COMPSCI 590CC: Cloud Computing **(Due: 10/15/19)**

Programming Assignment #2

Instructor: V. Arun

Name: Jared Nussbaum, Soubhik Rakshit
SPIRE ID: 30393726, 32208881

1 PART 2

In this part we implemented a total order of commands being run across distributed servers using a centralized coordinator based protocol.

In the Constructor we initialize the *this.myID* to be the server's *nodeID*. We also create a coordinator clock called *this.coordinator_counter* with a value of 0, and a local clock called *this.counter* with a value of 0. We also have keep track of what the coordinator node is based on which node in *nodeConfig* has the smallest node address hash then in the case of ties, the smallest port. It is also where we create the *serverMessenger* and the session to connect to the local Cassandra instance and create and connect to keyspace with *myID* as the name of the keyspace. I also instantiate a *PriorityQueue* for storing commands in the order specified by the coordinator and a *HashMap* for keeping track of which headers go with which messages.

When a server gets a message from the client, it checks if it is the coordinator server. If it is, then it stores the header and puts it in the map with a unique (for the specific server) *Integer* id as the key. It then increments the *this.coordinator_counter* counter, and integrates the new counter, request from the client, the key for the header in the map, and the node's *this.myID* into a JSON message that is sent to all nodes in the *nodeConfig*.

If it is not the coordinator server then it stores the header and puts it in the map with a unique (for the specific server) *Integer* id as the key. The client's request then gets packaged into a JSON message that also includes a *COUNTER* value of -1, key for the header in the map, and the server's *this.myID* and sends it only to the coordinator.

When the coordinator gets such a message from another server as signified by the message's counter being -1, it increment's its coordinator clock, stores it in the received as the new *COUNTER* in the JSON and sends the message with the counter as the new coordinator clock instead of -1 to all nodes in *nodeConfig*.

When a message from the coordinator server is received by a server as is know by having a *COUNTER¿-1*, then it adds the message to the priority queue that puts JSONs with lower *COUNTER* values towards the head.

In a loop, it then checks if the JSON at the head of the queue is the next message it was expecting from the coordinator by seeing if the message's counter would match the local clock if the local clock was incremented. If it wouldn't match then it just continues. Otherwise it pops the JSON and executes the request. If it was the original server that sent the request, the node ID in the request's JSON should match *this.myID*. If it is the original server, then it retrieves the header that matches the message's header id from the *HashMap* and sends the default response back to the originating client using the request and the header.

This will run in a total order because the coordinator creates the serialization of events locally and the algorithm ensures that servers will store requests that arrive outside of that serialization that the coordinator specifies. Servers will only run the next expected request in the serialized order and will do so when it has such a request and has already ran preceding requests in the serialization. They also only respond to the client only when completing the request that it received from the client.