Assignment 4: KVSdesign _558team5 Design Document for Distributed Key-Value Storage

Purpose of the Assignment/System:

This assignment is the extension of the previous design created by our team. The purpose of this assignment is to implement a distributed key-value pair storage system with the help of class skeleton prepared. There are three goals to achieve as 'functionality that was required for previous assignment', 'ability for the actual nodes to join an existing ring' and 'ability for actual nodes to leave the ring in an orderly fashion by announcing their intention to neighbors'.

Assumptions:

1. The current system has all the requirements mentioned in the previous design document.
2. 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.
3. 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.

Changes in High-Level Design of previous assignment:

When we started implementing on the available high-level design, we have found out few changes. Some of them are incorporated according to the comments given for assignment 3. These are described below:

1. We will likely not be able to use actual Futures as declared in Java, because they are not remote objects and ours will have to be remote objects. So instead of 'Future', we implemented using a busy waiting mechanism in Store Data and Retrieve Data methods of AbstractKVSNode class.
2. Generating a simple test Client class KVSClientTest.
3. The Object in RemoteMessage class must actually be Serializable, so the message is made as String Object so as to be Serializable. This object will be moved around the ring for communication.
4. According changes made in 'Log' information so as to make the logging information more readable.

Dynamic System Changes Implementation

Communication between Clients and Nodes

This is the heart of the assignment in which the nodes in the ring communicate within themselves and the client can communicate with the nodes in the ring.

1. **Client Communicate with nodes in the ring: Synchronous communication.**

Client can retrieve and store data synchronously using retrieveData(String) and storeData(String) respective methods in KVSNode. When a client request a retrieve query using retrieveData to a KVS node, the node hashes the string value using Hasher class, and gets the hash value (0 <= hash value < 2^m), then there are two conditions:

a. If the hash value is the same as the node id, it the retrieve query is processed locally, and returns value (If the value is in database, returns "OK", if not "NO").
b. If the hash value is not the same as the node id, it follows below process:

   Give a message ID to the string and put it in to my_sent_message table (Map) -> send message to the final destination node using the shortest path -> Wait until the result for the sent query comes back -> If the return value comes back within a given time (5 sec), returns the return value, if not, returns "Request Failed".

The node generates a message ID with getMessagId in AbstractKVSNode for the string and store it to my_sent_message table (Map) with Result object which has a boolean flag indicating the result comes back or not, a boolean flag indicating the result is passed to the client or not (to overwrite a new message ID when the ID is useless (the result is already passed to the client) and result string.
Then sends out the string to other node which should deal with the string using sendMessage(source node id, destination port id, RemoteMessage). The string message is wrapped in RemoteMessage which has information of original departure node, final destination node, return or non-return value flag, query type, message id and the string message. The shorted path is determined by RoutTracker in util package. Next is to put the client request "wait" for certain amount of time (5 sec) using loop (busy waiting). If the result comes back within the given time, it returns the result, if not, it returns "Request Failed" to the client.

2. **Nodes communicate within the ring: Asynchronous Communication**

Nodes communicate with sendMessage (source node id, destination port id, RemoteMessage) method asynchronously. The string message is wrapped in RemoteMessage which has information of original departure node, final destination node, return or non-return value flag, query type, message id and the string message.
When a node gets a remote message from other nodes, it follow below process.

Put the message to MessageProcessorForEachChannel Runnable thread and pass it to the relevant BlockingQueue indicating a communication channel between two nodes -> All the BlockingQueues are merged into my_transfer_queue using MessageProcessorForConvergingChannel Runnable. -> By each MessageProcessorForConvergingChannel, each message is processed sequentially by one thread.

The main job of MessageProcessorForEachChannel is just pass the messages from a specific node (defined by the previous assignment) to the converging channel. Actually this process is

not critical functionality for this application (we can directly put all the messages from all the nodes to the converging channel without any problem), but this process is needed to mimic "real" system, and makes more sense. The size of the BlockingQueues are defined in KVSNode as a static final value. All the BlockingQueues for every channel is stored in my_port_queues. The messages from each BlockingQueue are merged into Converging BlockingQueue (TransferQueue). From JavaDoc 7, a TransferQueue may be useful for example in message passing applications in which producers sometimes (using method transfer(E)) await receipt of elements by consumers invoking take or poll, while at other times enqueue elements (via method put) without waiting for receipt.
MessageProcessorForConvergingChannel mainly does one of the below three jobs depending on the conditions.

a. First, process a result from other node which was originally sent out from the node processing the MessageProcessorForConvergingChannle.
b. Second, process a query message which should be dealt in the node. If the final destination of the message is the same as the node id, then it process the message.
c. Third, it passes the message again to other node to let the message get the final destination node. If the final destination of the message is not the node, then it passes again to other node.

Rebalancing: The rebalancing is possible by two means: Adding a Node, Leaving a Node.

