COP 5615 – PROJECT 3

Chord Simulation

Priyanshu Pandey (UFID: 21081358)
Sourav Kumar Parmar (UFID: 86511933)

Instructions to run the program Build system: Linux (Ubuntu):

- Unzip the **PriyanshuPandey_SouravParmar_Project3.zip**.
- Change directory to the following
  Path: *PriyanshuPandey_SouravParmar_Project3/Project3*
- Execute the following command:
  sbt "project project3" "run <numNodes> <numRequests>"

--numNodes = number of peers in the network.
--numRequests = number of key look-up requests sent by each node.
eg: ***sbt "project project3" "run 1000 2"***

Explanation:

Join:

Every time a node wants to join the network, it tells the boot-strap node "refNode" its key value which is calculated by truncating the m bits of sha-1 value of its name which is done by "consistenthash()".

When truncating the sha-1 value, there is a possibility of collisions in the network as "m" bits of sha-1 are not guaranteed to be unique if m < 160. In order to handle this problem in smaller sized network, name of node is appended with a random string which is sent to the sha-1 calculator. If the sha-1 value is found to collide with a key already existing in the network then this process is repeated until we get a unique identifier to the node.

The node which wants to join asks the boot-strap node to find its predecessor and successor in the network. It then updates its successor and successor fields and send the message to its predecessor and successor to update their respective fields with the newly added node.

The newly added node then node update its finger table and finger table of all other nodes whose finger table needs to be updated after its addition to the network as described in *Ref-1 (figure6).*

Sending Messages:

A message is generated randomly and a corresponding 'm' bit hash called "key" of that message is generated and is passed to be search for its location in the network by every node. Lookup algorithm works in the following way:

Case1: A node look if this key lies between it and its predecessor. If yes then the location where key would be stored is current node and it will report that the node is found.

Case2: Otherwise, if the key is present between the current node and its successor then the key will be stored at its successor where it will be routed thereby increasing hop count to 1.

Case3: Otherwise then the current node will forward the lookup query to the closest preceding finger table entry where the above cases will be executed again in the same order.

Following this algorithm the message is forwarded each time keeping a count of the hop it takes to reach there. Before terminating at a node where the key is found, bootstrap node is informed about the number of hops it took.

Bootstrap node keeps accounting of all the hops and reports the average hop count once all the messages have been routed.

Observation:
Avg. Hop Count v/s Size of Network : m=24

Number of Nodes	**Average Hop Count**	**Log(*numNodes*)**
10	1.5	3.32
20	3.6	4.32
50	4.28	5.64
80	4.45	6.32
100	4.81	6.64
200	6	7.64
400	6.83	8.64
700	7.99	9.45
1000	8.24	9.96
2000	9.25	10.96
4000	10.745	11.96
6000	11.34	12.55
8000	11.46	12.96
10000	11.7396	13.28
20000	13	14.28
50000	14.54	15.6

Conclusion:

From above table it can be easily verified that:

Average Hop Count < Log(*numNodes*)

For every *numNodes*, we can verify that we are able to achieve logarithmic time average look up using the DHT.

Refer to "PriyanshuPandey_SouravParmar_Project3-Bonus" for fault tolerant implementation.

Further work

- The network was simulated only on a single machine across multiple actors. A distributed model where peers are located at remote location and connected by IP was not tested.

References

- Stoica, I.; Morris, R.; Karger, D.; Kaashoek, M. F.; Balakrishnan, H. (2001). "Chord: A scalable peer-to-peer lookup service for internet applications" (PDF). *ACM SIGCOMM Computer Communication Review* **31** (4): 149. doi:10.1145/964723.383071
- https://en.wikipedia.org/wiki/Chord_(peer-to-peer)
- http://doc.akka.io/docs/akka/snapshot/scala/actors.html
- http://stackoverflow.com/questions/3188672/how-to-elegantly-check-if-a-number-is-within-a-range
- https://www.youtube.com/watch?v=q29szpcnorA