Pastry and Chord Simulation

Harinder Pal (2014MCS2123)
Kapil Thakkar (2014MCS2124)

September 21, 2014

Abstract

This document contains the results of experiments carried out during chord and pastry simulation. The results are then compared with the theoritical aspects

1 Pastry

Pastry is a scalable, distributed object location and routing substrate for wide-area peer-to-peer applications. Each node in the Pastry network has a unique identifier (nodeId), when presented with a message and a key, a Pastry node efficiently routes the message to the node with a nodeId that is numerically closest to the key, among all currently live Pastry nodes.

Pastry is completely decentralized, scalable, and self-organizing; it automatically adapts to the arrival, departure and failure of nodes.

Figure 1 shows a hypothetical Pastry node with nodeId 10233102. Each Pastry node contains a leaf set, a routing table and a neighborhood set.

Figure 2 depicts the working of Pastry while routing.

1.1 Experimental Results

All the below results are taken with b = 4, |L| = 16 and |M| = 32. The emulated network environment maintains distance information between the Pastry nodes. Each Pastry node is assigned a location in a plane; coordinates in the plane are randomly assigned in the range [0,1000].

It is assumed that the application provides a function that allows each Pastry node to determine the "distance" of a node with a given nodeId to itself. A node with a lower distance value is assumed to be more desirable. (such a functionality is provided in the emulation)

1.1.1 Average number of routing hops

The number of Pastry nodes are increased from 100 to 10,000 and each time 100,000 trials are performed to calculate the average.