1. *Adding a node*: A new actual node can join the system at any time when the system has less than 2^m actual nodes. The add node method will cause the new actual node to take over all the virtual nodes its position in the ring to the next actual nodes position. If the requested spot is already an actual node, it cancels the addition. If there are no virtual nodes for the new actual node to take over as described in the method above, a new actual node which handles no virtual nodes will be created in the requested spot. The virtual nodes taken over will pass their current data storage in the form of a tree set using getDataSet() and setDataSet(Set<String>). These methods get the data and set the data for a virtual node, respectively. After any sort of (hostile or un-hostile) takeover over virtual nodes, these actions will be logged in the two affected actual nodes log, the already existing actual node that is having its virtual nodes taken over and the new actual node. The implementation is well documented, so might be reasonable to understand.

2. *Leaving a node:* The node leaving with orderly fashion: When the actual node tries to leave by its own will, then it should give all its responsible nodes' information to its neighbor actual node, or the next actual node in the ring The method removeActualNode() handles removal of any actual nodes. This method turns the actual node requesting to leave into a virtual node and moves it as well as its previously owned virtual nodes to an existing actual node (the next one in the ring heading clockwise). If that actual node leaving is the only one in the ring, the removal is cancelled and the actual node that attempted to escape is severely reprimanded for its insolence (in its log). Otherwise any actual node leaving is logged in its respective log. Again the implementation is well documented, so might be reasonable to understand.

README file Instructions

How to run a system

Before 'run ant' command to build the project and run, you need to 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, isfirstnode, 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, is firstnode=y, rmihost=<host name>, rmiport=<rmi port number>, nodeid=<the node id you want, which is nodeid < 2^m>).

For example:

Config.properties set up: m=3,

buildKVS= y

isfirstnode = y

rmihost =localhost

rmiport = 8005

nodeid = 2

The log will have information as:

```
log2-txt file:2013-05-26 17:37:08.998: Startup Information:
Node ID: 2 (Actual Node Id provided)
"M" value: 3
Total nodes: 8
2013-05-26 17:37:08.998: Nodes with open channels to node 2: 3, 4, 6,
(Connection to the actual node using modulo formula)
2013-05-26 17:37:39.048: Successors: None
```

1. 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=3,

   buildKVS= n

   isfirstnode = n

   rmihost =localhost

   rmiport = 8005

   nodeid = 5

   The log will have information as:

   log-5.txt:
   ```
   2013-05-26 17:45:04.206: Startup Information:
   Node ID: 5
   "M" value: 3
   Total nodes: 8
   ```

```
2013-05-26 17:45:04.211: Nodes with open channels to node 5: 6, 7, 1,
2013-05-26 17:45:22.304: Successors: 2
2013-05-26 17:45:22.369: Virtual nodes managed by node 5: 6 7 0 1
```

When we add node 5 as the actual node, the previous log2-txt also gets updated because the virtual nodes of the respective actual nodes gets divided. Hence, is shows the message as 'Reloggin actual node virtual nodes.'

```
Log2-txt: 2013-05-26 17:37:08.998: Startup Information:
Node ID: 2
"M" value: 3
Total nodes: 8
2013-05-26 17:37:08.998: Nodes with open channels to this node: 3, 4,
6,
2013-05-26 17:37:39.048: Successors: None
2013-05-26 17:45:04.201: Virtual nodes managed by this node: 6 7 0 1 3
4
2013-05-26 17:45:22.394: Virtual nodes managed by this node: 3 4
(updated log)
```

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.

2. Node leaving the system in orderly fashion: An actual node and corresponding virtual nodes leave the system when the ring is stable. The node will always leave in an expected way. It is assumed that while a node is leaving, no additional actual nodes will be added, or removed. When the node leaves, all its virtual nodes are taken over by the next actual node in the ring going clockwise. We made an actual node leave when remove () method on the node is invoked by a client. That was the easy way to test the leaving part we can come up with.
   We test it using KVSClientRemoveNodeTest. When the method is invoked, the actual node pass all the datasets in the actual node and the virtual nodes the actual node has been simulating to the predecessor actual node, then rebind the positions in RMI registry with the new actual and virtual nodes, then empty the BlockingQueue in the leaving nodes, then leave. We assumed that the rebind command simply change the reference of the Remote object, so it happens without any noticeable interval, so exchanging messages in the ring would not be affected by the rebinding.
3. A Batch of storing and retrieving data while a node is leaving and adding to the ring
   A batch (100) of storing data was performed by KVSClientStoreTet.java, and while it stores the data in to the ring, we added and removed a node from the ring. Then we retrieved the same data from the ring using KVSClientRetrieveTest.java, and we found that all the hundred Keys were successfully retrieved from the ring. By this test, we were able to know our system was pretty robust with respect to adding and leaving a node situation. Also using 'KVSTestClient' we have tested the correct value is retrieving for particular key.

Test Cases:

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

Yet again an interesting team assignment to re-design and implement a distributed key-value storage System. As we scratched the idea of 'Future' after lot reading and understanding the system, it was challenging to replace the fundamentals of communication. So Sky had come up with idle wait mechanism using interrupt which is a complete new learning for all of us. Also implementing many new

concepts as TransferQueue, BlockingQueue, maintaining a message table, message passing and use of RMI registry in detail. We spent lot of time in designing the system, so life was little bit easier. But we have found of different levels of issues while implementing which we will mention in Executive Summary.

References

(1) Assignment 4 description given by Prof. Daniel Zimmerman
(2) Assignment 3 design Document