Assignment 5: Design Document for Robust Distributed Key-Value Storage

Purpose of the Assignment/System:

This assignment is yet again the extension of the previous assignment implemented by our team. The purpose of this assignment is to implement functionality to increase the robustness and additional functionality of the distributed key-value storage system. There are three goals to achieve as 'functionality that was required for previous assignment', 'ability for the ring to handle the accidental leaving of any node' and 'ability to perform different queries as first key/last key, node numbers for all the actual nodes on the ring'.

Assumptions:

1. The current system has all the requirements mentioned in the previous design document.
2. An actual node simulates all virtual nodes which are before the node id of its successor.
3. When each node joins the ring or announces its intention to leave the ring, we assumed that the ring is stable. Here stable means that no actual node is currently entering or leaving the ring and the ring is not currently rebalancing, there will not occur situations where multiple nodes enter the ring simultaneously, leave the ring simultaneously, or enter and leave simultaneously.
4. The ring can only change by one actual node at a time, and that change is always completely finished before a new change starts. For this to maintain, we are using 'Wait-time' for ring to balance.
5. While the ring rebalances itself for the join and leaving, message exchanges or client queries are not getting affected.

Changes in the implementation of previous assignment:

We have changed the implementation according to the comments given.

Join a new actual node to the ring:

Algorithm

Initially all the nodes in the ring have token as 'ACTIVE'. A new actual node A asks virtual node B (who is already in the ring) to join in the ring. Whole process consists of different steps: first, A requests join to B, Second, B stops the processing and make its 'ACTIVE' status as false. Here, the thread will run 'CheckforNodeRebound' which is to check that the new virtual nodes simulated by the caller's predecessor has been bounded. Third, this checks the new virtual node Id is supposed to be simulated at last. Once it confirms the new virtual nodes are bound, and finally it executes the computation to remove old virtual node B.
In this process, A gets a return value Map<String, Collection<String>> (all required data the actual node needs to startup itself and simulates its virtual node)

Backup Mechanism:

When the client requests any virtual node and the value found by ring is with the node whose active node is broken. Then the backup mechanism gets activated. In this case, the predecessor active node has to take the backup for its successor active node and its respective virtual nodes. The backup data is transferred (using BACKUPREQUEST) to the active node through message passing in clock-wise direction. And finally, the active node who has the back-up data will give the data to the node to which client queried for.

If B is an actual node,then

"Declined"

Else if B is a virtual node.

B let its actual node C know the request from A. C first halts all the message process in the requested virtual nodes and change the alive token to "DEAD", then grab all the data from the ID of the virtual node B to the end of the virtual node list with a generated process code using MAP<String, Collection<String>>, then pass all to the virtual node B, and B pass it to A.

If the "DEAD" virtual nodes are sending the messages to other nodes, there is no problem because it simply send all the message out again to the final destination again using lookup. However if the "DEAD" virtual node is the node queried by a client, and the client waits on the methods in the "DEAD" virtual node, we need to make sure that it still maintains the communication between the client and the "DEAD" virtual node until it returns the result to the client.

Then the actual node (B or C) send out messages to its successor actual nodes to update their actual node list with the new actual node id just added to the ring.

Leaving an existing actual node from the ring normally

We are testing the leaving of an actual node functionality by invoking remove () functionality by a client for testing purpose.

Algorithm

If A is an actual node.

Find A's predecessor B

If finds,

Node A stops all the message processes in A and also its virtual nodes stops the message processes. Next turns the alive tokens of itself and its virtual nodes "DEAD"

Run thread 'CheckforNodeRebound' and check for the ACTIVE status of the node using isActive(). Then pass to the successor B message. Pass all the data in A and its virtual nodes (2D array as same as one used in Join) to node B using QueryType. VNODE_DATA

If the B gets the message is true, then

Resume all the message process in A and empty all blocking queues in this node and its virtual nodes as they all are simulated by B. Next, send update successor message to other successors, then A can now leave the ring and then A left the system using System. Exit (). At this moment, the client will get Remote exception because the node which was communicating with the client left.

Else, then

Return "Failed to leave."

Else

Returns "This node is the only actual node in ring, cannot leave."

Else if A is a virtual node.

Return "This node is a virtual node, cannot leave."

Fault-Tolerance Implementation

Node leaving the ring without notice

This is the part of the assignment which deals with the fault-tolerance of the system. It means ability for the actual nodes to leave the ring without notice, i.e. the ability for the ring to correctly detect and repair itself. Here we assumed that while the client stores the data in a node, if it is not completely stored (which means that it did not finish a back-up process), then returns 'fail' to the client. If the system succeeds in storing clients' data, though the actual node accidentally left, the completely stored data will not be lost.