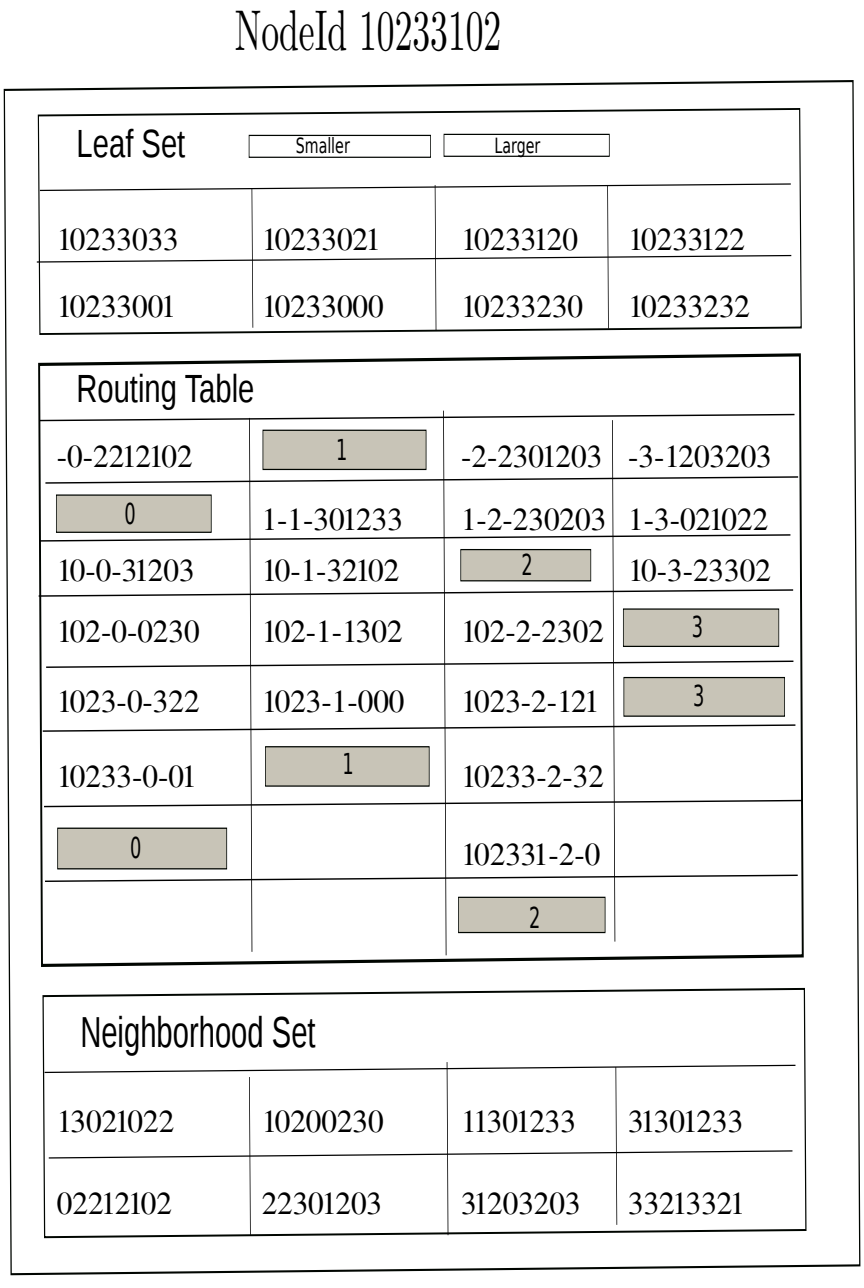


Figure 1: Hypothetical Pastry node

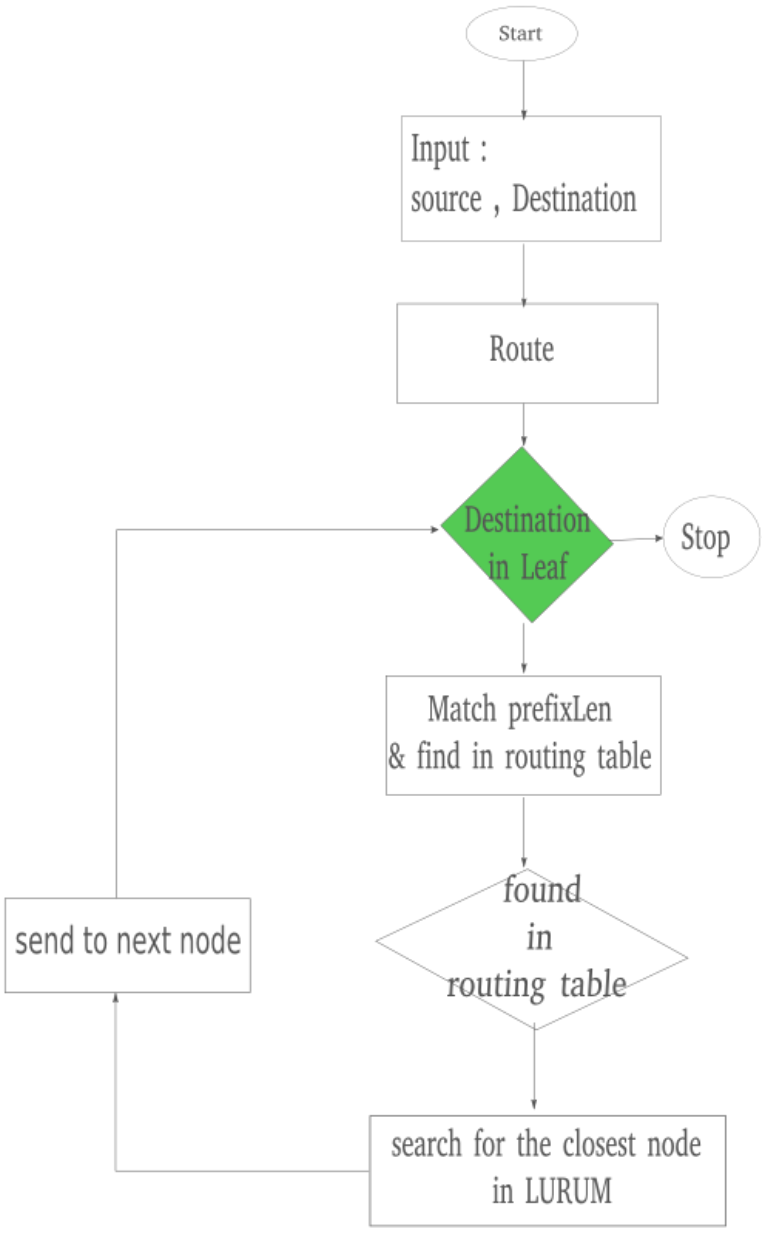


Figure 2: Pastry Routing

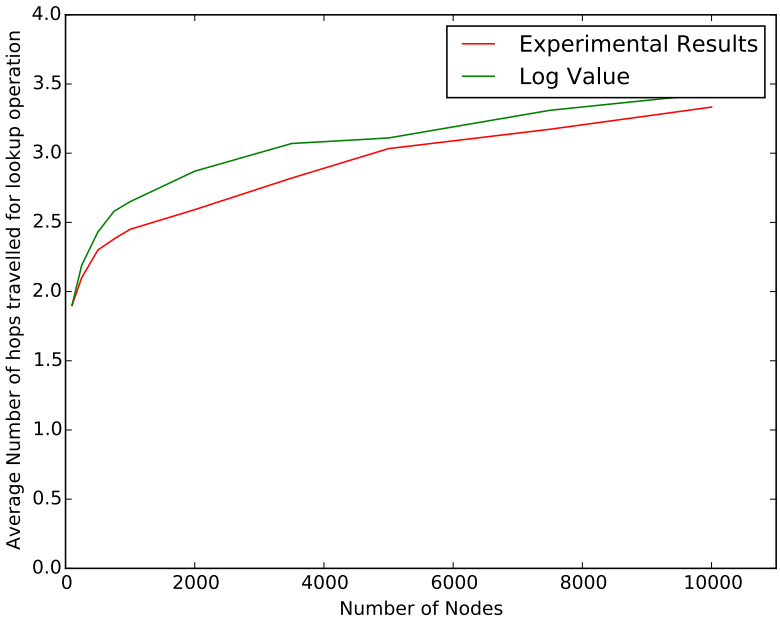


Figure 3: Average Number of Hops for Lookups as a Function of Network Size

The results (Figure 3) show that the number of route hops scale with the size of the network as predicted and is approximately equal to $log_2^b N$

1.1.2 Average number of messages for node addition

Similar to the experiment for "average number of routing hops", the number of Pastry nodes are increased from 100 to 10,000 ; average number of messages are calculated (Figure 4).

While addition the routing is performed with a special join message and later the newly added node sends the state to all the nodes in the route Path, leaf set and neighborhood set. The number of messages shown are the messages used to sent the state. The number of messages increase with increase in the size of the network.

1.1.3 Node Failures

Figure 5 shows the impact of failures and repairs on the route quality. The experiment is performed with 500 nodes (50 failing) to calculate the average number of routing hops in the 3 scenarios: a) no failure, b) failure with no routing table repair, c) failure with routing table repair. Each time 200,000 lookups are performed to calculate the average.

The results show that the average number of routing hops in case of "no failure" is approximately same as that of "failure with routing table repair", though the average number of hops in case of "failure with no routing table repair" is more.

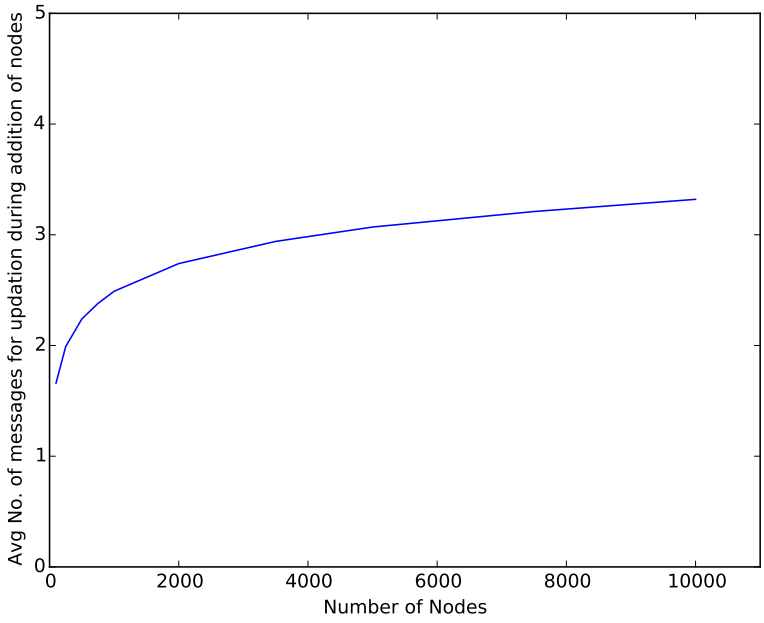


Figure 4: Avg No.of Messages for Updations during Addition as a Function of Network Size

