CSE 586 PA4 design document

Ubit:qiantao; number :50061614

“N”---- quorum size

“R”----read quorum size

“W”---write quorum size

Insert(update) <key,value> pair

There are 3 steps:

I. Client->Coordinator
The client compute the location in the ring according to the hash value of key. Then client tell the designated coordinator to insert the <key value> pair. However, it is possible that the desginated coordinator fail, in this case, the client will take the successor of the designated as the coordinator.

II. Coordinator->Successor
The coordinator take charge of replicating data over sucessive “N”.It also take charge of votes.
The cordinator take charge of allocating version number for every key. When insert a key, the allocator first search its local storage, if not exist, allocate the version number as 0,write the <key, version,value> to local storage; if already exist, allocate new version number as (old version number +1), then update the <key, version,value> to local storage. At last, the coordiantor tell all its successors to insert <key, version,value> to their local storage. Coordinator need to know if every successor is accessible and if every successorcan successfully write the <key,version,value> to its local storage. Coordinator need to ensure at least “W” copies(including the coordinator itself) are made, if not, coordinator return “Fail” to Client.

III. Every successor accept write command from coordinator, and write to its local storage

Query a specific key

I. Client->Coordinator
The client compute the location in the ring according to the hash value of key. Then client tell the designated coordinator to query the key. However, it is possible that the desginated coordinator fail, in this case, the client will take the successor of the designated as the coordinator.

II. Coordinator->Successor
The coordinator take charge of votes.When query a key, the allocator first search its local storage, if exist,get the version number. Then, the coordiantor tell all its successors to query the specific key. Coordinator need to know if every successor is accessible and if every successorcan successfully can return <key,version,value> to the

coordinator. Coordinator need to ensure at least “R” copies(including the coordinator itself) return <key,verion,value> triple, if not, coordinator return “Fail” to Client. At last, the coordinator select the maximum version number from these “R” results, and it is the newest copy, and return the <key,value> to client.

III. Every successor accept query command from coordinator, and query its local storage, returen <key,version,value> triple to coordinator.

Consistency

Quorum “R” +”W” >”N” ensure the the read operation always can return the newest written copy.

A node recover from failure

When one node recovers from failure, it will contact with its “N” successors, let them pass back all <key,version,value> triples to it. The coordinator then vote from these <key,version,value> triples, get the newest version for every key, then insert(if exist, update) the newest version to the node ’s local storage. The above operations ensure the node not miss any update of every key belong to its partion.

Some programming trickies in this assignment

Tricky 1:ContentProvider.oncreate() can’t call blocking functions, so should fork another asynchronous thread to connect to socket.

Tricky 2:The original design is the code depend on socket timeout to judge if the peer process fail. However, this programming assignment run in simulating enviornment and the port 11108 still is alive even the “SimpleDynamo” on “avd0” is killed. The other process still can connect port 11108 and write to this port. One way to judge if “SimpleDynamo” on “avd0” is dead, is that “readLine()” from the socket connected to port 11108 return null. If SimpleDynamo” on “avd0” is alive, “readLine()” from the socket is blocked or read some value. This tricky is applied in method `insert2coordinator, coordinator_recv_query, query2coordinator, process_recovery`.

Is this implementation high efficient?

First my implmentation is thread-safe, it has no global variables,so the methods are reentrant.

Second, Coordinator’s server thread accept “write” and “query” commands from clients, and relay these commands to successors serially, and wait the results from successors and then do the vote, at last it return the voted result to the client. Doubtless, this implementation is low-efficient, the better way should be the Coordinator’s server thread fork another new thread when its handle “write” and “query” commands, moreover, it should “relay” these commands to sucessor parallelly. I am not familiar with Andorid multitask programming, so I delay this high efficient implementation to the future.