Implementation of DHT

Goal:

The purpose of this project is going to working on one of the most cited implementation of DHT: Chord. “Chord: A Scalable peer-to-peer lookup service for internet applications” which public and innovate from MIT Laboratory for Computer Science. An important problem that challenges peer-to-peer applications is to professionally locate the node that stores a specific data item. This paper presents Chord, a distributed lookup protocol that addresses this problem. Chord provides support for just one operation: given a key, it maps the key onto a node. Data location can be easily implemented on top of Chord by connecting a key with each data item, and storing the key/data item pair at the node to which the key maps. Chord adapts proficiently as nodes join and leave the system, and can answer queries even if the system is unceasingly changing. Results from hypothetical analysis, simulations, and experiments show that Chord is scalable, with communication cost and the state maintained by each node scaling logarithmically with the number of Chord nodes.

The major job and task of this project is to implement different function of a node class. And we need to implement some interface of a class, and the program need to be tested against a set of number function calls. Also as the description, the keys and node identifiers are 8-bits. And, the Hash functions are omitted in the project for the same reason.

Accomplishment:

First, I build the finger table correctly and print the finger table in the screen when a new node joins.

Second, I finish correct keys are moved when a new node joins the DHT network, and print the keys that are migrated.

Third, I correctly look up keys. Print the sequences of nodes get involved in this procedure.

Fourth, I have implement the Node::leave() correctly.

Process:

For the paper, the author mentioned Chord maps keys onto nodes, traditional name and location services provide a direct mapping between keys and values. A value can be an address, a document, or an arbitrary data item. Chord can easily implement this functionality by storing each key/value pair at the node to which that key maps. For this reason and to make the comparison clearer, the rest of this section assumes a Chord-based service that maps keys onto values.

Chord:

When a node joins the DHT network, the node needs to build its own finger table for routing purposes. To bootstrap this process, the node is provided with another node that is already in the Chord network by some external mechanism. The following member function should be implemented. When a node joins, some keys should be migrated to it as well. As a practical optimization, a newly joined node can ask an immediate neighbor for a copy of its complete finger table and its predecessor. N can use the contents of these tables as hints to help it find the correct values for its own tables, since N 's tables will be similar to its neighbors'. This can be shown to reduce the time to fill the finger table to *O(log N)*. When a node leave the DHT network, the keys maintained in the node should be migrated to another node. Since Chord paper does not describe the procedure to handle node leave in detail, and I try to complete the following function: "void Node :: leave();"

Node Joins:

