ECE658 Internet Engineering

Lab-5: Distributed Hash Table Based Content Searching in a Structured Peer-to-Peer Network Report

Student Names: Varun Bhat, Venkatesh Babu Musuvathy Santharam

Introduction:

The objective of the assignment is to develop a structured peer to peer distributed system (DS) following chord algorithm where each node has certain distinct resources. Any node in the system may request for a resource. It has to get the resource if found or get a message saying that resource is not found in the network of nodes. The nodes are formed in the DS by using a bootstrap (BS) server. Once the nodes are connected until a required scale (say 40 nodes), a master node is used to initiate a query picked using Zipf's distribution, to randomly selected nodes. The master node learns the nodes in the DS by contacting the BS server. The queries are injected into the DS according to Zipf's distribution. SHA1 function is used to obtain hash key of nodes as well as resource entries. 16 bits of the hashed key are used in limiting the key space to represent a node or entry. Every node is responsible for certain number of entries which is decided using its hash key value.

Definitions:

Latency is defined as the time taken for a node in the distributed system to get a response for a query. Hops is the number of nodes a query visits.

Architecture:

The code is executed in an event driven approach. A Library is written called AsyncNode, that handles the request messages and pushes them into an event driven thread. The architecture is similar to a Node JS architecture and is written to work similarly to such an architecture. The Thread manager in the Async Node is used to manage new events pushed on to the system. It is responsible to start, manage persistent threads and stop the threads on request.

A Parser is implemented to handle all the messages received or to be sent. Therefore, the main program concentrates on the architecture of the main code without having to worry about the thread handling and socket interface part of the code.

Chord Class is a Class that is used to maintain the hash table and functions necessary for the peer address and node files.

The application is written in a scalable approach. i.e., any number of events can be registered by just listening to the keyword that is received in the message.

Chord Implementation procedure is shown in the pseudocode below

Pseudocode:

<u>For Nodes in Distributed System:</u>

```
1. Register with BS server. Get the details of peer nodes in the distributed system
2. Get the successor of this node from the peer node list obtained from BS server. Send "UPFIN
   0" with the node's ip, port and key to the peer nodes
3. Get the keys from successor that this node is responsible for
```

4. Pick half of the entries from the top list of the resources
5. Wait for further commands i.e. wait until 20/40/80 nodes join the distributed system
6. Pick rest half of the entries from the bottom list of the resources
7. Send ADD key command for each entry in the bottom list of resources to the finger table members
8. If you receive UPFIN command, then
    a. if the type is peer add, then add the entry into your finger table if the key of the newly added peer fits into the table and send UPFINOK command, else forward the command to closest preceding node
    b. if the type is peer depart, then check if the key of the departing peer exists in your finger table, if so then remove the key and update it with successor of the key, else forward the command to closest preceding node
9. If you receive ADD command, then
    a. check if the you are the successor of the key, if so then add the key details into your key table, send ADDOK command to the node containing the resource key
    b. else, forward the command to closest preceding node
10. If you receive GETKY command from your predecessor then:
    a. get the keys from your key table that that predecessor node is responsible for
    b. send the keys to the predecessor using GETKYOK command
11. If you receive ENTRIES command, then display the entries you have
12. If you receive DETAILS command, then display your IP, PORT, Key details
13. If you receive FINGERTABLE command. then display the finger table entries
14. If you receive KEYTABLE command, then display the key table entries
15. If you receive EXIT command, then send your keys to your successor using GIVEKY command, unregister from BS server and exit the network
16. If you receive EXITALL command, then send your keys to your successor using GIVEKY command, send EXIT command to successor, unregister from BS server and exit the network
17. If you receive GIVEKY command from predecessor, then read the keys and update your key table, send GIVEKYOK command to the predecessor
18. If you receive STARTSEARCH command, send a SER command with the entry to be searched for
19. If you receive SER command then
    a. get the key of the entry to be searched for
    b. find the successor of the key i.e. look in your finger table and forward the command to the closest preceding node
    c. increment the hop count of the query
    d. if you are the successor of the key, then check if you have the entry in your key table
    e. if yes, send SEROK command with the IP, PORT and Key details of all the nodes that contain the queried entries to the requested node
    f. if no, send SEROK failed command
20. All the commands are handled as events by a node

Network Size	Max(Max Hops)	Max(Min Hops)	Max(Average Hops)
20	5	3	3.75
40	6	3	4.5
80	11	6	8

