**Hierarchical Peer-Peer Server**

Programming Assignment 3

Abheek Mondal

A205438046

CS 550

Monday, 27 March 2023

In this assignment, you are going to remove the central indexing server and implement a pure distributed file-sharing system. You need to design a hierarchical peer-to-peer (P2P) system. These are the main program files that I created. It also specifies a super peer's host and port to connect to for registering and searching for files on the network.

**weak\_peer.py**

The peer node implementation has several methods, it shares a config.json with the super node which includes its host, port, and file directory for the node.

The methods are as follows:

* load\_files() to load the files in the node's directory into a set
* update\_files() to check for updates to the files directory every 5 seconds and register/unregister files with the super peer as necessary
* register\_files() to register files with the super peer
* unregister\_files() to unregister files with the super peer
* send\_query() to send a query to the super peer and receive a response from a peer that has the requested file, if available
* download\_file() to download a file from a peer
* send\_to\_super\_peer() to send data to the super peer
* handle\_client() to handle incoming client requests for file downloads or queries
* start\_server() to start the weak peer node as a server listening for incoming client requests
* list\_files() to list the files available on the network via the super peer
* user\_interface() to provide a command line interface for interacting with the network (e.g., searching for files, listing available files, registering/unregistering files, adding/deleting files)

**super\_node.py**

The super node implementation has several methods and shares a config file with the weak\_peer node, which includes important information.

The methods are as follows:

* handle\_weak\_peer() to handle incoming connections from weak peers and register/unregister/query files in the network
* handle\_super\_nodes() to handle incoming connections from neighbor super peers and broadcast/query files in the network
* broadcast\_query() to broadcast a query to all neighbor super peers and connected weak peers and check for query hits in connected weak peers
* query\_hit() to send a query hit response to the original sender of a query
* start\_server() to start the super peer node as a server listening for incoming connections

**Deployment Scripts**

There are 2 big deployment scripts that I implemented.

l*aunch\_supernodes.py*

*all2all.py*

The first code defines a list of configuration files for the super peer nodes, superpeer\_configs. It then loops through each configuration file and starts a new subprocess using subprocess.Popen(). The subprocess runs the super\_node.py script with the configuration file as an argument. It is used to start multiple super peer nodes simultaneously, each with its own configurations.

The second one is more intensive. It is an extension of the P2P file sharing system. It deploys a set of super-peers and weak-peers and then simulates a set of peer nodes (clients) that issue search queries to find a file in the network. The code measures the average response time for different numbers of peer nodes and plots the results. The code first deploys the super-peers and weak-peers using their respective configuration files. It then waits for a few seconds to allow the peers to start up. Next, it defines a set of functions to simulate the peer nodes and measure their response times. The send\_query function sends a single search query to a specified peer and returns the response time. The measure\_average\_response\_time function sends multiple queries to a specified peer and returns the average response time. The client\_task function is used to simulate a single peer node and measure its response time. It uses the measure\_average\_response\_time function to measure the average response time for the peer. The measure\_clients\_response\_times function simulates a set of peer nodes with different numbers and measures their average response times. It uses threading to run the peer simulations in parallel. Finally, the code calls the measure\_clients\_response\_times function with the specified parameters and prints the results. It then plots the results using matplotlib.

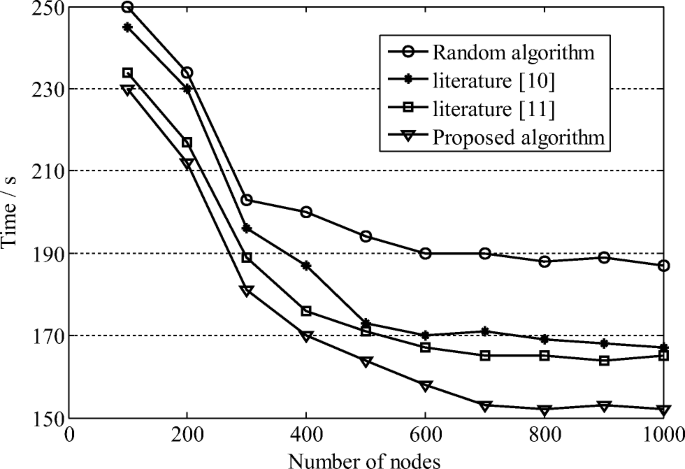
**Time Analysis**

I was having problems with this. I simply ran out of time, and I truly wish that I had started this project earlier. However, the weak peer node had connected with the supernodes, and was able to successfully make queries to the supernode, in regard to specific files.

Text

Description automatically generated

For all to all net, I wish I had more time, as it was able to generate results, and a graph. It was running into issues in regard to its sockets. However, I expect the graph to look similar to this.



Where the time taken per request is exponentially related to the number of nodes present. This is due to the excess of nodes present means that the search query has to echo through all of those nodes individually, causing the query to be more and more delayed.

**Conclusion**

Essentially, my code worked in terms of executing, and connecting to the server. I have problems with socket requests. It is also able to register and unregister itself from the super peer node. Also, I did not receive any feedback from the first programming assignment so that I knew how to update this assignment.