Terin Patel-Wilson and Rachel Protacio

CS 426 Project Proposal

10/31/14

**Overall Plan**

For our final project in Decentralized Systems, we will be designing, implementing, and testing the Chord Distributed Hash Table. This project will be divided into three main components: 1) creation of the DHT as a system, 2) an implementation for file sharing among users within the DHT, and 3) mechanisms to combat the typical pitfalls of the Chord algorithm (deletion of nodes, Sybil attacks, etc). Below we present a more specific outline for each part.

**Phase 1: Creation of the DHT**

Throughout this phase, the main focus will be the creation of the distributed hash table with the chord implementation. Firstly a relatively conflict-free hash function must be created so that nodes entering the Chord system can maintain unique positions and identities. From there, we must implement each node’s finger table which will contain log2(n) entries pointing to differing sections of the DHT. Finally, we will create a mechanism by which new nodes can enter the DHT. In order for the joining process to fully functional, we will need to create a direct UI element allowing nodes to become a part of the DHT, functionality for updating finger tables as new nodes enter, and also an implementation to relay key information for the newly added node.

**Phase 2: File Sharing**

By the end of this phase we will have created a functioning, if rudimentary Chord DHT. Building off of phase 1, we will have to create a easy UI interface for users to select files to be uploaded, calculate the hash of the given file, find the correct node within the DHT to keep track of the file (based on the hash), and finally transfer the file to the responsible node. From there, we will also need to create an interface for downloading said files. This will involve a similar ahsing process that we implemented for uploading files, but the end goal will be a search request for the starting node. Finally, in order for this search to work, we will have to make sure that each node precisely knows what information it is in charge of.

**Phase 3: Defending against common pitfalls**

During phase 3, the final phase, we improve upon our basic Chord DHT, and make it more secure from the most common ailments of DHTs in practice. Firstly, we must address the problem of disappearing nodes from the DHT. When a node vanishes, the information it controlled should be preserved; however, with the basic Chord implementation from Phase 2, no information will remain. We will have to create a redundancy implementation that secures the safety of all information within the DHT. Furthermore we will deal with relative memory capacities of nodes (both relative to their fellow peers and relative to the amount of information they are asked to be in charge of). We will come up with a flexible DHT that will allow users of many varieties of memory to participate. Finally, if time allows, we will also create a defense against eclipse attacks that aim to mask one or more nodes from the rest of the DHT. This will require some alteration of the default finger tables to make the overall Chord DHT more secure.

**Terin Patel-Wilson’s individual contribution:**

This first phase of our Chord implementation is focused almost exclusively on establishing the DHT that nodes can dynamically join and that updates in response to this real time expansion. Specifically within this system I will be in charge of three main parts: 1) The creation of a hash function for n = 256, 2) The implementation of a finger table data structure for all nodes, 3) the implementation of a local hash table which contains all nodes and their relative locations, and 4) a system by which each nodes finger tables are updated after a new node joins the system.

1. In terms of creating the hash function, I will implement it so that a total of 256 spots are possible. This will allow the table to be sufficiently big that we can test locally, but also sufficiently small that we will have to implement workarounds for densely populated DHTs. I will start with using a SHA-256 hash, modulo 256 and see how that fares in terms of collision avoidance and work from there.
2. The finger table is integral to the implementation of this DHT. They will contain information necessary for one node to make search queries from the rest of the nodes in the DHT. The finger table data structure will have to contain 3 main parts:
   1. The “Start” point for the search (powers of 2)
   2. The Interval that the corresponding node will be in charge of
   3. The IP address of the node in charge of the given interval.
3. The local hash table will contain hash location and IP address pairs. This will function allow nodes to both ensure that they join the system in a unique location and allow them to refactor their finger tables when necessary.
4. Updating the finger tables will occur after a new node has joined the Chord DHT, and the local hash table has been updated (done by Rachel). This refactoring will be a new reboot of the old finger table to make sure that all search queries will go to the correct node.