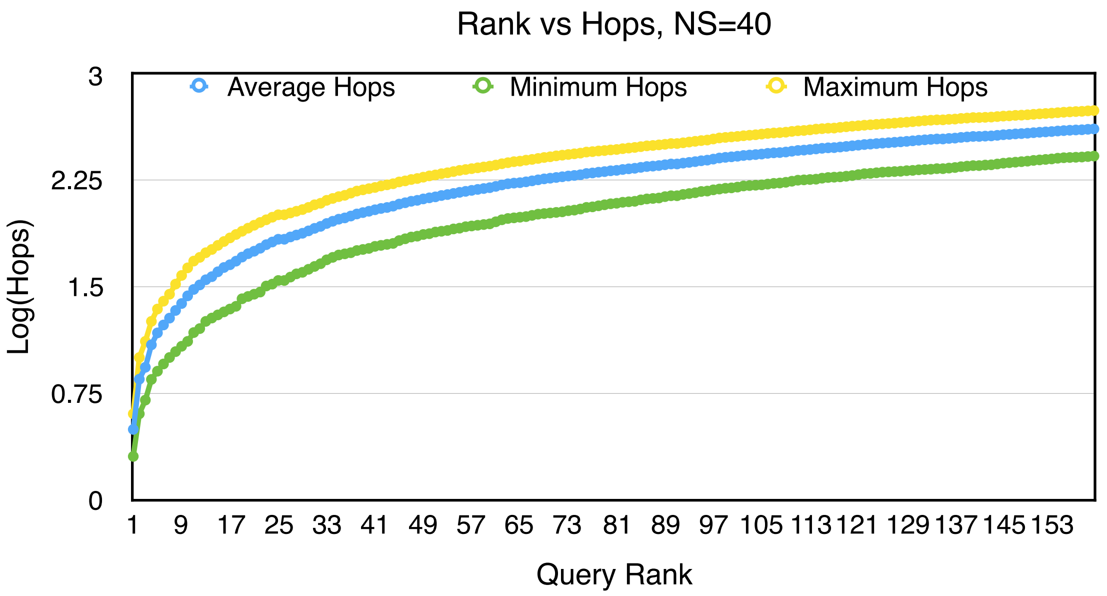
Table-1: Maximum hops for each network size is less than log(N) = 16

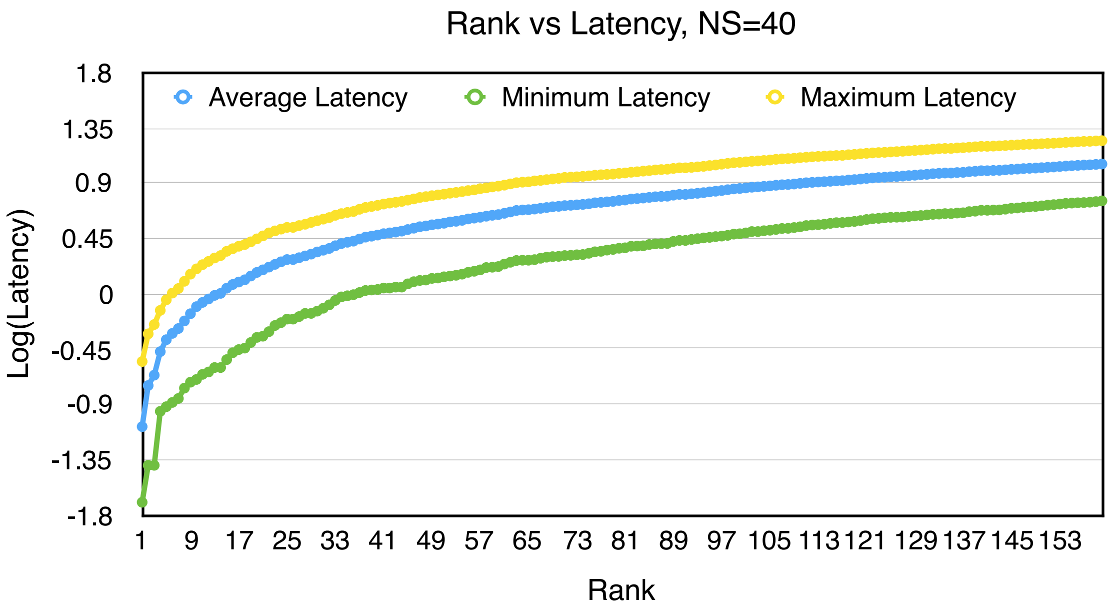
Network Size	Query Popularity	Max Hop	Min Hop	Average Hop
20	High	3	1	1.96
	Medium	3	2	2.25
	Low	3	2	2.67
40	High	4	2	3.11
	Medium	4	2	3
	Low	3	2	3.33
80	High	5	1	3
	Medium	8	2	4.75
	Low	3	2	2.5

