Implementation of Pastry DHT

Dipika Tanwar (MCS202456)

1 INTRODUCTION

This report presents the experiment data generated through pastry implementation. In theory, a pastry network with N nodes requires $log_{2^b} N$ Hop count on average to search a node in the pastry network from any starting node. In general start node is chosen to be the node geographically near to the searching node.
In pastry, both NodeId and KeyId are 128 bit unique numbers. Also, nodeId and KeyId are in the same space. Therefore searching a node is equivalent to searching a key. Implementation of pastry, requires to tune hyper-parameters b, breadth of pastry network tree, L size of the leaf set for each pastry node. Among which $L/2$ are on each left and right sub-tree respectively. M size of a neighbouring set, i.e. M nearby nodes in geographical distance.

Each pastry node has the following data structure available to it.

1. Leaf Set: L nearby node in Id distance.
2. Neighbouring Set: M nearby nodes in the geographic distance.
3. Routing table: R used to route a node.

In the below section implementation of pastry with various settings is depicted.

2 ASSUMPTIONS

Following assumptions are made for this implementation.

1. Implementation is Single-threaded. i.e. all the pastry nodes communicate information sequentially.
2. Upon addition of new nodes. The previous assignment of key-value is rearranged.
3. Deletion of a node is done gracefully. Before a node leaves the network, it informs all the other active nodes.
4. In case of deletion. The Routing table, Neighbouring set, and Leaf set of all other nodes recover immediately.
5. After deletion, key-value related to deleted nodes are redistributed.

3 PROGRAMMING ENVIRONMENT

Python 3.6 is used for this assignment. We have used the MD5 package to create a 128 bit uniform hash for NodeId. basic python package needed.

- hashlib: 128-bit unique nodeId and KeyId value.
- numpy: Array operations.
- matplotlib: Plots.
- math: Mathematical function i.e. pow, log, etc.

3.1 Source Files

1. main.py: Driver file for the program.
2. Pastry.py: Pastry DHT object.
3. Node.py: Node (Peer in Pastry) Object class.
4. IPNetwork: Provides API from overlay IP network. Used to get Network information.

4 CONFIGURATION

Following settings have been used during the execution.

1. Network breadth: $b = 4$.
2. Neighbouring set size: $M = 32$.
3. Leaf set size: $L = 16$.
4. Routing table: R. Size of routing table (32,16), for b=4.

However, all the APIs are configurable and the value of b can be provided as a parameter to pastry object.

5 EXPERIMENT

Sample Program execution logs.
**********************Start Simulation ****************************

Started Pastry Simulation With initial Nodes = 1000

Adding data Points = 10000

Performing random search Queries = 1000000

Average Nums of Hops = 2.518307

Performing deletion of half nodes = 500

Performing random search Queries = 1000000

Average Nums of Hops = 2.384908

Total number of nodes : 500

Total number of data elements : 16168

Total number of search : 2000000

Total number None Del Query: 500

Routing Table Of **92fb0c6d1758261f10d052e6e2c1123c**

S.N. Successor

0 — ['0ff8033cf9437c213ee13937b1c4c455', '1ff8a7b5dc7a7d1f0ed65aaa29c04b1e', '2f885d0fbe2e131bfc9d98363e55d1d4', '3fe94a002317b5f9259f82690aeea4cd', '4ffce04d92a4d6cb21c1494cdfcd6dc1', '5fd0b37cd7dbbb00f97ba6ce92bf5add', '6faa8040da20ef399b63a72d0e4ab575', '7fe1f8abaad094e0b5cb1b01d712f708', '8fecb20817b3847419bb3de39a609afe', 'a01a0380ca3c61428c26a231f0e49a09', 'b056eb1587586b71e2da9acfe4fbd19e', 'c042f4db68f23406c6cecf84a7ebb0fe', 'd045c59a90d7587d8d671b5f5aec4e7c', 'e00da03b685a0dd18fb6a08af0923de0', 'f033ab37c30201f73f142449d037028d']

