Project Report

Peer to Peer Pastry Protocol

Shubham Agiwal 20562669 – Karan Sharma 00174451

Introduction: Peer to Peer systems have gained popularity in the recent past. This is due to the fact that p2p systems have many interesting technical aspects like decentralized control, self-organization, adaptation and scalability. In this project we have implemented the pastry protocol which is one of the popular p2p protocols (chord, can, pastry). Pastry can be used in the application layer for routing and object location in an overlay network. The most common applications of pastry are – global data storage, data sharing etc.

Pastry Protocol:

- Each node in the Pastry network has a unique identifier(node_id).
- When a node receives a message and a key, Pastry nodes efficiently route the message to the node with node_id numerically closest to the key.

Structure of a node:

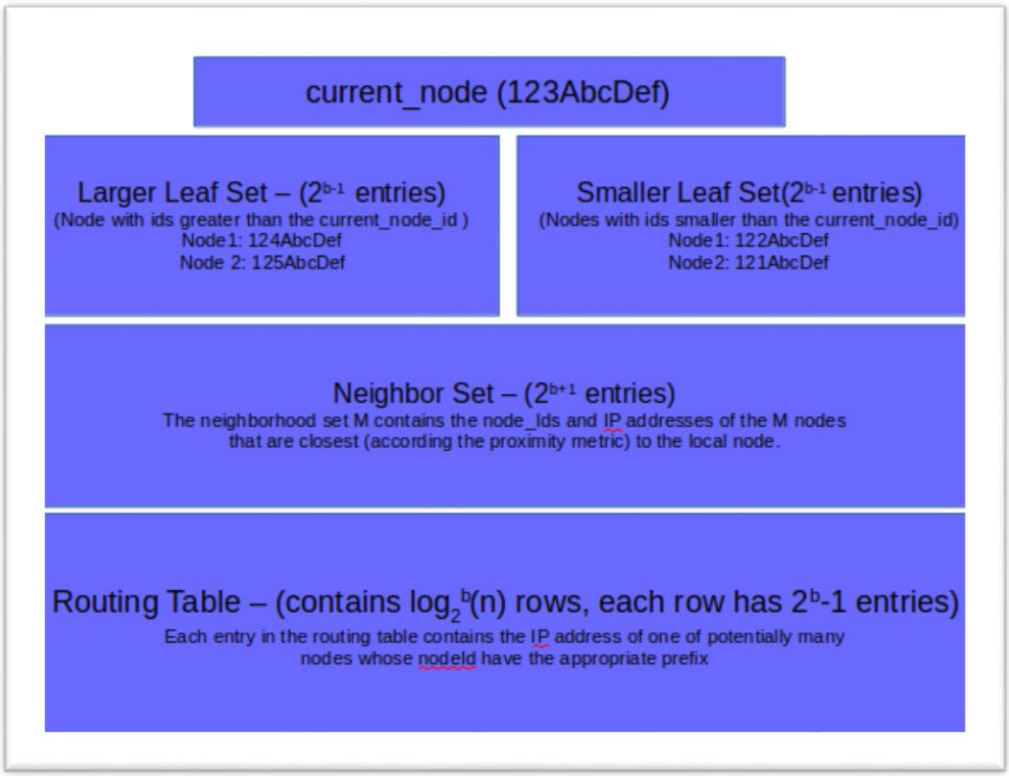

Fig1. The above shows the structure of a node. In our case we have taken the b-value to be 2

Implementation Details:

- The first step in our program is creation of the pastry.
- We build our pastry by initializing the pastry with the number of nodes and number of requests each node has to get.
- We call the build_topology function in our Boss module.
- The build_topology function will do the following:
    I. It will call larger_leaf_set function of function in the Node module. This will return the larger leaf set for the local node.

II. It will call the smaller_leaf_set function in the Node module. This will return the smaller leaf set for the local node.
III. It will call the neigbor_set function and routing_table function from the Node module and will create the neigborhood set and routing table for the current node.
IV. Thus we get the larger leaf set, smaller leaf set, neighbor set and routing table for that node. This process is repeated for all the nodes until the all nodes have been setup.