Table-2: Comparison b/w Max, Min and Average hops for different NW size

Network Size	Per Query Cost	Per Node Cost
20	1.73	23.89
40	2.97	31.19
80	3.11	51.67

Table-3: Query Cost and Node cost vs NW size


Rank vs Hops, NS=40
Average Hops
Minimum Hops
Maximum Hops
Log(Hops)
3
2.25
1.5
0.75
0
1
9
17
25
33
41
49
57
65
73
81
89
97
105
113
121
129
137
145
153
Query Rank


Rank vs Latency, NS=40
Average Latency
Minimum Latency
Maximum Latency
Log(Latency)
1.8
1.35
0.9
0.45
0
-0.45
-0.9
-1.35
-1.8
1
9
17
25
33
41
49
57
65
73
81
89
97
105
113
121
129
137
145
153
Rank

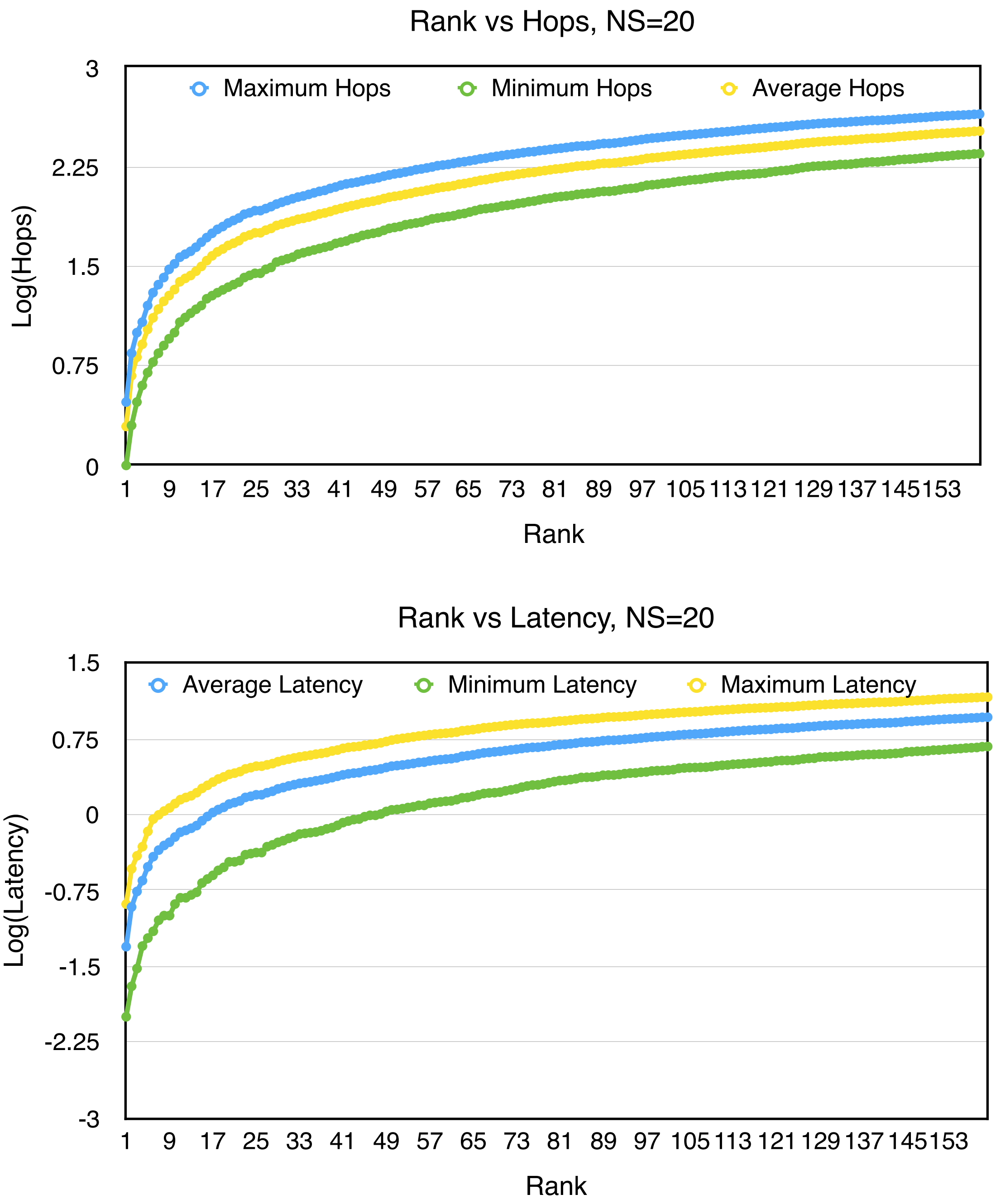


Figure-1: Max, Min and Average Hops and Latency for NW size = 40

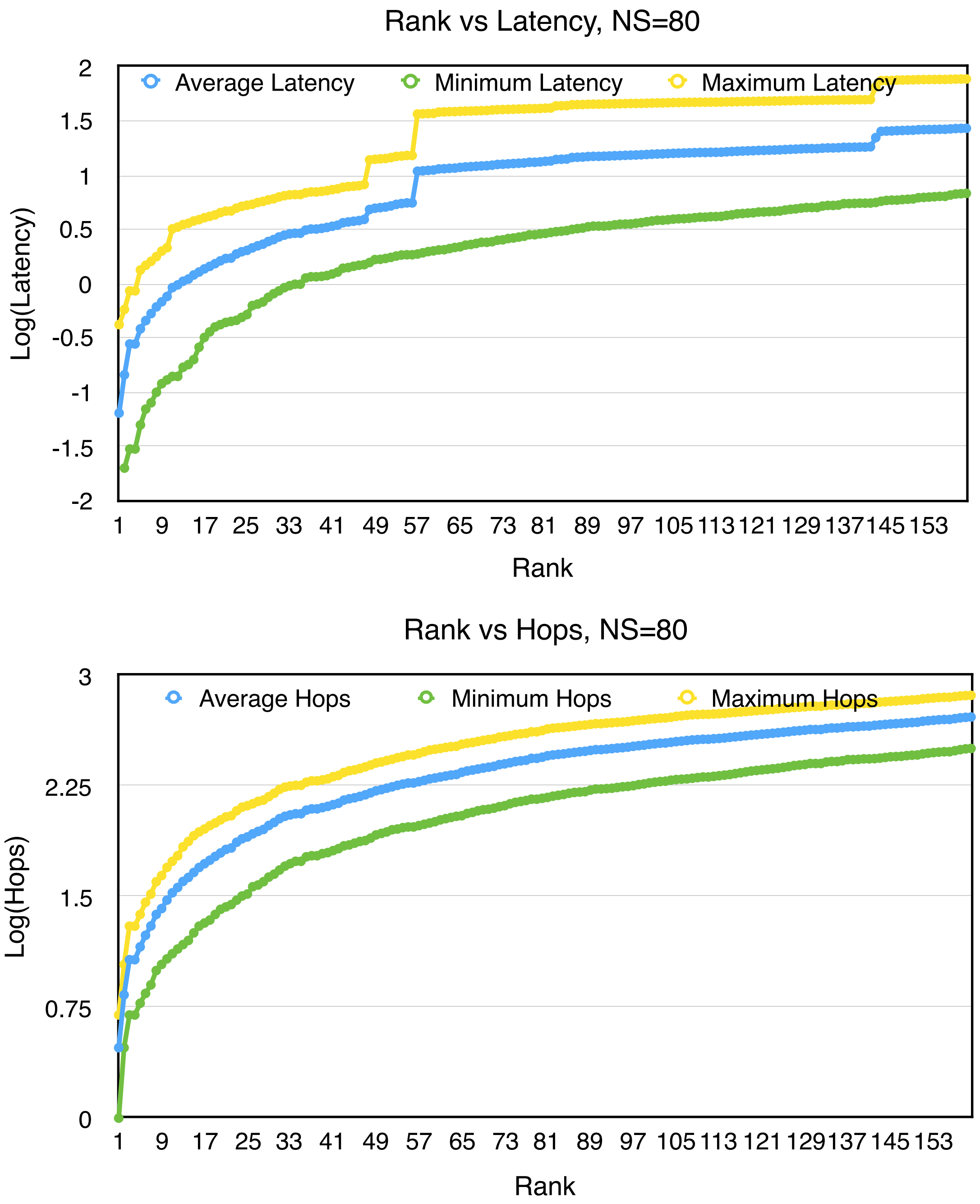


Figure-2: Max, Min and Average Hops and Latency for NW size=20

Figure-3: Max, Min and Average Hops and Latency for NW size=80