1 — ['90794e3b050f815354e3e29e977a88ab', '9188905e74c28e489b44e954ec0b9bca', '934815ad542a4a7c5e8a2dfa04fea9f5
'941e1aaaba585b952b62c14a3a175a61', '950a4152c2b4aa3ad78bdd6b366cc179', '96b9bff013acedfb1d140579e2fbeb63', '9766527f2b5d3e95d4a733fcfb77bd7e', '9872ed9fc22fc182d371c3e9ed316094', '9908279ebbf1f9b250ba689db6a0222b', '9a1158154dfa42caddbd0694a4e9bdc8', '9b04d152845ec0a378394003c96da594', '9c01802ddb981e6bcfbec0f0516b8e35', '9dcb88e0137649590b755372b040afad', '9e3cfc48eccf81a0d57663e129aef3cb', '9f396fe44e7c05c16873b05ec425cbad']

2 — ['92262bf907af914b95a0fc33c3f33bf6', '9232fe81225bcaef853ae32870a2b0fe', '92977ae4d2ba21425a59afb269c2a14e', '92cc227532d17e56e07902b254dfad10']

3 — []

4 — []

5 — []

6 — []

7 — []

```
8 — []
9 — []
10 — []
11 — []
12 — []
13 — []
14 — []
15 — []
16 — []
17 — []
18 — []
19 — []
20 — []
21 — []
22 — []
23 — []
24 — []
25 — []
26 — []
27 — []
28 — []
29 — []
30 — []
31 — []
*************************End Simulation ********************
```

In the following section, experimental result are shown for three pastry configurations. 100, 500 and 1000 initial nodes.

5.1 SETTING-1

Pastry Network is created with 100 initial nodes. Figure 1 shows the probability distribution of hops count. with 10000 data points and 1000000 random search queries. Figure 2 shows probability distribution after removing 50 nodes from the network.

- Average hops count before deletion 1.664197
- Average hops count after deletion 1.528287

5.1.1 JUSTIFICATION

In theory we can achieve $log_{2^b}N$ hops count.So, actual number of hops count for 100 nodes is $log_{2^b}100 = 1.6609640474436813$, actual number of hops count required for 50 nodes is $log_{2^b}50 = 1.410964047443681$.
Experimental data supports data generated from pastry paper [1].

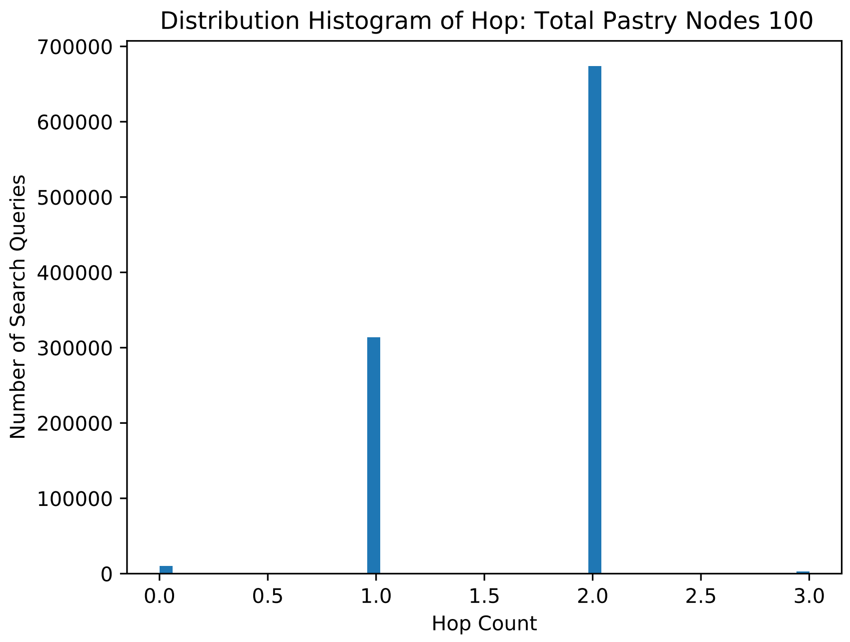


Figure 1: Distribution of Hops count. Pastry Network With 100 Nodes. 10000 Data Element with 1000000 Search queries.

