SIX MONTH WORK IN DISTRIBUTED SYSTEMS Ac. year 2016-2017, 9th Semester, E&M School Final Delivery Date: 10/03/2017

In this task you will design a simple DHT, simplified version of it. It is not necessary to implement the finger tables or routing through them. Also you will not need to handle node failures. The key functions you will perform (a) the division of Ids space (nodes and objects), (b) routing, (c) node insertion, (d) node departure, and (e) data replication.

The application you are developing will be an emulator that will handle multiple nodes, it must implement all the functions of DHT processes, open sockets, respond to incoming requests. It should also implement a simplified version of the member protocol, it should handle node insertion and exit. The requirements for each node are as follows:

1. Each node must implement all functions for communicating with server / client threads / processes, sockets).

2. Each node receives a unique id derived from applying one to its serial number. For example the first node will have ("2") coke.

3. Each node implements the insert (key, value), query (key> pairs where both key and value are stings. The key must pass a hash function before being used for any of the above functions (to find the correct position in the ring) )

4. When insert(key, value) is called with a key already stored in, it removes the old key value and stores the new one.

5. Function query (key) must take care of the special character that should return all <key, value> pairs stored in per node.

6. The node will implement ring routing, following its design. This means that the node will hold pointers to the previous and next node of the logical ring (or logarithmic number of nodes if you want to tables) and forward any query to its next until the request reaches the correct node. Once the correct node receives it processes it and responds directly to the node that started the request. It is repeated that you do not need to route using them, however you are welcome to try

For hashing, use SHA1.

7. Your system should handle new node insertions (join (nodeId)) and node departures (depart (nodeID)). To do this you need to specify a node (obviously your original node ie "1") that will accept all join / depart requests. When inserting a node departure, the affected nodes should properly update the markers in the previous and next nodes necessary for routing messages and redistribute their keys so that each node is responsible for the correct keys. You do not have to deal with simultaneous join / depart - You consider that a node enters or leaves after the previous entry or exit is completed. You do not need to handle insert / delete / query requests simultaneously with join / depart. You believe that these demands come after the system is in balance. You don't have to worry about node failures. We consider nodes to leave the system only voluntarily through the depart procedure.

8. After checking that the basic functions described above are working, you will also import replication of the <key, value> data stored on the system. The replication factor will be a variable k (measurements will be made with different values ​​of k). This means that each <key, value> pair should be stored except for the node responsible for the hash (key) and the k-1 next nodes in the logical ring. Replication should take into account all the basic functions of DHT (insert, delete, query, join, depart).

9. You will implement 2 types of consistency for replicas: (a) linearizability and (b) eventual consistency.

(a) In the case of linearizability, there should be strong guarantees that all replicas always have the same value for each key and that each query will always return the most recent value written. For linearizability you can implement either quorum replication or chain replication.

Quorum replication: In this case, to achieve linearizability, both reader quorum and writer quorum must be equal to k-1 (why?). One node will be the coordinator (who?). The coordinator for any read / write should always communicate with the other k-1 nodes that have replicas to read or write a value. For write operation, all values ​​can have versions to distinguish old from recent copies. For read operations if the nodes in the reader quorum have different versions of the same object, the most recent copy is returned.

Chain replication: In this case, a write always starts from the primary node responsible for a key and moves sequentially to the k-1 residues that have copies. The last node in the row returns the result of the write. A read should instead read the value from the last node in the row.

(b) In the case of eventual consistency the changes will be disseminated lazily to the copies. This means that a write will go to the primary node responsible for that key, and that node will return the result of the write. It will then make sure to send the new value to the next k-1 nodes. A read will read from any node that has a copy of the key it is requesting (with the risk of reading a stale value).