Following is the process:

a. *Storing Process:*

When the 'store query' gets the exact node A which should process the store query: Node A stores the message in the node. Then, instead of sending result back to the original departure of the message, change the final destination node of the message as the predecessor actual node B with message Type as MessageType.BACKUP and send message again to B.

When the backup message got to the predecessor actual node B, store it to the Map<Integer, TreeSet<String>> with <node id, data set> pair. Then send a return message (OK) to the original departure of the message (a client is waiting).

b. *Detecting a node failure and restore process:*

Accidentally leaving/ broken nodes can be detected by the system in case that when a client queries a key to any alive node in the system, and the value of the key is stored or supposed to be stored in the node which is down.

If the client queries directly to the broken/ left node: The system will throw a 'remote Exception' to the client. Then the client can query again another nodes in the ring, and eventually it will query on an alive node (Progress).

When a node A gets Remote exception when it invokes sendMessage () method to send message, it puts the thread wait, then do below:

If the node is a virtual node, then

Let its actual node B knows about it. Then B send a message to an actual node C which is the predecessor of the dead actual node using message passing around the ring.

Else,

Node C sends a BACKUPREQUEST to an actual node A which is the predecessor of the dead actual node. The node A sends the data using BACUP_ALL_IN_PRED.

When C gets the message from A or B, it then it simulates virtual nodes which will take over the positions of dead actual and virtual nodes with the backup data with "ALIVE" token, then send message back to the A or B actual node "The node is up!"

When node A or node B gets the message back from C, it wakes up the thread waiting on the node A, then resends the message.

Additional Functionality:

– Determine the first key (in lexicographic order, as determined by the method String.compareTo()) for which a value is stored in the ring.
– Determine the last key (in lexicographic order, as determined by the method String.compareTo()) for which a value is stored in the ring.
Implementation:
For example: m=3, total number of nodes =8
Actual nodes: 1, 4, 6
Client ask for first key to the actual node 1(value)
Actual node 1 will ask its virtual node and compare the keys in alphabetical order, then send the message to the next actual node 4 (it will ask the first keys to its virtual node) and compare in alphabetical order. This process will be followed for all actual nodes in the ring. When node 6 returns its first key to node 1, Node 1 re-calculates the first key if he gets any client request in the meanwhile. (Crossing the snapshot line) And finally, node 1 will return the first key comparing all others keys at that moment to the client.
Similar process we can follow for determining last Key.

– Count the number of keys for which values are stored in the ring.
Implementation:
Again, considering the above example, actual node 1 will pass its own + virtual nodes keys number to the next actual node 4. Repeat the process for all the actual nodes and finally get the count of number of keys for which values are stored in the ring.

– Obtain the node numbers for all the actual nodes in the ring (without using the RMI calls).
Implementation:
Every node will send a token as a message to the next node, when the actual node encounters, add the actual_node_id to the list. Again, send the token all around the ring and get the final list of all actual_node_ids. In above example, the list will contain, node ids: 1, 4 ,6

README file Instructions

How to run a system

First you need **to run RemoteRegistry with port number (8005) and registry name as (RemoteRegistry)** (This specific values user has to give to test the clients storing/ retrieving Data or Node failures using different test cases).

Console output:

```
RMI registry listening on port 8005
Remote registry bound with name 'RemoteRegistry'
```

Then set up the config.properties file. Everything is commented in the config.properties file. The most important properties in the properties file would be buildKVS and nodeid. But also one needs to take care of the WAIT_TIME in ActualKVSNode.

The wait time for the first actual node to wait other actual nodes joining in the ring is defined in ActualKVSNodeImpl WAIT_TIME.

After you set up the config.properties file, execute below commands.

ant compile

ant kvsnode

When you want to run a kvsnode as a first actual node in the ring before the KVS system up, you need to setup the file (m=<any m which will generate 2^m node>, buildKVS=y, rmihost=<host name>, rmiport=<rmi port number>, nodeid=<the node id you want, which is nodeid < 2^m>).

