Implementing “Chord: A Scalable Peer-to-Peer Lookup Protocol”, and Applying It For A Distributed File System

Alex Scarlatos, Wendy Zheng

We plan to implement the Chord protocol, described in the paper “Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications” and apply it to a distributed file system. The Chord protocol allows for a distributed network of nodes to be responsible for a set of data, while providing fast lookup time (O(logN) in number of nodes). Additionally, it is able to add new nodes and drop old ones, without requiring excessive adjustments throughout the network.

Chord would be useful for a distributed file system since the client wouldn’t need to know where the data is stored, and could find that data in reasonable time. This is optimal compared to retrieving data from a central location, which is at risk of failing and would receive a high amount of traffic. Rather, any client could insert data into the system and retrieve data out of it. The only tricky part is how any client would know what files are hosted on the network. There are several ways to implement this, and we plan to compare the options in terms of efficiency, reliability and practicality.

1. Use a tracker node. This special node would be the point of entry for adding new data to the network, and would be responsible for keeping track of the name of every file on the network. When someone wants to add a file, they send it to this special node, which adds the key to its own special lookup table, and then sends that data out to the Chord network where it will live. If someone wants to know what files are on the network, they ask the tracker node and it will return its table to them. Some implementations of BitTorrent similarly use a tracker. The problem with this is that this node is a possible single point of failure and will receive high traffic.
2. Retrieve the list of keys dynamically. If a client wants to know all the files on the network, they ask any Chord node that they are aware of, and it will discover this list for the client. There are two ways of doing this. We plan to compare the effectiveness/practicality of both.
    a. The node returns all its keys to the client and asks its successor to do the same, and this request will propagate around the Chord ring in linear time.
    b. The node returns all its keys to the client and will ask every node in its finger table to do the same (there are m nodes in each finger table, and the last entry crosses approximately half the nodes in the network). This will resolve the query much faster, but will result in a high amount of replicated requests.

To test the effectiveness of Chord, we will set up a network on MiniNet, with a preassigned group of processes running Chord. At random times, nodes will join the network, leave the network, and fail outright. We will examine how Chord handles different levels of variability and reliability by adjusting the probabilities of the network changing or failing.

I. Stoica et al., "Chord: a scalable peer-to-peer lookup protocol for Internet applications," in IEEE/ACM Transactions on Networking, vol. 11, no. 1, pp. 17-32, Feb 2003.