Design Doc:

In this Programming assignment, the previous assignment code has been used. The current super peer node is similar to the previous indexing server with some changes that will be detailed later. At the same time, the weak peer node is similar to the previous node of the assignment 2 with some minor changes.

Therefore, the only code details that are going to be discussed are the changes with respect to the previous assignment and not all the code implementations.

Super Peer Node:

To get the super peer running, several arguments must be passed down in the python script:

- The IP address of the super peer node
- The Port of the super peer node server
- All the ports of the neighbors' super peer nodes.

Therefore, the setup is fixed. At the beginning, all the super nodes passed in the arguments are going to stablish a connection between them and, until the connection is not properly set, the super node is not going to accept weak nodes connections. At the same time, those connections are going to be saved in the values of the dictionary of super_nodes.

The super node is going to have 4 dictionaries:

- Reg_nodes: all the peer nodes that are registered for P2P downloads. This dictionary is as follows:
    - Each key corresponds to the address of a client (IP and port)
    - Each value is a tuple which corresponds to:
        - The port of the client's server (so that it can be shared to other clients to establish the connection to download files)
        - All the files that the client hosts.
- Super_nodes: all neighbor super nodes. The key corresponds to the super node server port and the value will be the socket in which the connection has been stablish.
- Weak_nodes: the weak nodes that are connected to the super node. The key corresponds to the weak node server port and the value will be the socket in which the connection has been stablish.
- Files_prop: The information of the files asked with the command query to be able to perform a back propagation. The key corresponds to unique identifier of the message and the value to the server port of the node that sent the query (the previous server port).

The weak peers register automatically when they get connected and, when they register, the connection with their server is also establish and save in the weak_nodes dictionary.

At the same time, two more commands have been created:

- Query: It has replaced the previous download command. The format of this message is as follows:

  Query <file> TTL UID own_server_port

  The first query message is created in the weak nodes, but the super nodes also keep updating and broadcasting this command between them. It contains the following information:

  TTL: Time to leave. It will be either 1 or the total number of super nodes minus one. It is setup in the weak node and every time the query is broadcasted from a super peer to its neighbors, its value is decreased by one.

  UID: Unique identifier of every query message to be able to perform the back propagation correctly. It is created in the weak node and it is kept unchanged.

  Own_server_port: It is the port of the server of the peer who sends the query message. This allows the back propagation of the messages with the queryhits.

  Hence, when a super node receives a query message it does the following sequence of activities:

  1. It checks if the file requested is hosted in one of its weak nodes registered for P2P downloads.
     If a weak node hosts the file, it directly sends a queryhit message back to the node (weak or super) that asked for it.
  2. It saves the server port of the query message in the dictionary files_prop with its unique identifier as the key.
  3. If the TTL is greater than 0, it subtracts 1 from its value and sends the same query message with the new TTL and its own server port to all its neighbor super nodes except for the one that sent it.

- Queryhit: this command has the following structure:

  Queryhit UID <list_nodes>

  UID: The same unique identifier as the query message that generated the queryhit.

<list_nodes>: List of all the weak nodes registered in a super node that host the file requested with the query message.

It gets autogenerated when a super node finds at least one registered weak node that host the file requested with the query message.

To send the queryhit, the super node performs the following activities:

1. Gets the server port to which it has so send the message searching the unique_id in the files_prop dictionary.

2. Checks if the server port is a super node or a weak node by looking up both dictionaries.

3. Retrieves the connection (socket) store in the dictionary of the server port and resends the same queryhit message.

Weak Peer Node

To get the weak peer running, 5 arguments must be passed down in the python script:

- The IP address to connect to the super node.
- The Port of the super node
- The topology of the network (‘all’ or ‘linear’)
- The Port of the weak peer server
- The folder in which its files are kept

As with the super peer node, the weak peers will have a dictionary called files_pending with the following structure:

1. Key: Unique identifier of the Query message
2. Value: Array with the following values:
    a. Position 0: Flag that determines if the message has not yet been download.
    b. Position 1: time when the query message was sent.
    c. Position 2: sum of the times it took to get each queryhit for the query message.
    d. Position 3: total number of queryhits for the query message.

As in the programming assignment 2, the weak peer node is composed of a client and a server, however, there has been some changes.

First, the command ‘download’ has been changed names to ‘query’. When the client sends to the super node the query message, it just writes: query <file>.

However, a unique identifier is generated randomly (uuid4()) and the message that is sent to the super node is like this:

Query <file> TTL UID own_server_port

Where TTL is 1 if the topology is ‘all’ or the total number of super nodes - 1 if the topology is ‘linear’. This has been selected to prevent an unreasonable large amount of query messages being broadcasted indefinitely through the network. However, every query message has to reach all the super nodes at least once. Hence, in an all-to-all topology, it is enough with a TTL of 1 and in a linear topology, the worst scenario would be the broadcast of a query message from one end to the other.

Finally, the unique identifier of the query is store as a key of the dictionary files_pending with its values as:

[True, initial_time, 0, 0].

Also, a thread is deployed to run the function print_query_time with the unique id of the message as a parameter to print the average time to receive all the queryhits.

On the other hand, there is a thread that acts as a server that keeps listening for possible connections. Whenever it receives a new connection, it creates a new thread calling the function client_manager.

This function waits until it receives some information that can be:

- The request of a file from another weak peer: it sends the file to that peer as it was done in the Programming Assignment 2.

- A queryhit message from the super peer: the weak peer gets the unique id from the message and retrieves the information stored in the dictionary pending_files:

    - It gets the time when the query message was sent
    - It computes the difference in time and adds it up to store the sum of the time of all the queryhits received for that specific query.
    - It increments by one the total number of queryhits received for that specific query.
    - If the flag is still True (just for the first queryhit of each query):
        - it decodes the queryhit message to get the list of nodes that host the file
        - Then, it selects one of the nodes at random
        - It turns the Flag value to False, so that the file does not download again with successive queryhits.
        - It executes a thread to connect and download the file from the peer selected as it was done in the Programming Assignment 2.

Finally, there is a thread running for every query message executing the function print_query_time. This function waits 0.5all seconds and then, it retrieves the information stored in files_pending for the query message to print the average time taken for all the queryhits as follows:

Sum_times / number_queryhits