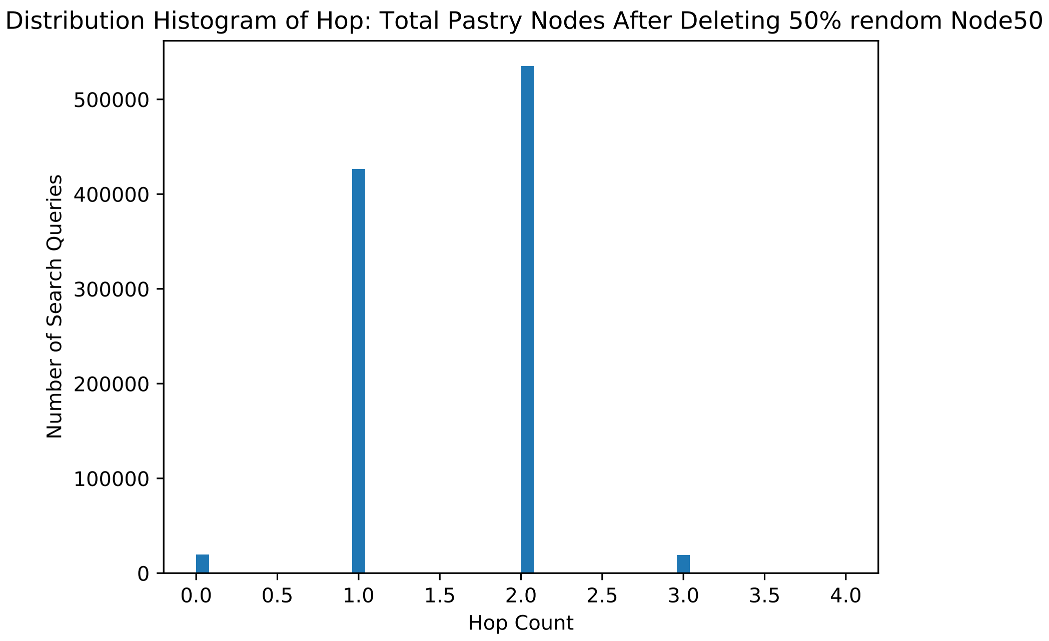


Figure 2: Distribution of Hops count. Pastry Network With 50 Nodes(After deletion of 50% Nodes from intial 100 nodes). 10000 Data Element with 1000000 Search queries.

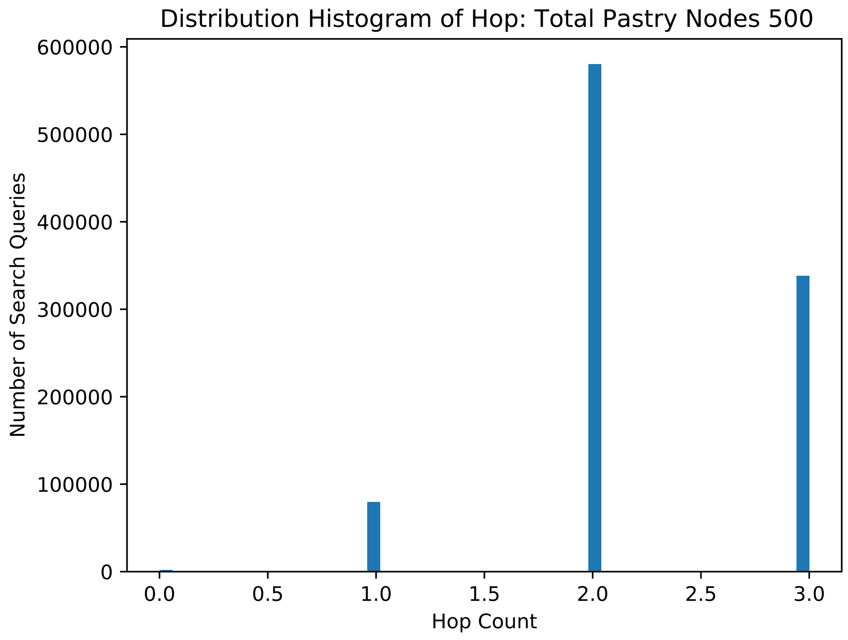


Figure 3: Distribution of Hop count. Pastry Network With 500 Nodes. 10000 Data Element with 1000000 Search queries.

