Programming Assignment 2:

Design Doc

Brandon Dotson
CS 550-01
11/6/2017

Design Overview

The starting point for this programming assignment was the first programming assignment. Given the task of shifting from a centralized indexing server to a decentralized indexing server, I had to figure out a way to manipulate the library implementing remote procedure calls, rpclib, once more. Using the library's documentation, I found out it was possibly for a client process to bind to multiple server processes. For this reason, the most important aspect of my implementation was having each peer bind to each of the available servers simultaneously. The only caveat was that the configuration file needed to match the number of servers currently running. If not, all the peers would hang indefinitely. By ensuring the configuration file was correct, all of the major groundwork was laid out for the server. The most of the notable changes had to be done on the client-side code. For example, each of the existing cases in my switch statement had to now account for all the other indexing servers. Ideally, this would be done with functions, but the limitations of the rpclib library forced me to resort to the more ungraceful approach of stacking many if-else statements. This means I had to duplicate my code seven times in all situations where my client would have to interact with the servers.

At the same time, I decided to address a few of the loose ends I had last time. Most notably were the fixes for the "quit" and "close" commands that would be used to close clients and (all) servers respectively. I also fixed some issues I had with the "registry" and "obtain" commands. Unfortunately, I was unable to find a proper way to implement tests for these commands, as the nature of my code conflicted with the basic looping of these commands. Fortunately, the search testing is better than before, and I have even implemented a more efficient way of running the tests for all the peers at once. This testing relies on a function called "test_init" to delay all the peers long enough for all the rest to run the same test commands together. This results in much less of a hassle when testing the searches.

Trade-offs

There are quite a few trade-offs this time. This first one that comes to mind is the IP and port selections. Last time, I allowed the servers to use any IP addresses and ports. In the interest of time and simplicity, I hard-coded the same values for the configuration file and the server initialization, though any others would have worked fine had the filestream been programmed to read them.

Another big trade-off comes from how the peers and servers interact. The server does all the remote procedure calls sequentially, so any loops will cause all additional peers to wait. Threads would solve this problem. Lastly, I put all my focus on programming the search tests for multiple servers and peers at the cost of the other two tests.

Future improvements or Extensions

In order to improve on this project, I would focus on several missing features. First, I would make sure the server ports (and possibly IP addresses) could be read from the peers' configuration files. Next, I would program the server to use worker threads, as blocking calls cost the peers a lot of time in some cases. Finally, I would make sure to implement the missing test functions. There were also some missing features from before that could be added. For example, the full implementation networking capabilities for the peers is still missing.