- A random key is generated in the Node module – this key is hashed into a message and then routed to the node which is numerically closest to the key.
- This above is accomplished using the routing of the pastry:
  I. When a key is received on a node say – that it starts at the source. The source node at this instance will check in its leaf sets if there is a match for the key been sent. If there is a match we increase the hop count by 1 and send a message back to the Boss module that the key has been received.
  II. If the key is not present in the leaf sets we check the routing table. The routing table helps in routing by using the concept of longest-prefix-matching. There is a function in the Node module for matching the longest prefix.
  III. If there is no match in the routing table we will combine all the four tables mentioned previously in the node structure and look for our key. If the key is found we will send a message to the Boss module the key has been received and the average hop count will be printed.
  IV. If still there is no match and the hop count equals to the product of the number of requests times the number of nodes we will send a message to the Boss module signaling it to stop the program.

Results:

Number_of_nodes	Number_of_requests	Average Hop Count
100	**10**	**1.3610**
200	**10**	**1.3470**
300	**10**	**1.3356**
400	**10**	**1.3265**
500	**10**	**1.2972**
600	**10**	**1.3383**
700	**10**	**1.2705**
800	**10**	**1.2841**
900	**10**	**1.2564**
1000	**10**	**1.3581**
2000	**10**	**1.3622**

Number_of_nodes	Number_of_requests	Average Hop Count
For 100 nodes		
100	**50**	**1.4056**
100	**100**	**1.4190**
100	**150**	**1.4110**
100	**200**	**1.4446**
For 400 nodes		
400	**50**	**1.3811**
400	**100**	**1.3812**
400	**150**	**1.3800**
400	**200**	**1.3863**
For 800 nodes		
800	**50**	**1.3149**
800	**100**	**1.3090**