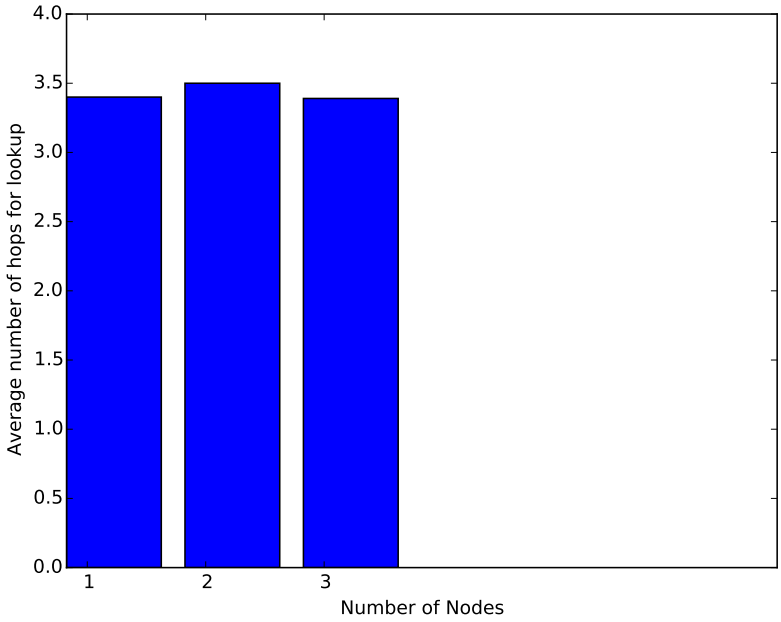


Figure 5: 1-No Failure, 2-Failure with no repair, 3-Failure with repair

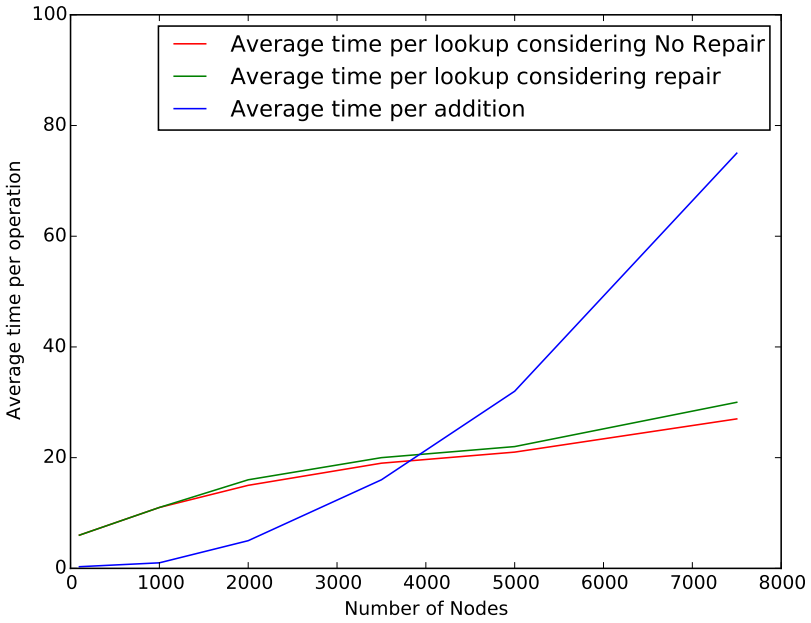


Figure 6: Average time per operation as a Function of Network Size

1.1.4 Time per operation

Lenovo IdeaPad Z570 with Intel Core I5 processor is used to calculate the time per operation. Three experiments were performed (Figure 6): a) the time per addition b) the time per lookup (after 20% deletion) considering no repair b) the time per lookup (after 20% deletion) considering repair of failed entries

The time per operation increases with the increase in the number of nodes which is as expected since more number of messages (or hops) will be transferred. The time per operation in experiment (c) is more than that of (b) , since time will be required to repair whenever a failed node is found.

1.1.5 Efficacy per routing

The number of Pastry nodes are increased from 100 to 75,00 and each time 100,000 trials are performed to calculate the average number of leaf set accesses, routing table accesses and neighborhood set accesses. Along with this percentage success of these accesses is also noted.

The results (Figure 7 & Figure 8) show that the percentage success of leaf set access is very low (almost zero in the beginning) and it increases slowly. This is expected since random diverse nodeIds are taken into account. Also the number of neighborhood set accesses is very low since majority of the time an entry is found in the routing table, which is as expected.