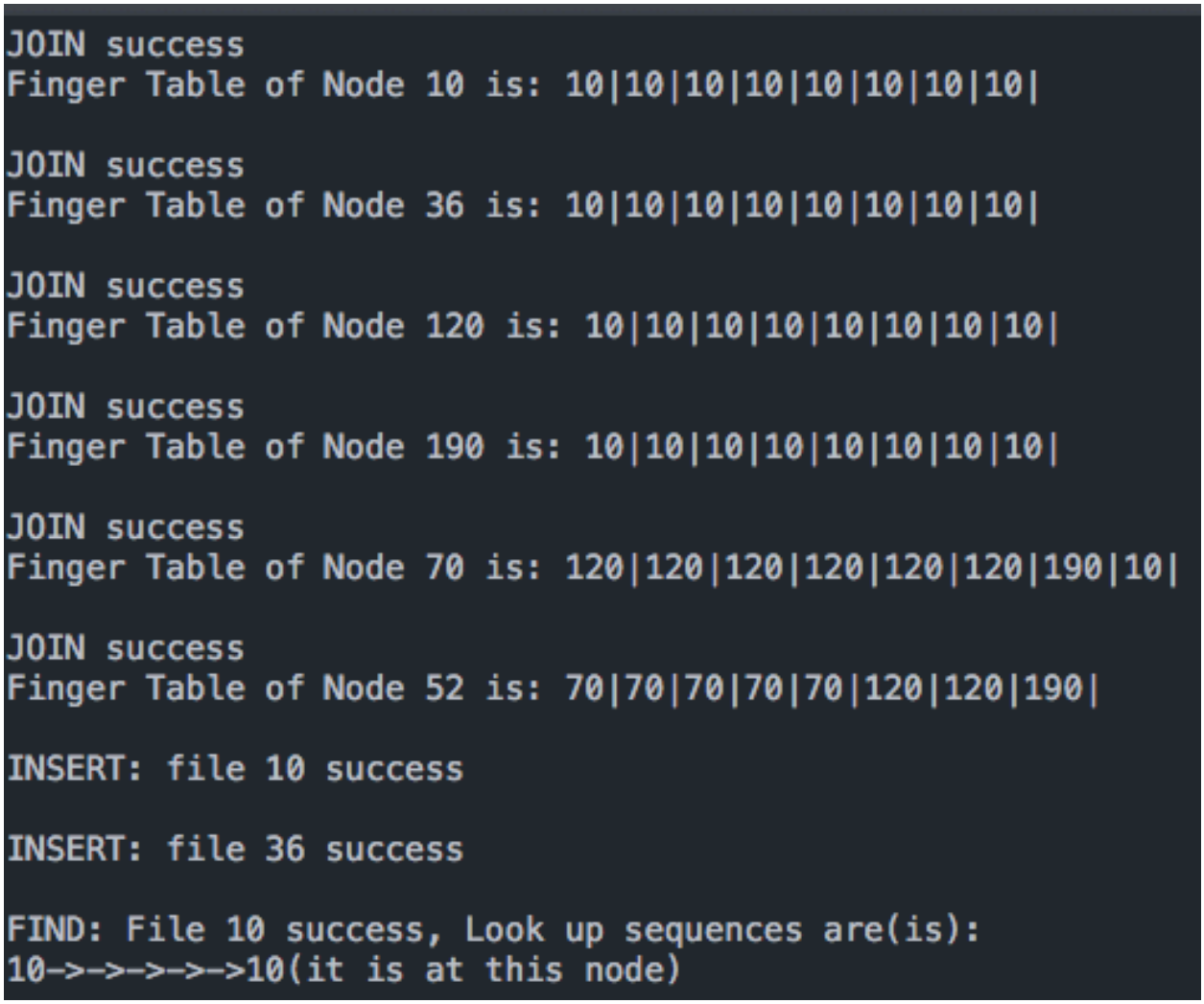
This function should be able to locate the node the keys should be migrated to as well as to notify other nodes to modify the finger table. In order for lookups to be fast, it is also desirable for the finger tables to be correct. To simplify the join and leave mechanisms, each node in Chord maintains a *predecessor pointer*. A node's predecessor pointer contains the Chord identifier and IP address of the immediate predecessor of that node, and can be used to walk counterclockwise around the identifier circle. To preserve the invariants, Chord must perform three tasks when a node "n" joins the network. We should study the paper, Initialize the predecessor and fingers of node "n", then, update the fingers and predecessors of existing nodes to reflect the addition of "n".

In Chord, each node maintains part of the distributed hash table. On condition that the querying key is not maintained locally, the node needs to query the key from the Chord network. Likewise, the node is also able to insert or remove keys in corresponding node. Therefore, the following three functions are crucial too. Because the routing table is distributed, a node resolves the hash function by communicating with a few other nodes. A distributed hash table that maps keys onto values. For example, an application could authenticate data by storing it under a Chord key derived from a cryptographic hash of the

data. Similarly, an application could replicate data by storing it under two distinct Chord keys derived from the data's application- level identifier.

Conclusion:

The Chord protocol solves this challenging problem in decentralized manner. It offers a powerful primitive: given a key, it determines the node responsible for storing the key's value, and does so efficiently. Chord will be a valuable component for peer- to-peer, large-scale distributed applications such as cooperative file sharing, time-shared available storage systems, distributed indices for document and service discovery, and large-scale distributed computing platforms.



If joining nodes have affected some region of the Chord ring, a lookup that occurs before stabilization has finished can exhibit one of three behaviors. The common case is that all the finger table entries involved in the lookup are reasonably current, and the lookup finds the

correct successor in *O(log N)* steps. The higher-layer software using Chord will notice that the desired data was not found, and has the option of retrying the lookup after a pause. This pause can be short, since stabilization fixes successor pointers quickly.

References:

“Chord: A Scalable peer-to-peer lookup service for internet applications” which public and innovate from MIT Laboratory for Computer Science by Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan.