DHT Report and Instruction Document

Name : Varun Pandey
Student ID : 5108436

Name : BhagavathiDhass Thirunavukarasu
Student ID : 5078702

Instructions on compiling and running the application

Please refer to README.txt

Description of important classes

1. **WordKey** : This class is responsible for capturing the hash values of the word which has to be hashed into the DHT. Note that the “stringForHashKey” field contains the final 160-bit hash value that is used.
2. **NodeKey** : Analogous to WordKey, this class captures the hash values for the actual nodes of the system. Even in this class the final 160-bit hash value for hashing into the chord ring is contained in the “stringForHashKey” field.
3. **WordEntry** : This class captures the word and meaning for any dictionary entry. Objects of this class are stored in the hash tables which are present at the node level of the chord ring.
4. **HashingHelper** : Provides the method to generate the 1600-bit hash value for a byte array. The default hash algorithm is SHA-1
5. **FingerTableEntry** : Objects of this class find their way into the finger tables which are stored at node level in the chord ring. Note that a finger table entry is essentially a way to keep record of the node keys for the node and the structure of the class reflects this.
6. **Node Interface** : This interface extends the Remote interface and contains the declarations of the important API’s which are provided to the client and to other nodes to query the nodes and the ring.
7. **NodeImpl** : Implementation of the node interface. Objects of this class are present on the chord ring and are also used to perform operations on the ring. Nodes are generic in nature, except the node-0, which is the master node that is responsible for talking to the client in most of the cases and also the point of contact for the chord ring. We however did not extend out a special class for node-0. There are however some functions present in the Node Interface which can only be called on the object of node-0. These are

a. *create ()* : called when node-0 is added to the ring. This function essentially creates the ring and places node-0 in it.

b. *join()* : other generic nodes call this function on node-0 object to join the ring defined by the node-0. join() is also responsible for resetting the successor and predecessor relations between the nodes once a new node has been introduced. It also calls redistributeKeys() to move around the set of keys which will now be the responsibility of the newly introduced node.
c. *join_done()* : a synchronization tool, helps avoid conflicts between multiple nodes trying to join the ring at the same time. The “busyInJoin” boolean variable helps control this.

8. The other important functions provided by the Node interface are:

a. *find_node()* : given a word, this function returns the node in the ring that will be responsible for storing this word.
b. *insert()* : once we know the node responsible for storing a word, this function helps store the word into the hashmap at that particular node.
c. *lookup()* : once a word has been inserted, this function helps a client lookup a word in the dictionary.

Class to create new nodes

NodeStart : this is the driver class which is called to launch a new node to join the ring. Depending upon the value of the nodeNum, it branches off to either create the master node node-0 which then initiates the ring or a generic node which looks up the master node to join the ring.

Class with RMI functionality

Ring : All functions which make RMI calls (apart from the client) have been dumped in this class. It contains most of the important functions for implementing chord like creating the ring, locating a node, finding the successor/predecessor of a node, redistributing the keys of the node, etc.

Client Classes :

1. **Client** : This is an interactive test client. It takes the dump of all the words in the text file and inserts it into the chord ring. Once this is done, it keeps asking the user to lookup the meanings of words from the DHT. Note that this client can be run at any state of the ring once the main node has been created (that is the chord has been initialized). Please run this client periodically to check if the meanings of the words in the dictionary can be retrieved from the DHT or not. Helps determine the sanity of the DHT.
2. **TestCasesClient** : Contains the test cases, more about which has been described in the README.txt file. Contains positive and negative test cases which validate the state of the DHT after multiple nodes have joined

3. **GetRingDetailsClient** : This is a very useful client which helps print out the state of the DHT at any instant. It will give the user an idea about how the words are distributed among the various nodes and also the successor/predecessor/finger table info of all the nodes

Important : Since the address space for the chord is pretty large (160-bit) and the number of words relatively small, the distribution can be pretty sparse and sometimes some nodes may not contain any words at all. We suggest that the user run different combinations of node joins to see the different ways in which the keys get distributed once when the nodes are added into the chord!