5.2 SETTING-2

Pastry Network is created with 500 initial nodes. Figure 3 shows the probability distribution of hops count. This is obtained by inserting 10000 data points and after that performing 1000000 random search queries. Figure 4 shows probability distribution after removing 250 nodes from the network.

- Average hops count before deletion 2.2473.
- Average hops count after deletion 2.09346.

5.2.1 JUSTIFICATION

In theory we can achieve $log_{2^b}N$ hops count. Therefore number of hops count for 100 nodes is $log_{2^b}500 = 2.2414460711655217$. Number of hops required for 50 nodes is $log_{2^b}250 = 1.9914460711655217$.
Experimental data supports data generated from pastry paper [1].

5.3 SETTING-3

Pastry Network is created with 1000 initial nodes. Figure 5 shows the probability distribution of hope count. with 10000 data points and 1000000 random search queries. Figure 6 shows probability distribution after removing 500 nodes from the network.

- Average hop count before deletion 2.516525.
- Average hop count after deletion 2.379276

5.3.1 JUSTIFICATION

In theory we can achieve $log_{2^b}N$ hops count.So, actual number of hop count for 1000 nodes is $log_{2^b}1000 = 2.4914460711655217$, actual number of hop count required for 500 nodes is $log_{2^b}500 = 2.2414460711655217$.
Experimental data supports data generated from pastry paper [1].

6 HOW TO RUN

In the project base directory. Run the following file.

- python main.py

Figure 7 shows screen-shot, of console while program in execution.

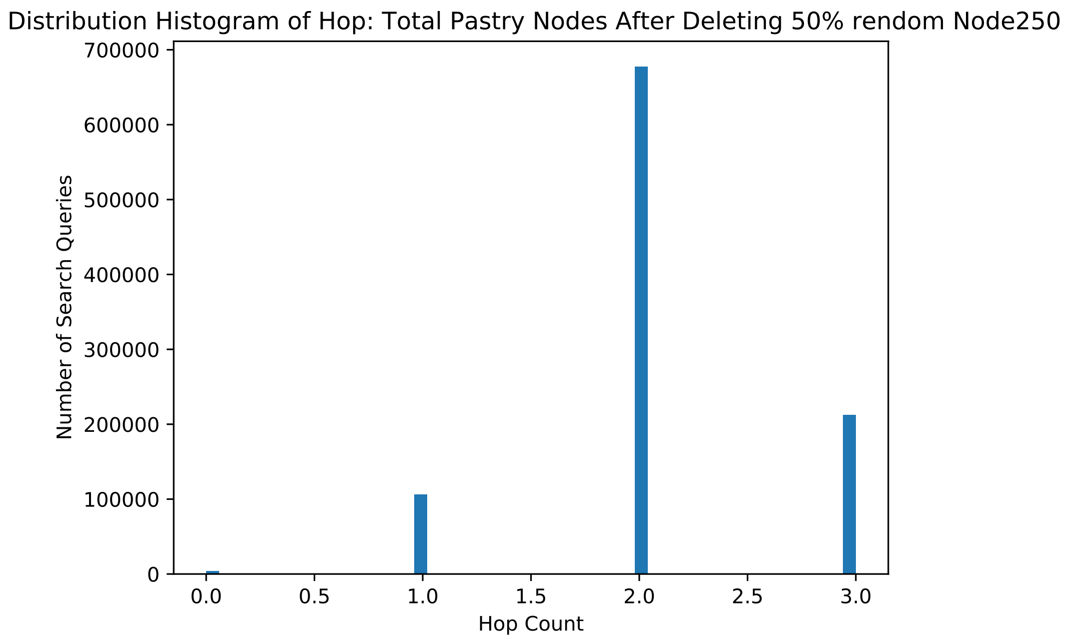


Figure 4: Distribution of Hop count. Pastry Network With 250 Nodes. 10000 Data Element with 1000000 Search queries. After deletion of 250 nodes from initial 500 nodes.

