COP 5615: Project 3 Chord Protocol

Sumeet Pande UFID: 4890-9873

Drumil Deshpande UFID: 8359-8265

Introduction:

The main purpose of this project is to implement the chord network P2P protocol using Scala and Akka actor framework. As per our implementation the user will input two parameters the total number of active nodes and the number of messages to be transmitted per active node. We basically have created two types of nodes the Master Node and Worker Nodes (which actually form the chord P2P network). The master node initiates nodes whose number is significantly larger than the number of active number. After that, a randomization algorithm based on encryption of random string by making use of SHA-1 algorithm activates the random nodes. Once the nodes are activated they create their own finger Table for lookup purposes and join with other nodes to form a p2p network based on the chord protocol. Once this is done, we start the process of sending messages to other active and based on the different parameters calculate the average number of hops per sent message.

Code Structure:

Main: Parsing the input given by the user if correct. Creating the actor system and the master actor.

Master Actor:

(a) Start Case: Depending upon the number of active nodes provided by the user initialized total nodes whose number is significantly higher than actual number. After initializing the total numbers it selects some active nodes based on a random functionality based on SHA-1 encryption and calls another case MakeNodeActive.

(b) Case MakeNodeActive: Calls each individual active node and initiates the process to create a finger Table as well as update the same when other active nodes join the network. It ensures that the same is done for all the active nodes in the network.

(c) Case StartScheduling and Received: Starts the message passing process where each active nodes passes message to other nodes and keep a tab of the total number of messages and total hops. Terminates the program when the total # of messages equals (number of message each active node * no of active nodes).

Worker Actor:

(a) Case Join and UpdateSucc: These method are responsible for creation of finger table for look up purposes. Also the finger table for each node gets updated when a new active node is added to the network.

(b) Case startCalling and beginCallingProc and passMsg: These cases are basically used for message passing mechanism. The worker actor after each second select another random node, passes message to the node and depending upon the look up functionality the message passes from the source to destination.

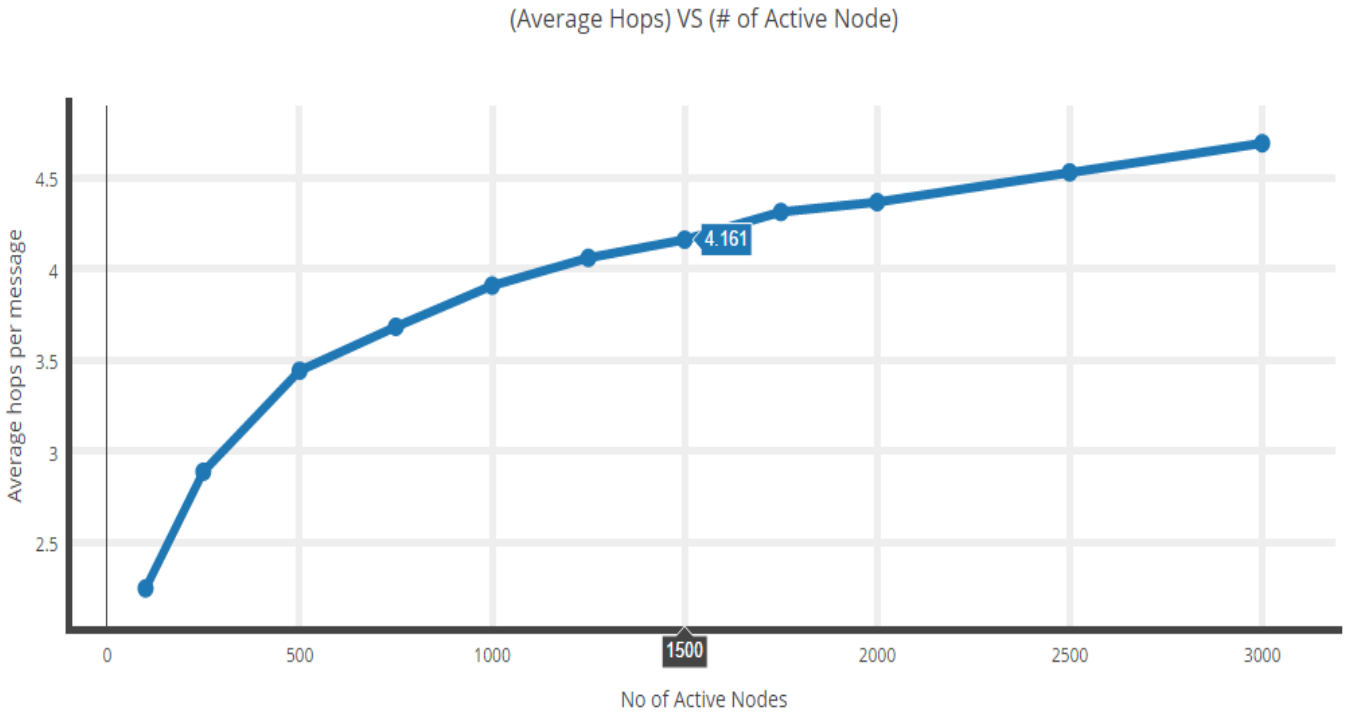
Deliverables:

