Distributed Operating Systems COP 5615

Project 3 Report

Prateek Jain (UFID # 1184-2993)

Mayank Wadhawan (UFID # 5914-8122)

Design:

Chord Implementation Summary:

We have used SHA-1 hash function (160 bit) to hash the values of node id and the key value to be searched. Each node in a chord network will store its m (set to 160) neighbors in its finger table. Additionally, the node will also store its predecessor and successor.

Node Addition:

When a new node (n) joins the chord network, it will ask a bootstrap node (already in the system) to find its predecessor and successor. Then the node will initialize its finger table with the help of bootstrap node. If the bootstrap node doesn't have the information regarding any entry, it will ask its neighbors (and so on). The node which has the information will send the new node (n) that information directly. After the finger table of the new node is initialized, the finger table of other nodes already in the network will have to be updated, i.e. the new node will have to enter in finger table of some nodes. The new node will now be a part of chord network.

Searching a key:

When a search request (hashed) is sent to a node n, the node will check if the key lies between the node's hash value and its successor's hash value. If the key lies in that range, then the node's successor is responsible for that key and the key is found. Otherwise, the node will check its finger table for closest preceding finger and the search request will be sent to that node (and so on).

Please find below the result of the system we dealt with our implementation:

Number of nodes	Average Hops/message
10	1.42
50	3.208
100	4.25
500	6.562
1000	6.63
2000	6.936
10,000	7.265
50,000	7.694

Largest Number of Nodes dealt with our implementation = 50,000 Nodes.

Fault Tolerance Mechanism:

For fault tolerant mechanism, each node watches its successors. When a node gets dead or fails due to some reason, the predecessor of that node gets notified. It changes its corresponding successor. Each Node maintains a list of r successors (r is kept 6). When the node gets killed, its predecessor picks the 2nd successor from the list, makes it, its current successor.

Then it notifies the predecessor recursively to make changes for the killed node in its finger table and replace the entry of the killed node by its successor. When a node is added as a successor to some node n, it copies the successor list of new successor, deletes the last entry of the list and updates the list with current node at the top in the list. Then it notifies the predecessors' recursively to update their respective successor list.

With this mechanism, at any point if a node fails or dies, the next available succeeding node becomes the successor and handles the request for the dead or failed node.

To test the fault tolerance function, all the i^{th} node ($0 < i < totalNodes - 2$) that were multiples of 20 were explicitly killed to simulate the behavior of dead nodes. After the dead nodes, the result was again calculated.

In the Code, to enable the explicit killing of Node and observe the fault tolerance mechanism, set flag ***bonusKillANode*** to True.

Please find below the result of the fault tolerant system we dealt with our implementation:

Number of nodes	Average Hops/message	No. of Nodes killed
50	3.06	2
100	3.784	4
500	4.362	24
1000	5.284	49