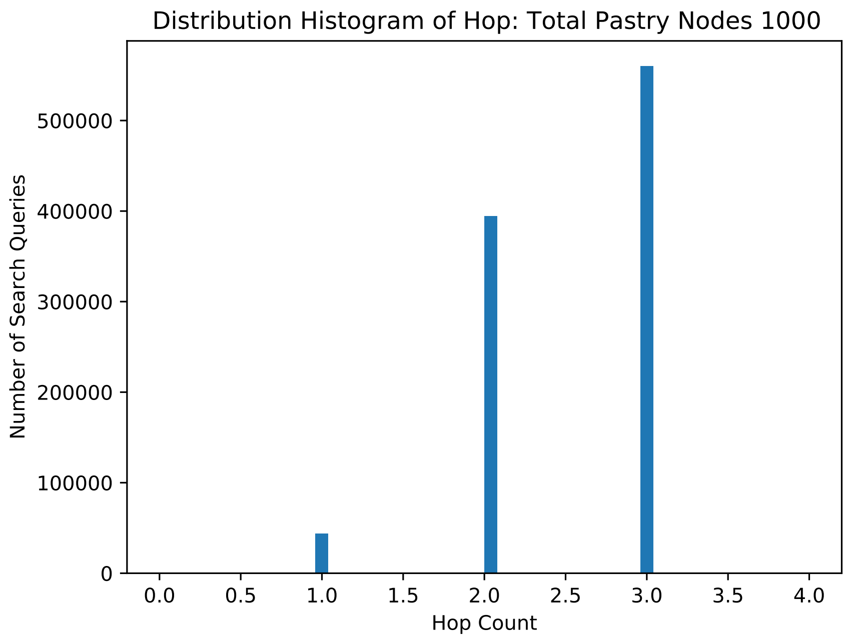


Figure 5: Distribution of hops count. Pastry Network With 1000 Nodes. 10000 Data Element with 1000000 Search queries.