PART 1: FAULT-TOLERANCE

Case 1 (Each test case done on local machine)

1) Start up a single actual node 0 with m = 8 \\Success
2) Add actual node 101. \\Success
3) Attempt to add actual node 201
4) Attempt to add actual node 257 \\ Error message, expected
5) Attempt to add actual node 101 again \\Error message, expected
6) Storing Data for 256 Nodes (Using KVSClientStoreTest.java) \\Success
7) Remove actual node 201 (Normal /non-crash way) (using KVSClientRemoveNodeTest.java)
8) Retrieve Data (Using KVSClientRetrieveTest.java)
9) Crash actual node 101 by manually killing terminal window
10) Another client (from Cluster server will query to Node id 50) \\Success
11) Attempt to retrieve Data (Using KVSClientRetrieveTest.java)
12) Attempt to kill node 0 (normal way) (using KVSClientRemoveNodeTest.java) \\error message

Case 2: (Each test case done on cluster)

The above all test cases we have performed on cluster machines and there was success in the expected output results. –Successfully performed providing all correct output

PART 2: ADDITIONAL FUNCTIONALITY

(1) Start up the system with m= 8, actual node as 0, 101, 201
(2) Store Data (Using KVSClientStoreTest.java)
(3) Attempt to determine the first key whose value is stored in the ring (Using KVSClientComputationQuery.java)
(4) Attempt to determine the last key whose value is stored in the ring (Using KVSClientComputationQuery.java)
(5) Attempt to count the number of keys (Using KVSClientComputationQuery.java)
(6) Obtain the list of actual node Ids (Using KVSClientComputationQuery.java)

PART 1: FAULT-TOLERANCE

For example:

1) Start up a single actual node 0 with m = 8

Config.properties set up: m=8,

buildKVS= y
rmihost =localhost
rmiport = 8005
nodeid = 0 (Atcual_Node_id)

The log will have information as:

Log0.txt:

```
2013-06-06 22:54:30.296: Startup Information:
Node ID: 0
"M" value: 8
Total nodes: 256
2013-06-06 22:54:30.299: Nodes with open channels to this node: 1, 2, 4, 8,
16, 32, 64, 128,
2013-06-06 22:54:31.657: Virtual nodes managed by node 0: 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109
110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166
167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185
186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204
205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223
224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242
243 244 245 246 247 248 249 250 251 252 253 254 255
2013-06-06 22:54:33.199: Successor actual nodes of this node 0: None
2013-06-06 22:54:33.205: List of Nodes which data is backed up in this node
0: None
```

2) Add actual node 101.

Next task is we have **to add one more node** after the KVS system up, then we need to change the config.properties as:

Config.properties set up: m=8,

buildKVS= n
rmihost =localhost
rmiport = 8005
nodeid = 101 (actual_node_id)

The log will have information as:

Log101.txt:

```
2013-06-06 22:58:20.686: Startup Information:
Node ID: 101
"M" value: 8
Total nodes: 256
2013-06-06 22:58:20.690: Nodes with open channels to this node: 102, 103,
105, 109, 117, 133, 165, 229,
```

```
2013-06-06 22:58:20.694: Virtual nodes managed by node 101: 102 103 104 105
106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124
125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181
182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219
220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238
239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
2013-06-06 22:58:20.697: Successor actual nodes of this node 101: 0
2013-06-06 22:58:20.712: List of Nodes which data is backed up in this node
101: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
2013-06-06 22:58:22.230: Node 101: Received a message (VNODES_DATA, null)
from a node 100
2013-06-06 22:58:22.846: Successor actual nodes of this node 101: 0
```

When we add node 101 as the actual node, the previous log0.txt also gets updated because the virtual nodes of the respective actual nodes gets divided. Hence, the updated log0.txt is:
Updated log0.txt:

```
2013-06-06 22:57:50.788: Startup Information:
Node ID: 0
"M" value: 8
Total nodes: 256
2013-06-06 22:57:50.791: Nodes with open channels to this node: 1, 2, 4, 8,
16, 32, 64, 128,
2013-06-06 22:57:52.130: Virtual nodes managed by node 0: 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109
110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166
167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185
186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204
205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223
224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242
243 244 245 246 247 248 249 250 251 252 253 254 255
2013-06-06 22:57:53.616: Successor actual nodes of this node 0: None
2013-06-06 22:57:53.621: List of Nodes which data is backed up in this node
0: None
2013-06-06 22:58:21.653: List of Nodes which data is backed up in this node
0: 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137
138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156
157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175
176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194
195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213
214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232
233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
252 253 254 255
```

2013-06-06 22:58:21.661: Virtual nodes managed by node 0: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
2013-06-06 22:58:22.187: Successor actual nodes of this node 0: 101 **(Updated logs)**
2013-06-06 22:58:22.190: Successor actual nodes of this node 0: 101
2013-06-06 22:58:22.201: Node 64: Received a message (VNODES_DATA, null) from a node 0
2013-06-06 22:58:22.207: Node 64: Sent a message (VNODES_DATA, null) to a node 96
2013-06-06 22:58:22.210: Node 96: Received a message (VNODES_DATA, null) from a node 64
2013-06-06 22:58:22.215: Node 96: Sent a message (VNODES_DATA, null) to a node 100
2013-06-06 22:58:22.217: Node 100: Received a message (VNODES_DATA, null) from a node 96
2013-06-06 22:58:22.227: Node 100: Sent a message (VNODES_DATA, null) to a node 101