Our program basically prints the average number of hops that have to be traverse to deliver the message. The screen shot for the same when Total Nodes =8192 and # of active nodes was 3000 is given as below:

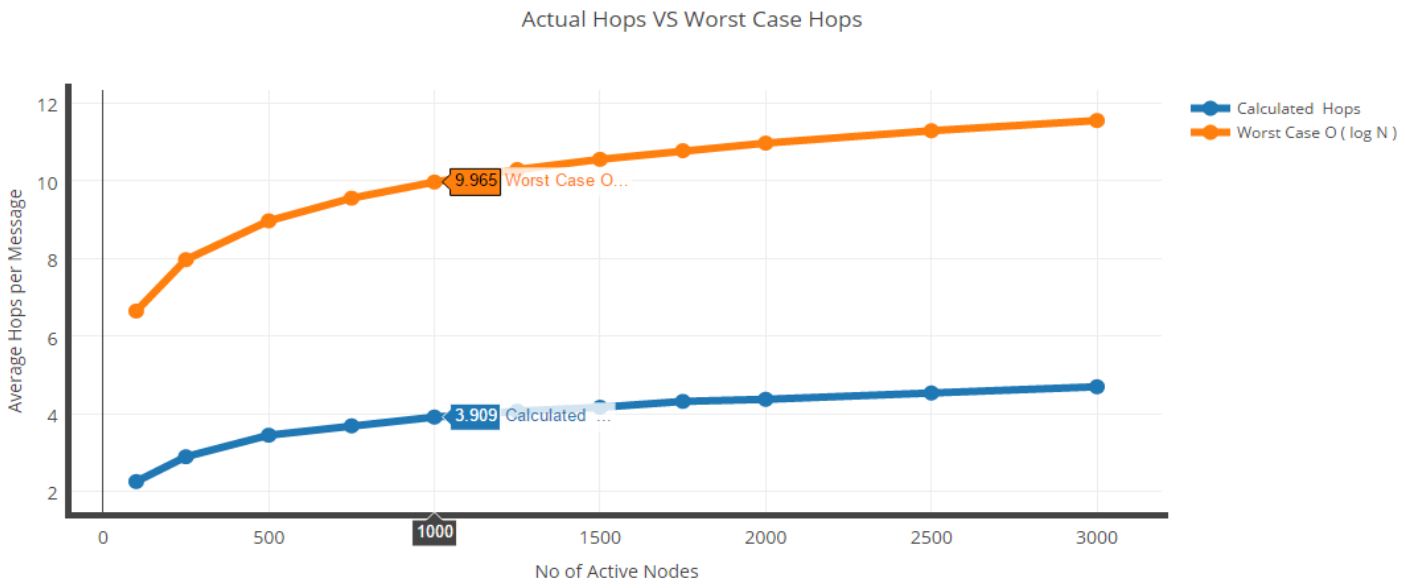
```
Console
<terminated> project3$ [Scala Application] C:\Program Files (x86)\Java\jre1.8.0_45\bin\javaw.exe (Oct 25, 2015, 2:21:24 PM)
Input Arguements Entered. Chord P2P protocol started
Master Node has been Initiated
The total number of active nodes : 3000
Value of m 13
Total Number of Nodes in p2p network is 8192
Average hops per message: 4.690633333333333
Total hops: 140719, Total msg Rec: 30000
Total number of active nodes: 3000, and total num nodes: 8192
```