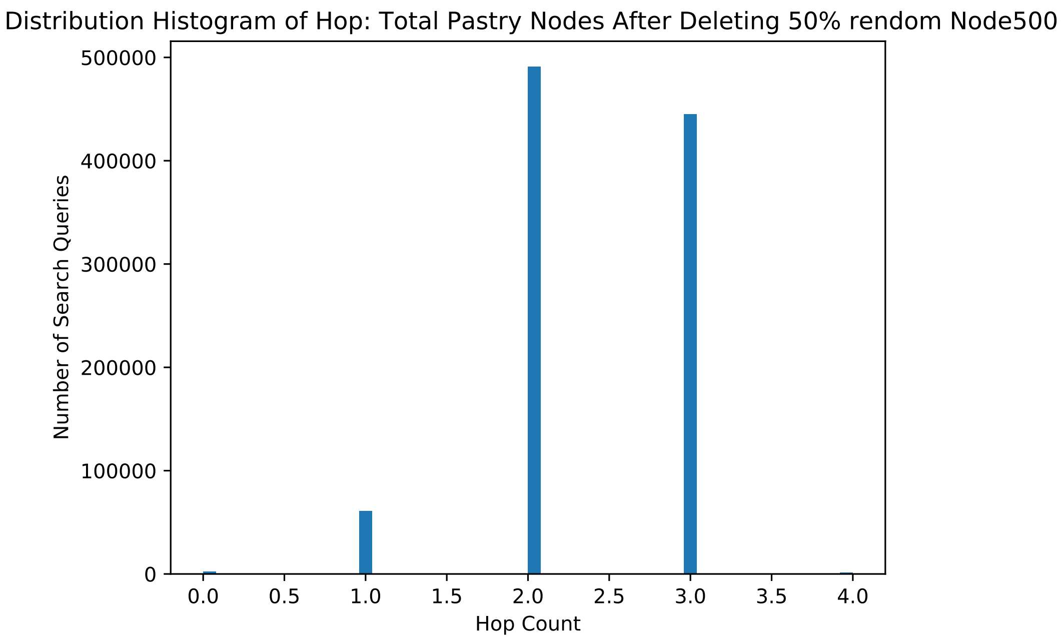
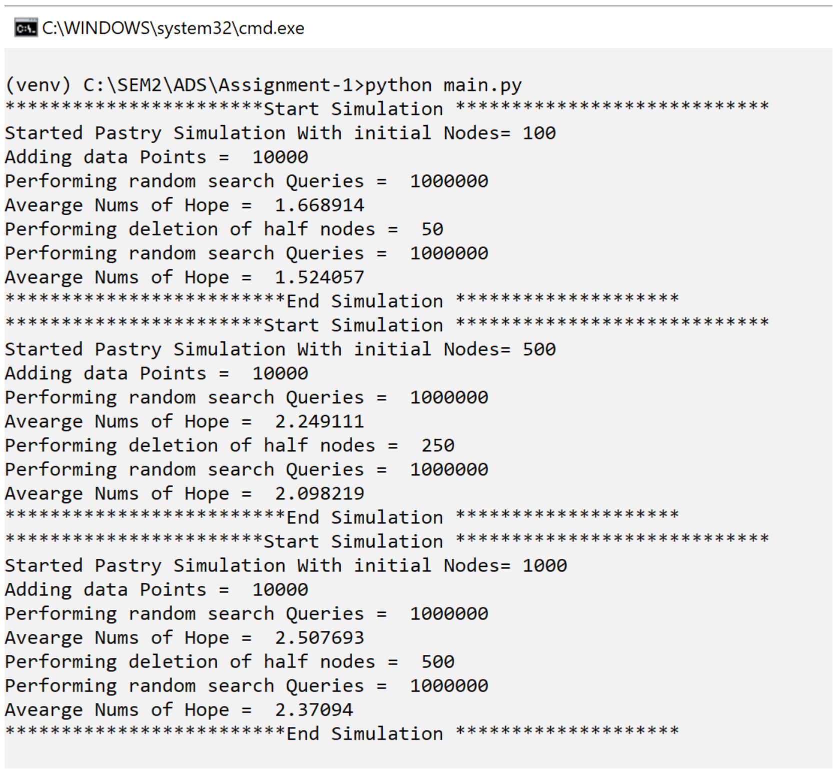


Figure 6: Distribution of hops count. Pastry Network With 500 Nodes. 10000 Data Element with 1000000 Search queries. After deletion of 500 nodes from initial 1000 nodes.



```
(venv) C:\SEM2\ADS\Assignment-1>python main.py
***********************Start Simulation ****************************
Started Pastry Simulation With initial Nodes= 100
Adding data Points =  10000
Performing random search Queries =  1000000
Avearge Nums of Hope =  1.668914
Performing deletion of half nodes =  50
Performing random search Queries =  1000000
Avearge Nums of Hope =  1.524057
*************************End Simulation ********************
***********************Start Simulation ****************************
Started Pastry Simulation With initial Nodes= 500
Adding data Points =  10000
Performing random search Queries =  1000000
Avearge Nums of Hope =  2.249111
Performing deletion of half nodes =  250
Performing random search Queries =  1000000
Avearge Nums of Hope =  2.098219
*************************End Simulation ********************
***********************Start Simulation ****************************
Started Pastry Simulation With initial Nodes= 1000
Adding data Points =  10000
Performing random search Queries =  1000000
Avearge Nums of Hope =  2.507693
Performing deletion of half nodes =  500
Performing random search Queries =  1000000
Avearge Nums of Hope =  2.37094
*************************End Simulation ********************
```

Figure 7: Screen shot of Program Execution

7 API Description

- pastry(nodeCount, b): constructor with nodeCount initial nodes and b Network breadth.
- pastry.addNode() : To Add a new node in pastry network. (Node ID is generated automatically)
- pastry.addKey(key, value): Add key value pair to pastry Network.
- pastry.findKey(key): Find key in pastry Network.
- pastry.statistics(): Print current pastry statistics.
- pastry.printNode(NodeId): Print node info (routing table, neighbouring set, leaf set of a node with nodeId)

References

[1] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, 2001.