TCSS 558 HW3 Design Document

Daniel McDonald, Jesse Carrigan and Sven Berger

December 2, 2013

Introduction

This project is meant to fulfill the basic requirements of the Chord protocol as outlined in the third and fourth 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 addition to the requirements outlined in HW 3, in this assignment our chord protocol implementation must be extended to:

- Allow nodes to gracefully enter and exit the network.
- Allow a client to set data to a node in the network.
- Allow a client to retrieve data from a node in the network.
- Log the activity of every node in individual files.

The basic design from HW 3 was robust enough to continue development without any major architectural re-write. Once instability was introduced to the network, a bug was discovered in how the protocol determined the address range for each node. This problem went undetected due to the way the predecessor and successor were determined for the same node, masking the vulnerability.

Removing Nodes From the Network

Nodes are added to the network the same way as in HW 3. When a node is removed, the node sends a signal to the predecessor indicating it should fix the finger table, and sends messages to all members of the local finger table to do the same. It then sets itself to null. Although it does not remove itself from the RMI registry, any attempt to access it will result in a RemoteException, which is then caught and null is returned. Data can be lost during this event.

Setting and Getting Data on Nodes

A node receiving a Get request will first check to see if the hash of the requested key is within the range of virtual nodes it's responsible for. If it is, it returns the value from that node. If it isn't, the node passes the request on to it's successor, who repeats the procedure. Eventually, if the key is within the node space, the value will be found and returned. If it isn't, null will be returned.

When a node is asked to store a value at a specific key, much the same thing happens. Each node maintains an internal Map of all the virtual nodes within it's range, and whatever values may be stored there. If the request is within this range, the value is stored at the hashed map value. If it is not, the request is sent to the node successor and on and on until the node responsible for the hashed key is found, and the value stored. If two keys should hash to the same value, whatever data is stored first will be overwritten by the second.

Logging Node Activity

When loaded, each node opens a file on the local machine, where the file name is the ID of the node. Whenever an action takes place for any node, it updates the file accordingly with a new record. A record consists of a timestamp, followed be relevant information. This file is never overwritten and holds the complete history of network activity.

An attempt was made to have nodes generate events observable by a special logging class, but this was found to be no better than the existing logging solution, and was ultimately abandoned.

Future Work

Homework 5 requires our chord implementation to be fault tolerant to the sudden loss of a node in the system. This means other nodes must re-learn their finger table automatically AND retain lost data. The requirements state that only one node at a time will ever be lost, so data redundancy could be as easy as storing the key values from either the predecessor node or successor, and then rebalancing this data when nodes join or leave.