Based on the number of nodes we have come up with some graphical analysis for different number of nodes or different number of messages. These are described as follows:-

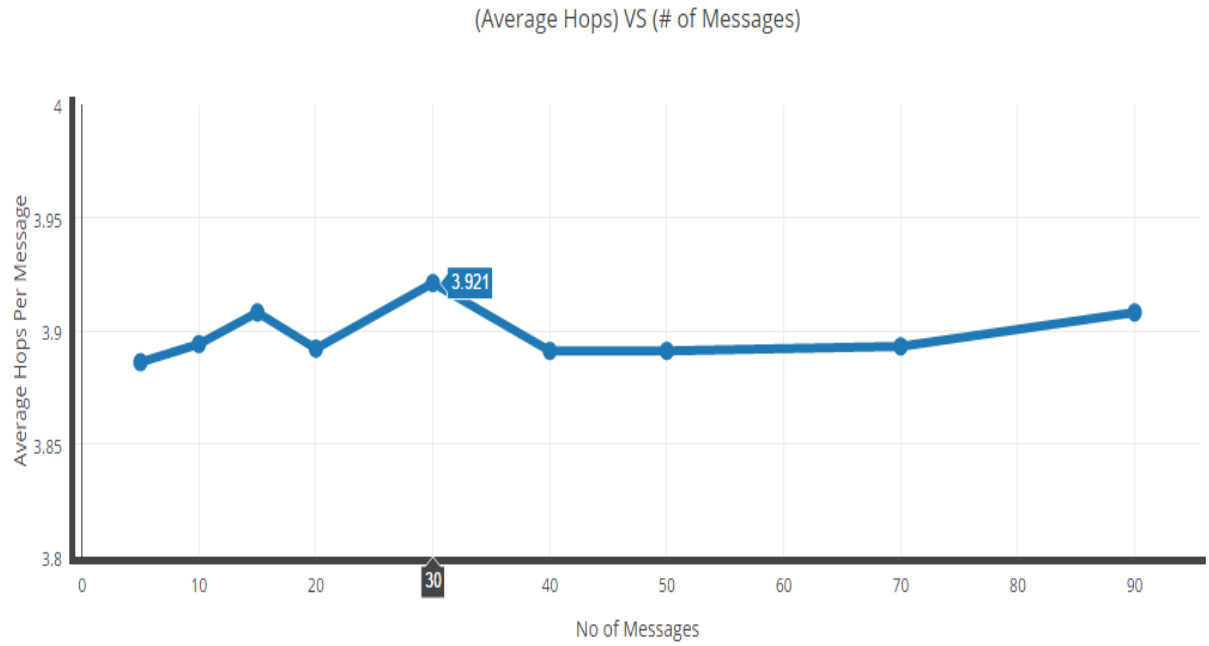
Observation 1: The average number of hops increases logarithmic with the number of nodes. We calculated the value of average hops against different number of active nodes varying from 100 to 3000 and plotted a graph for the same.



Observation 2: For each value of the average hops per message we have found that it is always below or equal to the worst case value O (log N) where N is the number of active Nodes. We also plotted the worst case value in all our entries and found that actual average is always smaller than the worst case. The graph for the same is attached as follows.



Observation 3: The number of messages transmitted per active node does not have any significant or major effect on the value of average hops per transmitted message. We actually found that this value remains more or less constant and actually converges as the number of messages increases. The graph for the same is as follows.



Challenges Faced:

The major challenges faced were to ensure certain level of sequential execution of the code so that the finger table gets updated properly. The code for creating the finger table was quite challenging. The other was the look up functionality for the same.

Additional Data:

The tabular information for our entire analysis is as follows:

Total # of Active Nodes	Calculated Average Hops	Worst Case Hops
100	2.248	6.641
250	2.888	7.965
500	3.442	8.965
750	3.683	9.55
1000	3.909	9.965
1250	4.061	10.287
1500	4.161	10.55
1750	4.313	10.773
2000	4.367	10.968
2500	4.53	11.287
3000	4.69	11.55

For 1000 Nodes	
# of Messages	Average Hops Per Message
5	3.886
10	3.894
15	3.908
20	3.892
30	3.921
40	3.891
50	3.891
70	3.893
90	3.908