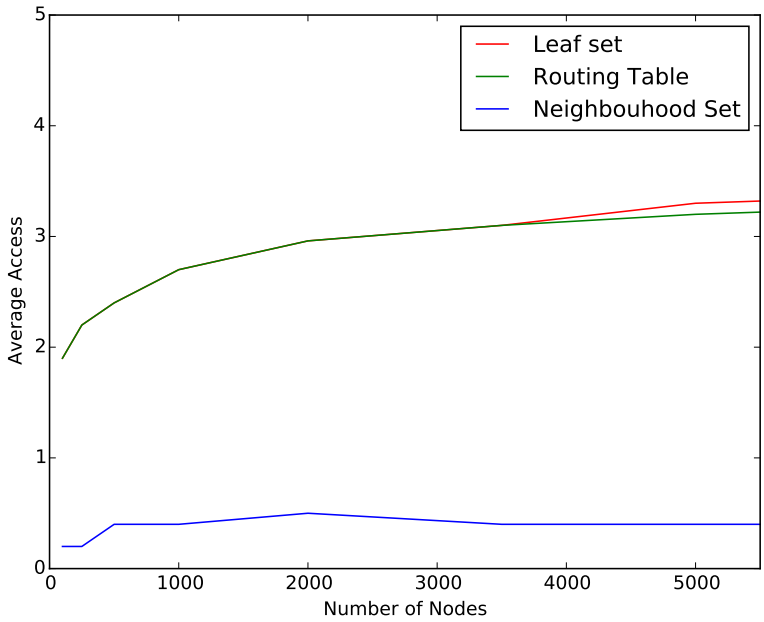


Figure 7: Average Accesses of tables as a Function of Network Size

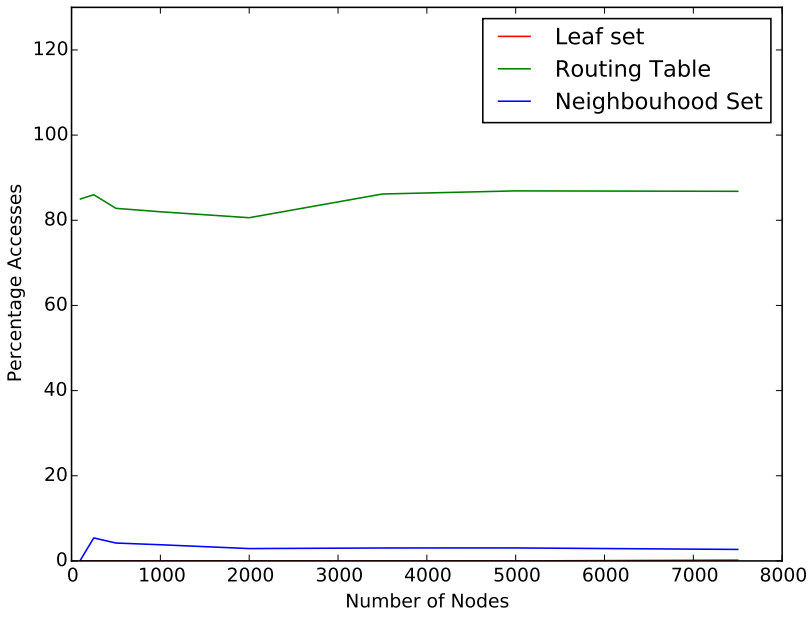


Figure 8: Percentage Accesses of tables as a Function of Network Size

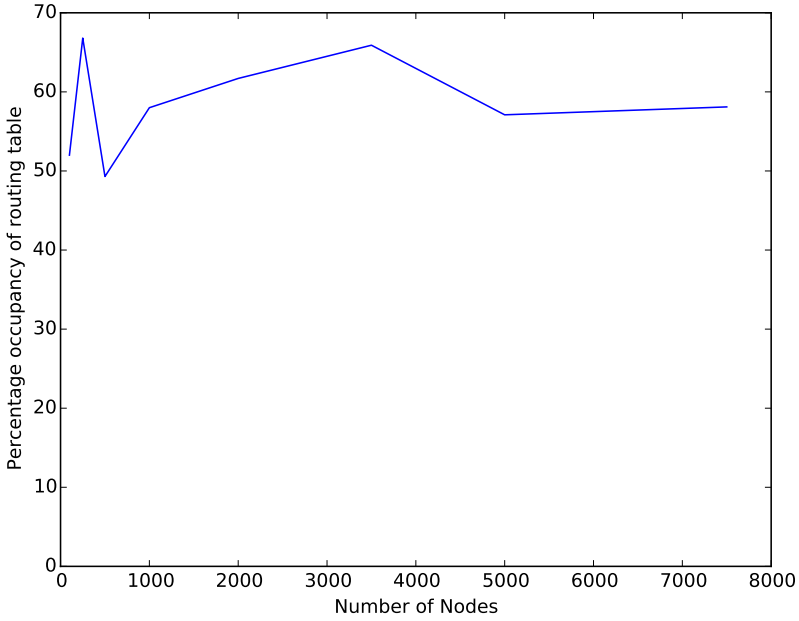


Figure 9: Occupancy of routing table as a Function of Network Size

1.1.6 Routing table occupancy

Routing table occupancy is recorded for different number of nodes. The below graph shows such results.

Figure 9 shows the routing table occupancy calculated assuming the dynamic routing table i.e. the number of rows are less for less number of nodes. Figure 10 shows the routing table occupancy calculated assuming a static routing table with 32 rows.

In the first case the routing table occupancy on average varies between 40-60 % in all cases, this is expected since the routing table capacity also increases. In the second case, the routing table occupancy increases gradually with increase in number of nodes.

2 Chord

2.1 Working of Chord

Chord is a distributed look-up protocol that is used in peer to peer network. The nodes in network are connected in a ring fashion. Each Node is given some id. Each node is responsible for all the keys which falls between id of its predecessor and id of that node itself Figure 11.

Each node maintains 3 things, its successor, predecessor and finger table. Finger table has basically 3 columns. First is start, which states the range of

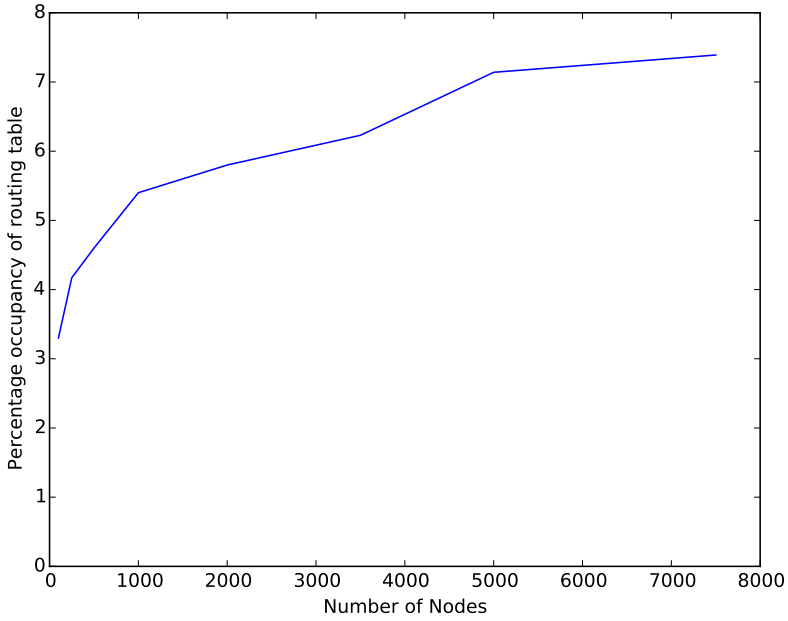


Figure 10: Occupancy of routing table as Function of Network Size

start. It is calculated by adding node id with 2i-1, where "i" is the row number of finger table. Second column specifies the interval which is in between the start of current row to start of next row. Third column stores the successor of the start. For example consider the example given in figure 12.

