

Please run **Server** on VM1:

go build group35\_mp3.go

./ group35\_mp3 server <node\_port\_number>

server node\_port\_number

A 8000

B 8001

C 8002

D 8003

E 8004

**Or** you can run the script to **launch 5 servers at once**:

sh run\_server.sh

Please run **Coordinator** on VM2:

go build group35\_mp3.go

./ group35\_mp3 coordinator 8000

Run **Clients** on other VMs:

go build group35\_mp3.go

./ group35\_mp3 client <client\_name>

Install Graph Implementation:

go get github.com/twmb/algoimpl/go/graph

* (Notice: export GOPATH=<graph installed directory> might be needed for build group35\_mp3.go)

**Implementation Details:**

**Server:**

The communication among clients, coordinator, servers are hanlded by RPCToServer function. Based on client-sided command, proper server-sided action is invoked by rpcClient.Call function. When there is a write action needed to be perfomed on the client side, client will invoke WriteLockRequest function, then based on the requested account status (number of readers and writers), the server will either return the account information immediately or put the request onto the queue for further notice. As soon as a write permission is granted by UpdateAccount function, server will deliver the messge containg requested account information to fulfill client’s write request. However, a write request is not always granted. The principle of granting permission is as followed:

**There can only be one writer at a time.**

**When there is a writer, there can be no reader.**

**When the reader is the process to request for write permission, when there is no other reader, the request will be granted immediately. The reader will become the writer, there will be no reader during the writing process**

**When there is a reader or writer other than the process requesting for write permission, the requester has to wait until it’s the only reader and there is no other writer or it’s only writer and there is no other reader or writer.**

The read permission request is following the similar procedure, but it’s valid to have multiple readers at the same time as long as there is no writer.

Change function is usually invoked at the commit phase to update server-sided balance based on temp balance from clients. As soon as a commit has been made, UpdateAccount will be invoked to grant more permissions for requests on the queue.

**Coordinator:**

When a new transaction is started (“BEGIN” is entered), coordinator would receive an RPC message from the client which contains the id of this transaction. Then the coordinator would generate a new transaction in its buffer by creating a graph node. When the transaction is finished (“COMMIT” is entered), the coordinator would find the corresponding transaction and delete it from the buffer also the node.

**Client:**

There are six commands in Client:

1. BEGIN:

When the client entered BEGIN in the console, the client would send an RPC message to the coordinator, tell the coordinator that it’s going to generate a new transaction.

1. DEPOSIT

When the client entered DEPOSIT in the console, the client would check if it has already got the write lock in this transaction and if not, it would send RPC request to the server ask for the lock on the user. If the answer is SUCCESS, it would be able to modify the account otherwise it would abort the transaction.

1. BALANCE

When the client entered BALANCE in the console, the client would check if it has already got the read lock in this transaction and if not, it would send RPC request to the server ask for the read lock on the user. If the answer is SUCCESS, it would be able to read the account balance otherwise it would abort the transaction.

1. WITHDRAW

When the client entered WITHDRAW in the console, the client would check if it has already got the write lock in this transaction and if not, it would send RPC request to the server ask for the lock on the user. If the answer is SUCCESS, it would be able to modify the account otherwise it would abort the transaction.

1. COMMIT

When the client entered COMMIT in the console, the client would go through the buffer in the current transaction, if any balance is negative in these accounts, the transaction would be aborted. Otherwise, the client would send RPC message to the server, notifying the changes.

1. ABORT

When the client entered ABORT in the console, the client would directly abort the current transaction.

1. A walk-through of a simple transation that clarifies the roles that the clients, servers, and coordinator (if any) play; i.e., what messages are sent, what state is maintained by which of the nodes, etc.

The client would firstly enter BEGIN, and then the coordinator would generate a node for this transaction. Commands like DEPOSIT, BALANCE and WITHDRAW, would only changes the value inside the temporary buffer inside the client. And during these operations, the clients have to send Read/Write Lock request to the server and wait for their time to execute. Only after COMMIT, the changes would be uploaded the server.

1. An detailed explanation of your concurrency control approach. Explain how and where locks are maintained, when they are acquired, and when they are released. If you are using a lock-free strategy, explain the other data structures (timestamps, dependency lists) used in your implementation.

Since in this mp, locks are only required in DEPOSIT, BALANCE and WITHDRAW, the clients only ask for locks during these three commands. The server has a structure containing a pair of each account and his lock. When the server receives the lock request, it would check if there are other clients holding the read/write locks, and decide whether to grant this lock based on two-phase locking algorithm.

1. A description of how transactions are aborted and their actions are rolled back. Be sure to mention how you ensure that other transactions do not use partial results from aborted transactions.

When a transaction is initialized, all the changes made by the commands will be stored in the temporary buffer which would only be committed after COMMIT is entered. So if the transaction is aborted, we only need to delete the temporary buffer and do not need to roll back since we cannot modify the data on the server directly.

Language used: “Go” for client