Project 3 - Chord - Bonus

Rupayan Das - 63992113
Pirouz Naghavi - 30158432

Getting Started

Running the project

- *dotnet fsi Project2_bonus.fsx <numberOfNodes> <numOfRequests> <numOfDeadNodes>*
- eg: dotnet fsi Project3_bonus.fsx 1000 10 10

Implementation

While setting up the network, we assign random nodes to be permanent dead nodes. No peer in the network knows that these nodes have failed or are dead during the initialization phase.

For the implementation of the failure model, we bring along the successor list which becomes necessary in the case of dead nodes. We also run a checkPredecessor function periodically to check if the predecessor is still alive or not. If a predecessor is dead, we assign the predecessor of the current node to be -1 and let the stabilization function take care of it eventually.

To handle requests which may encounter dead nodes, we set up two dictionaries. The first dictionary takes the message index as the key and the destination node as the value. Once the destination node receives the request it sends the sender node an acknowledgment message confirming it has received the message. When the sender receives the message back from the destination, it removes the item from the dictionary.

Hence, when a message index gets stale, in other words, its difference from the current index increases by a tunable amount, the worker understands that its fingers, successor from successor list, or immediate successor is no longer active and removes it from either finger table, successor list, or successor. This message dictionary is not only for requests, but for any message leaving a node to immediate contact nodes which are: fingers in the finger table, successor in the successor list, or immediate successor. Knowing that each node is constantly stabilizing and updating fingers, we know every node's immediate contact nodes are getting checked and are removed if they are unresponsive or dead. However a request could fail later as it is propagating through the chord even if every node is updating its immediate contacts, so each request is saved in the dictionary with the request string as the key and status as the value. When each request is sent it is added to this dictionary and when it reaches final destination its status is updated along with the total hops of the particular node. To check that every node’s requests reach the final destination, the node checks that status of the request every three seconds or so to make sure request status is set to one as received. If after three seconds the request status is still zero as despite being sent and has not reached the destination, the worker resends the request to make sure every request reaches the final destination and as usual when every request of every worker reaches its destination the simulation is terminated.

Experiments and Findings

For our experiments, we chose the number of nodes to be 1000 and number of requests to be 10. Then we kept increasing the number of dead nodes to determine how resilient our implementation is. And these are the results.

Dead Nodes (%)	Avg Hop Count	Time taken (sec)
1	5.10	9.43
10	4.75	15.65
20	4.48	19.69
30	4.47	16.21
40	4.42	16.36
50	5.08	27.89
60	5.60	65.21

We observed that, beyond 60% dead nodes in the network, the system seems to come to a standstill. If given enough time, the system may terminate after accomplishing all the required amount of requests for each alive node.

Another thing we observed was that when a node joins the network, the dispatcher assigns it a random node in the network for it to execute the findSuccessor function. However, when the number of dead nodes in the network becomes large, the probability of the randomly assigned node turning out to be dead is high as well. Keeping a list of alive nodes and sending it to the dispatcher seems counterintuitive as that will make this a more centralized network. Hence, we provide a list of randomly picked nodes from the network to the joining node. This increases the chances of the joining node to find its successor with the help of an alive node.

Despite this, when the % of dead nodes in the network reaches a substantial number, every node in the list may also turn out to be dead, raising an error in the program.