Whenever any request comes for any id, it checks its finger table to finds the next node to which request should be forwarded. For example in above network, if node 3 wants to find successor of id 1. Since 1 belongs to circular interval [7,3), it belongs to 3rd entry of finger table, so node 3 checks successor at 3rd entry, which is 0. Because 0 precedes 1, nod 3 will ask node 0 to find successor of 1. In turn, node 0 will decide looking at its finger table that that 1's successor is node 1 itself, and return node 1 to node 3.
Whenever any node joins the network, if it is the first node, then it will populate all its entries in the finger table with its node id only. But if it is not the first node, then using multicast, it will contact one of the node which is situated near to it. By making use of that node, it will create its finger table and will initialise other parameters. After that it will also notify other nodes to change their entries if applicable.
Similarly, when any node leaves the network, then also entries need to be changed correspondingly. For that purpose, node does not store just one successor, but some set of successors. Each node keeps on checking whether its successor is alive or not. If not then it will change the successor from its successor list and will also notify it to change its predecessor. Also, it periodically keeps on checking its finger table entries. If any change occurs in the network,

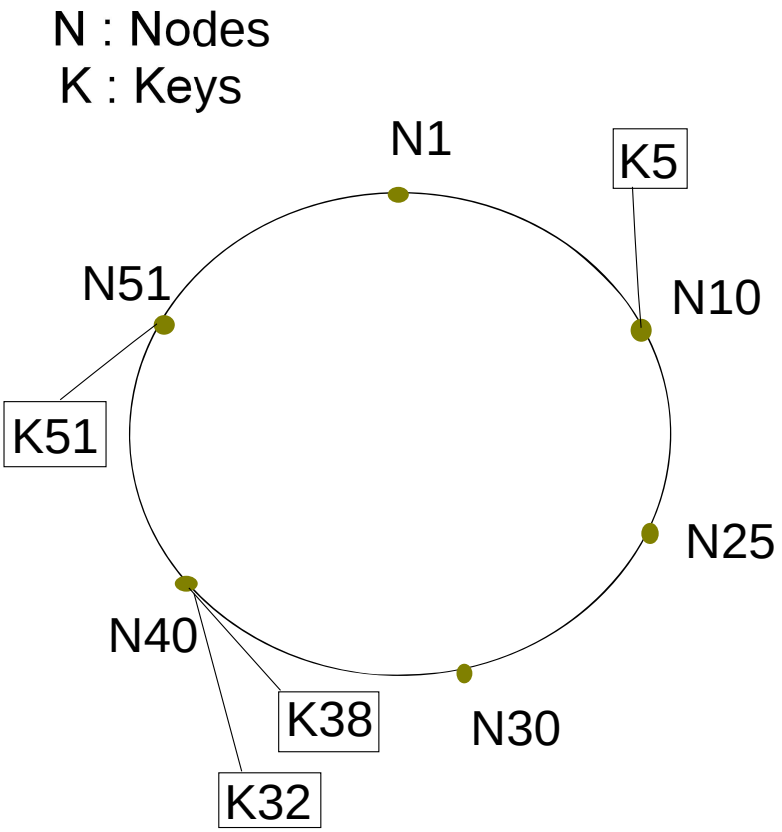
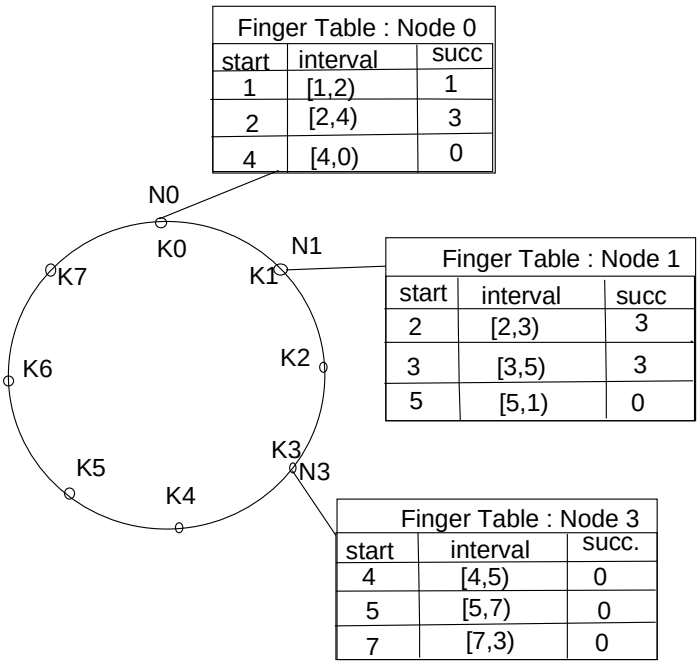