800	**150**	**1.3181**
800	**200**	**1.3096**

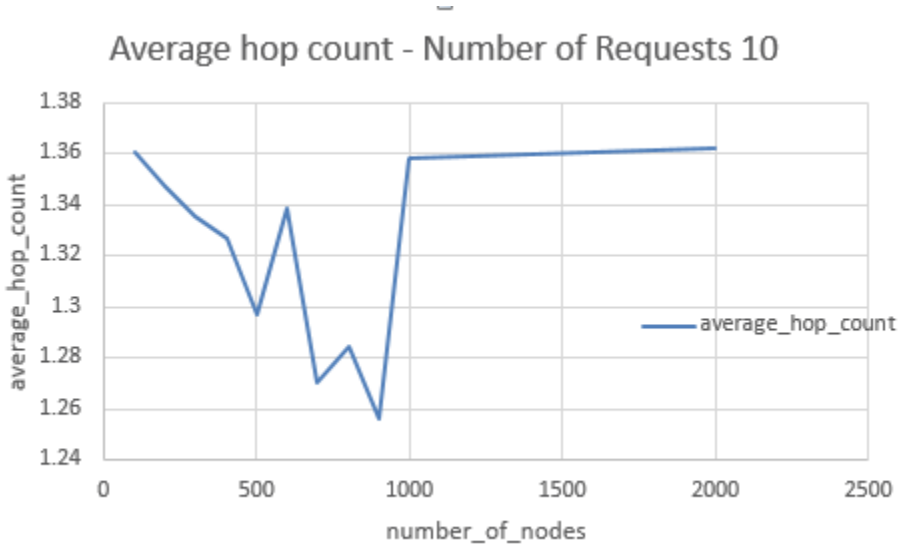

Fig-2 Graph showing average hop count for different number of nodes

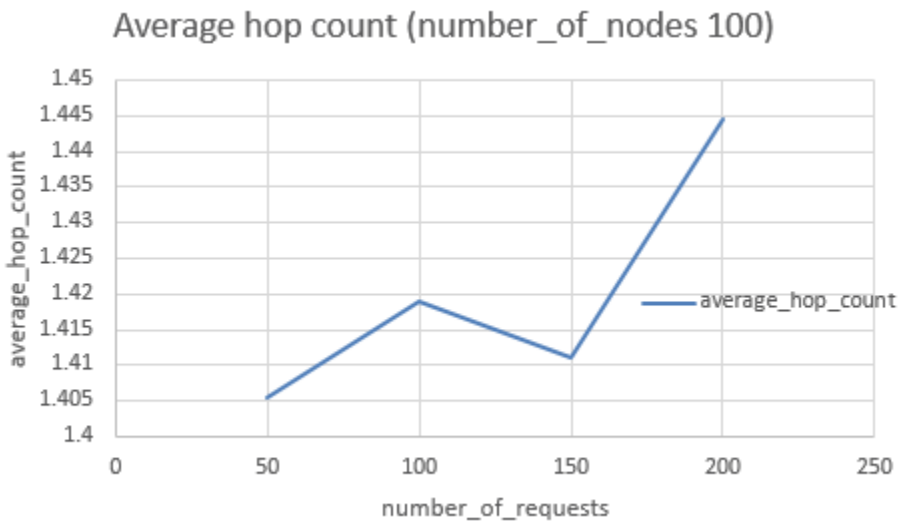

Fig-3 Graph showing average hop count for different number of requests

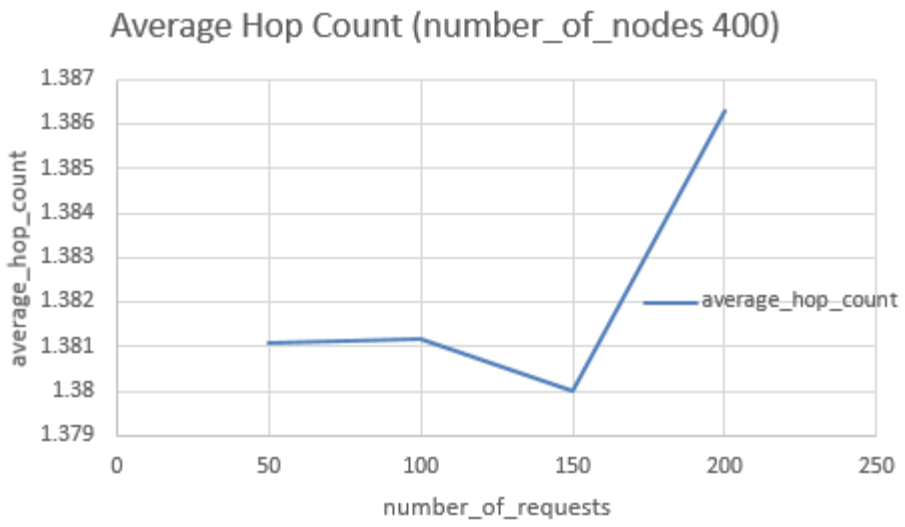

Fig-4 Graph showing average hop count for different number of request

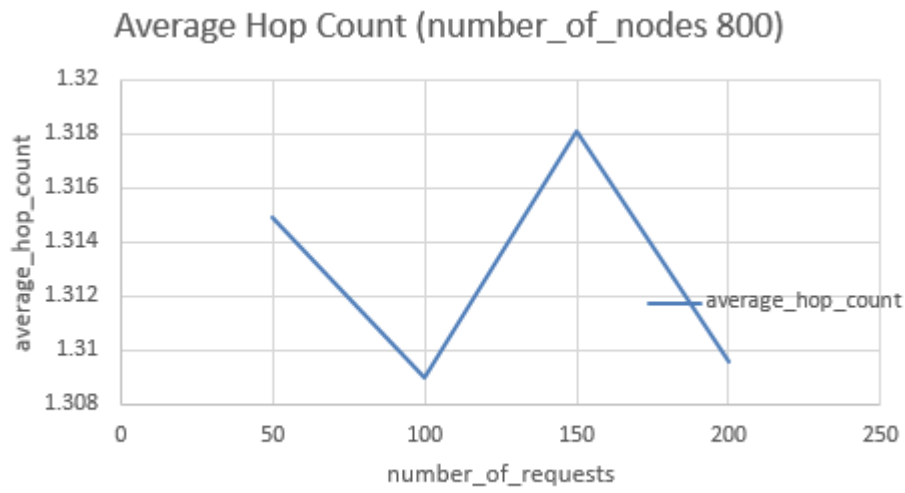

Fig-5 Graph showing average hop count for different number of requests

Conclusion: After looking at the graphs we finally concluded that:

- When number of requests are same and we increase the number of nodes in the network we see that the average hop count decreases and increases till 900 nodes – but when we run our program for 1000 nodes or greater there was an increase in the average hop count.
- We saw that the average hop count was in the range [1.25,1.36]. This is also because of the way our topology is designed.
- The nodes are created in a distributed index based architecture where a given node will have node_id-1 and node_id+1 as its neighbor. Their hashes might vary by a significant factor.
- We also noticed that when you keep the number of nodes fixed and increase the number of request say 50,100,150,200 initially there is a gradual increase but eventually average hop count increases.