TCSS 558 HW3 Design Document

Daniel McDonald, Jesse Carrigan and Sven Berger

November 18, 2013

Introduction

This project is meant to fulfill the basic requirements of the Chord protocol as outlined in the third homework literature. This document will explain the basic design of the entire project, and detailed design of those components necessary for this portion of the assignment.

Design Overview

Current Requirements

In the most basic terms, our implementation of the first part of this three-part assignment must be able to:

- Create the RMI registry all nodes will be known in.
- Create a node.
- Create functionality in the node such that it may:
    - Discover their successor.
    - Notify neighbors of possible layout changes.
- Allow the node to join the network.
- Enable the node to populate its own finger table.

Allow me to explain how Team 2 made this possible.

The RMI Registry

The first thing we needed is the RMI server to keep track of all the nodes. This server is used by the nodes exclusively to get the URI information for a specific neighbor node, and is not used to discover neighbors.

The Nodes

When a node starts, the calling function registers the node with the RMI server (using the hash of its network ID), and calls the node.join function. The joining node first finds the successor node, and if it's the first node in the network it sets the successor and predecessor, and intializes the finger table. It does this by querying a node object passed in from the calling function (in the case of the first node, this is a dummy seed node).

Each node internally knows m for the network, the local node number and the members of the finger table. Successor is always the first finger in the finger table. Since items in the finger table contain references to actual nodes, no additional data must be stored internally for each node. For example successor is called it queries the node itself, and never refers to any internal or central data.

Once the node is running and has set the internal data, it will periodically run a stabilize function which checks the successor and updates it if necessary, which then causes a chain reaction that will update the predecessor and finger table as well. Since every node runs this function independant of any other, network correctness is maintained. If a node should leave the network and the reference is null, this will make the necessary corrections.

Future Work

The final product must act as a distributed hash table, providing some client seamless access to the values stored at the location of the appropriate node hash key. To facilitate this task the clients must be extended to include methods to get and set data, and functions must be added to accomodate redistributing hash values when a node shuts down. Along with this, an internal data structure must be built to allow storage of keys for both the node and whatever virtual nodes are within its range.