3) Attempt to add actual node 201

Log201.txt:

2013-06-06 23:04:35.040: Startup Information:
Node ID: 201
"M" value: 8
Total nodes: 256
2013-06-06 23:04:35.044: Nodes with open channels to this node: 202, 203, 205, 209, 217, 233, 137, 73,
2013-06-06 23:04:35.047: Virtual nodes managed by node 201: 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
2013-06-06 23:04:35.049: Successor actual nodes of this node 201: 0 101
2013-06-06 23:04:35.064: List of Nodes which data is backed up in this node 201: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
2013-06-06 23:04:36.484: Node 229: Received a message (VNODES_DATA, null) from a node 101
2013-06-06 23:04:36.489: Node 201: Received a message (VNODES_DATA, null) from a node 197
2013-06-06 23:04:36.497: Node 229: Sent a message (VNODES_DATA, null) to a node 245
2013-06-06 23:04:36.501: Node 245: Received a message (VNODES_DATA, null) from a node 229
2013-06-06 23:04:36.504: Node 229: Received a message (BACKUP_ALL_IN_PRED, null) from a node 101
2013-06-06 23:04:36.509: Node 245: Sent a message (VNODES_DATA, null) to a node 253
2013-06-06 23:04:36.513: Node 253: Received a message (VNODES_DATA, null) from a node 245
2013-06-06 23:04:36.518: Node 245: Received a message (BACKUP_ALL_IN_PRED, null) from a node 229

```
2013-06-06 23:04:36.522: Node 229: Sent a message (BACKUP_ALL_IN_PRED, null)
to a node 245
2013-06-06 23:04:36.525: Node 255: Received a message (VNODES_DATA, null)
from a node 253
2013-06-06 23:04:36.528: Node 253: Sent a message (VNODES_DATA, null) to a
node 255
2013-06-06 23:04:36.533: Node 253: Received a message (BACKUP_ALL_IN_PRED,
null) from a node 245
2013-06-06 23:04:36.537: Node 245: Sent a message (BACKUP_ALL_IN_PRED, null)
to a node 253
2013-06-06 23:04:36.541: Node 255: Sent a message (VNODES_DATA, null) to a
node 0
2013-06-06 23:04:36.545: Node 255: Received a message (BACKUP_ALL_IN_PRED,
null) from a node 253
2013-06-06 23:04:36.550: Node 253: Sent a message (BACKUP_ALL_IN_PRED, null)
to a node 255
2013-06-06 23:04:36.555: Node 255: Sent a message (BACKUP_ALL_IN_PRED, null)
to a node 0
2013-06-06 23:04:37.165: Successor actual nodes of this node 201: 0 101
```

In this way, we can add any number of actual nodes in the system (less than total number of nodes) and the system gives the responsibility of virtual nodes with proper division along with the backup data stored.

4) Attempt to add actual node 257:

Console Error message:

```
2013-06-06 23:07:28.821: The KVS node got a remote registry located in host:
localhost and port: 8005 'RemoteRegistry'
2013-06-06 23:07:28.822: Invalid m or node id value in config file. Exiting
program.
```

5) Attempt to add actual node 101 again

Console Error Message:

```
2013-06-06 23:08:37.099: The KVS node got a remote registry located in host:
localhost and port: 8005 'RemoteRegistry'
2013-06-06 23:08:37.115: Okay, the node is up. You can test the system now.
2013-06-06 23:08:37.115: The claimed node id is occupied by an actual node.
You cannot take over an actual node. Try other nodes.
```

6) Storing Data for 256 Nodes

(Using KVSClientStoreTest.java)

Chunk of Data on Console:

```
2013-06-06 23:11:32.797: Hashvalue 95
2013-06-06 23:11:32.932: Test253, Result: STORED
2013-06-06 23:11:32.935: Hashvalue 96
2013-06-06 23:11:33.070: Test254, Result: STORED
2013-06-06 23:11:33.073: Hashvalue 97
```

In this way, the data will be successfully stored for all the values from 0 to 255 nodes.