Figure 11: Structure of keys and nodes



Finger Table Entries

Figure 12: Finger Structure

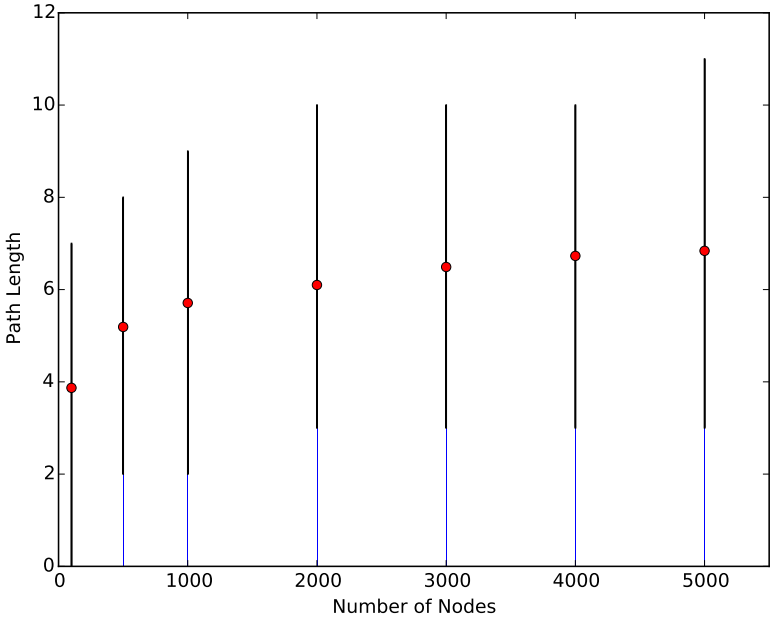


Figure 13: Path Length as a Function of Network Size

then it gets reflected into the table.

2.2 Experimental Results

2.2.1 Average number of hops per lookup operation

Figure 13 shows the average number of hops travelled by any look-up operation along with minimum and maximum value. As from the graph it is clear that average hops travelled by message is approximately O(log N). Also, as we go on increasing the number of nodes, the increment in the average value goes on reducing and value is also quite lesser than the log(N), where N is the number of nodes in the network.

2.2.2 Average number of updates per addition

Figure 14 shows the average (along with minimum and maximum value) number of updates required when any new node is added to the network. As the theory says, in Chord average number is $O(log^2 N)$, where N is number of nodes in the network. So, this result is proved from the above graph.

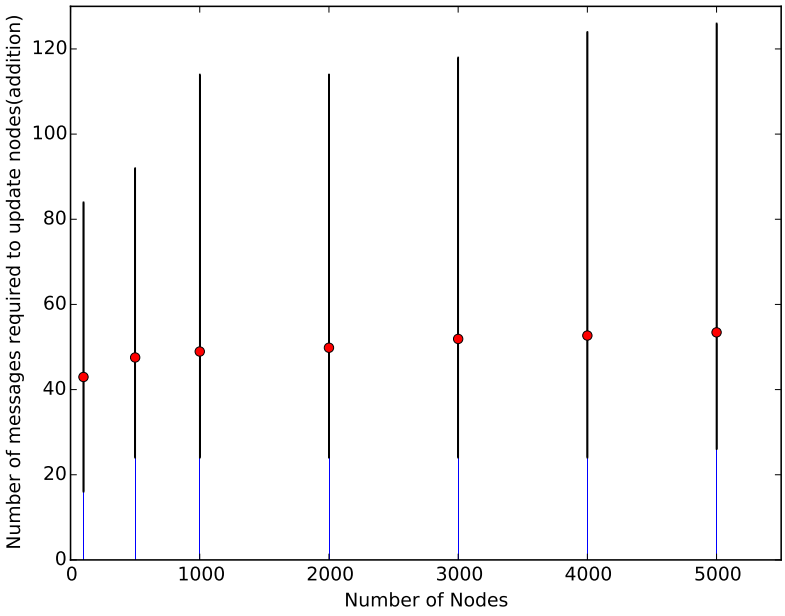


Figure 14: Number of messages required during addition as a Function of Network Size

