Report 5: Chordy

Mallu Mallu

October 12,2016

1.Introduction

In this homework I have built a distributed hash table using erlang, DHT is a decentralized distributed system that can be used to look up key: value pairs that are stored in a DHT. Also, the implementation followed the chord scheme. In the implementation of DHT we maintained a ring structure for the nodes. Also, in normal language DHT functions like a dictionary of nodes storing the information for key-value pairs. DHT is popular for its fault tolerant capability and considerable performance.

2. Main Problems & Solutions

a) Maintaining the ring structure

In the first implementation we maintained a basic stable ring structure of nodes, first of all key module is created in which I have used erlang random generator to generate a unique value everytime new key is created. Also, between function is implemented to locate the key between *From* and *To*. After key module we created node1 module in which each node will have key, predecessor, successor and messages for notification of new node, query for predecessor and status from the successor node are maintained. Stabilization should be done periodically. Also probe procedure is implemented to check if the ring is connected. Stabilization is done through sending the message {request,self()} to successor and then expecting the reply status for predecessor , after getting the reply the possibilities are its predecessor is nil or it is pointing to itself in both of the cases it should be informed to our existence. But when it points to some other node then ring should be stabilized.

b) Storage and adding new key: value pairs

Local storage for each is implemented in the storage module. The task for this module is to store {key, value}. Also, I have implemented this module using erlang lists library functions. In the module node2 I have implemented the functionality to add new nodes in the ring. This part of assignment was bit complex as I had to implement the look up procedure also for the DHT. Also, the handover is implemented to tackle the responsibility, but this part was complicated in terms of given code, it took

most of time for me to understand. I have verified the working of look up and add functions using test module. Also, in the ring structure for failure detection the implementation has been done through monitoring and the key aspect is to monitor your successors successor. This has been implemented through erlang:monitor/2 procedure which can detect the failure showing down message.

3. Evaluations

I have evaluated different this implementation by starting single node and then adding new nodes in the ring, Also I have tested look up procedure.

Results are as follows:

- **P= node2:start(10).**
  <0.58.0>
- **test: add(4000,8,P).**
  Ok
- **test: look up(4000,P).**
  {4000,10}
- **P ! probe.**
  Nodes: probe
  10 14>
  Time = 23 ms
- **test: check([4000],p).**
  1 lookup operation in 0 ms
  0 lookups failed, 0 caused a timeout
  .

Also as mentioned in the assignment to test four test machines by adding 1000 elements to the ring and then do a lookup of the elements. Then doing a lookup for one machine to handle 4000 elements. Finding are shown below:

Time taken for one machine to lookup 4000 elements	Time taken for four machine to handle 1000 elements
897 ms	297 ms
	201 ms
	140 ms
	46 ms

Average time for four machines to do 1000 lookups is 171 ms

4. Conclusion

In conclusion it can be said that through this assignment I have gained much clear understanding of the DHT concept. Practically quite good results have been observed in terms of performance in the distributed systems which have implemented DHT. One important feature which has been mentioned is the use of finger tables for routing which can help in locating key value pairs in a more efficient manner.