7) Remove actual node 201 (Normal /non-crash way)

(Using KVSClientRemoveNodeTest.java)
Chunk of Data from Log201 file:

```
2013-06-06 23:11:33.150: Node 201: Backup a message (IT255) with result
(SUCCESSFUL)
2013-06-06 23:11:41.077: Node 255: Stored a message (IT255) with result
(Request failed) then send it to predecessor to backup
2013-06-06 23:14:41.638: Node number 201 has left safely. Changed into
virtual node. Managed by actual node 101

The log file of 101 node gets updated:

013-06-06 23:14:42.079: Node 101: Received a message (VNODES_DATA, null) from
a node 97
2013-06-06 23:14:42.345: Successor actual nodes of this node 101: 0
```

8) Retrieve Data

(Using KVSClientRetrieveTest.java)
Data retrieved successfully

9) Crash actual node 101 by manually killing terminal window

Console Output:

```
2013-06-06 23:20:26.114: Finding a predecessor which will fixing the system.
Failure node id 101
2013-06-06 23:20:26.114: Okay, actual node 0 will fix this problem.
2013-06-06 23:20:26.114: Fixing has been started in node 0
2013-06-06 23:20:26.114: Removed successor 101 from this node 0
```

10) Another client (from Cluster server will query to Node id 100)

We have checked this functionality using Clusters and it is running successfully.

11) Attempt to retrieve Data

(Using KVSClientRetrieveTest.java)
We are able to store and retrieve data after rebalancing.

12) Attempt to kill last actual node 0 (normal way)

(Using KVSClientRemoveNodeTest.java)
Console output:

```
It's been connected to the RMI Registry
This node is only actual node, cannot leave.
```

PART 2: ADDITIONAL FUNCTIONALITY

(1) Start up the system with m= 8, actual node as 0, 101, 201

Successfully started up the system (as above logs given)

(2) Store Data

(Using KVSClientStoreTest.java)
Logs provided above. We have to store the data to compute first/last key, total number of keys.

(3) Attempt to determine the first key whose value is stored in the ring
(Using KVSClientComputationQuery.java)

```
It's been connected to the RMI Registry
First Key is: Test0
```

(4) Attempt to determine the last key whose value is stored in the ring
(Using KVSClientComputationQuery.java)

```
It's been connected to the RMI Registry
Last Key is: Test99
```

(5) Attempt to count the number of keys (Using KVSClientComputationQuery.java)

```
It's been connected to the RMI Registry
Number of keys: 256
```

(6) Obtain the list of actual node Ids (Using KVSClientComputationQuery.java)

```
Actual nodes are: 0
Actual nodes are: 101
Actual nodes are: 201
```

All the updated log files are already provided in SVN with the latest revision number with same actual node IDs.

Basic Test Cases: (apart from above Batch testing)

Sr.	Test Case Description	Provided Input	Expected Result	Obtained Result
1	Client providing wrong input in config.properties file	m value: Host Name: Port Number: actual NodeID:		
2	Requests from clients to a node at the same time			
3	RMI registered on wrong port Number	RMI port Number: negative value / Non-number value		
4	Actual Node not started with specific ID			
4	Actual Nodes not connected to neighbors properly	Log-Id.txt file	Successor list for node IDs	

5	Log File generated with specific information / not			
	Adding a Node			
6	When a ring is started, adding a node at particular position: 1) Position of Virtual Node 2) Position of Actual Node	Log-Id.txt file generated	Actual node should not get added at the positon of actual node. It should get added to the position of Virtual Node	
7	Adding a new actual node	Ring set up with 'n' no. of actual nodes, and now new actaul node added	Because of new actaul node, the virtual nodes should be divided properly	
8	Log-file updated	Ring set up with 'n' no. of actual nodes, and now new actaul node added	The log files should be updated accordingly	
9	Actual Nodes created at different Servers	Ring set up with 'n' no. of actual nodes, and now new actaul node added	The log files at different servers should be updated accordingly	
	Leaving a Node			
10	Actual node leaving informing its neighbors	Actual Node Id 'n' decided to leave	Its responsible virtual nodes should be given to predessor Actual Node	
	Client Communicating Ring System			
11	Client provides key ='text' and node id='number', particular node should return a value back to the client			

12	**Batch Testing:** First use StoreData, second add/ remove a node and then finally use retrieveData, the respective logs should get generated.			

Conclusion

Now the system is ready with primarily all required functionalities. The next task given is to make the robust system for fault-tolerance and also set up few with additional functionalities.

As we have gotten many comments in previous assignment, the challenge we accepted to implement the functionalities to add/ remove node normally. Also getting the backup in nodes and restoring then whenever required was complete new learning for us. The issues we have encountered are mentioned in executive summary.

References

(1) Assignment 5 description given by Prof. Daniel Zimmerman

(2) Assignment 4 design Document