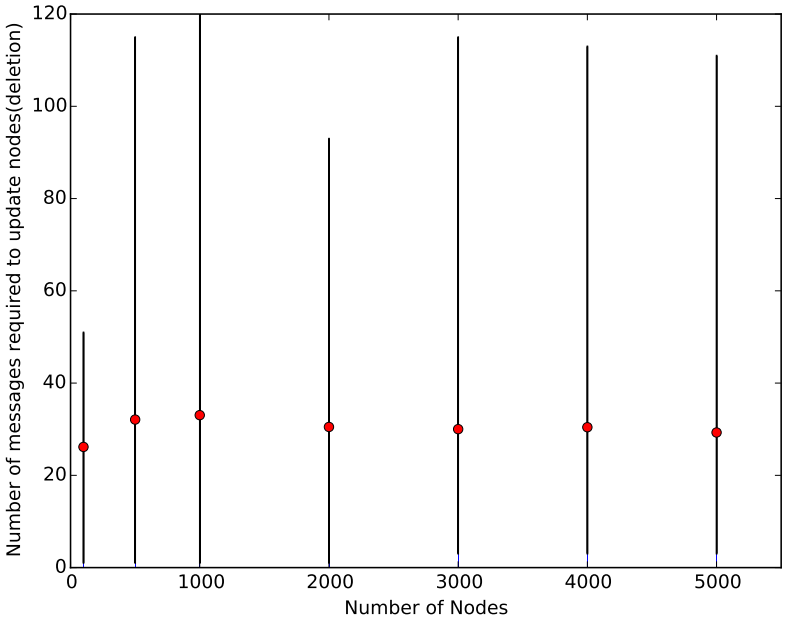


Figure 15: Number of messages sent as a Function of Network Size

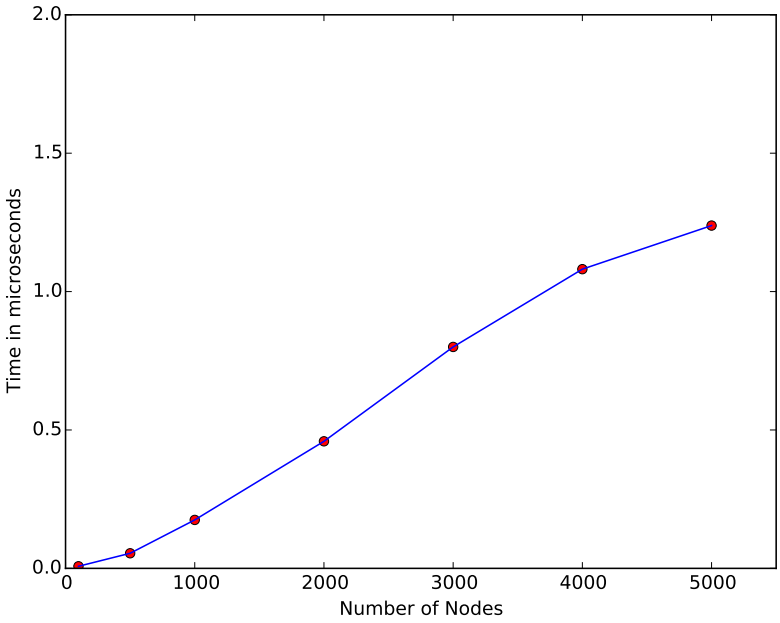


Figure 16: Average time required to perform one addition

2.2.3 Average number of updates per deletion

Figure 15 shows the average (along with minimum and maximum value) number of updates required when any node is deleted from the network. As the theory says, in Chord average number is $O(log^2 N)$, where N is number of nodes in the network. So, this result is proved from the above graph. Although, its average is less than the average of messages required to perform addition.

2.2.4 Average time to perform addition

Figure 16 shows the average time in microseconds required to perform one addition of node in the network on the simulator. Which is increasing in the linear fashion as the number of nodes in the network increases.

2.2.5 Average time to perform deletion

Figure 17 shows the average time in microseconds required to perform one deletion of node from the network on the simulator. Which is also, like addition, increasing in the linear fashion as the number of nodes in the network increases.

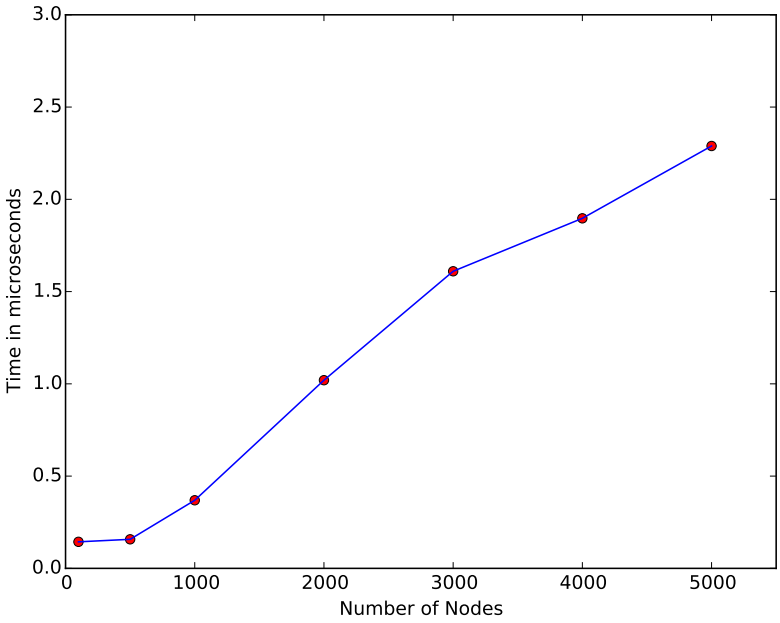


Figure 17: Average time required to perform one deletion

3 Comparison between Pastry and Chord

From the above experimental results, we can state the following statistics. The average time per operation is calculated for both Chord and Pastry. The results show that Chord takes more time per operation than that of Pastry since the average number of messages (or hops) for addition/deletion are more in case of Chord. (chord : $(logN)^2$ and Pastry : log N)
However Chord is more robust than Pastry. In pastry, message delivery will fail, if |L|/2 nodes with consecutive nodeIds fail simultaneously.