Design Doc:

In this Programming assignment, the Programming assignment 2 code has been used. However, the code has been modified with some improvements of Assignment 3 and the parallel download of the files.

Therefore, the only code details that are going to be discussed are the changes with respect to the Assignment 2 and not all the code implementations.

Indexing Serve:

The Indexing server is going to have, now, 3 dictionaries:

- Reg_nodes: all the peer nodes that are registered for P2P downloads. This dictionary is as follows:
    1. Each key corresponds to the address of a client (IP and port)
    2. 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.
- 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 queryhit back to the peer. The key corresponds to unique identifier of the message and the value to the server port of the peer that sent the query.

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

At the same time, the command download has been modified to query and works as follows:

The format of this message is as follows:

Query <file> UID peer_server_port

UID: Unique identifier of every query message to be able to send back the qeryhit correctly. It is created in the peer nodes and it is kept unchanged.

peer_server_port: It is the port of the server of the peer who sends the query message to be able to send back the queryhits.

Hence, when the indexing server receives a query message it does the following sequence of activities:

1. It checks if the file requested is hosted in the peers registered for P2P downloads.
2. It saves the peer node's server port of the query message in the dictionary files_prop with its unique identifier as the key.
3. If at least one of the peers hosts the file, it sends a queryhit message back with a list of all the peer nodes that host the file (and the files sizes).

At the same time, the command get_files_list has been modified so that it returns not only all the files available for download, but also their sizes.

Weak Peer Node

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: Name of the file that has been queried.
    b. Position 1: A dictionary with the following values:
        1. Key: Peer server port to which a chunk has been requested.
        2. Value: Another array as:
            a) The data of the chunk received by the client.
            b) The timestamp (current time) when it was received.

Therefore, the peer works as follows:

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> UID own_server_port

Finally, the unique identifier of the query is store as a key of the dictionary files_pending with its values as (file_name and an empty dictionary):

[file_name, {}].

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 chunk from another weak peer:
    1. It decodes the message received, which contains: the name of the file requested, the offset value and the chunk size it has to send.
    2. It reads the file and just keeps the bytes between the offset and the offset plus the chunk size (ignoring the rest of the bytes of the file).
    3. It computes the md5 hash of the chunk and sends it to the client for a final verification of the integrity of the download.
    4. Finally, it sends the chunk of the file.

- A queryhit message from the indexing server:

1. The peer gets the unique id from the query message and retrieves the file name stored in the dictionary pending_files.
2. It gets the list of nodes that host the file and the file size.
3. The number of chunks in which the file is divided are the number of peer nodes that host the file, and the chunk size is the size of the file divided by the total number of chunks.
4. Then, each chunk is downloaded by a different peer node in parallel with threads. Also, these threads are stored in a list with the following structure:
    - Position 0: The server port of the peer that is going to send the chunk.
    - Position 1: The actual thread that calls the download function in parallel.
5. The peer waits till all the threads are done with the join function and the data stored in the pending_files dictionary of each chunk is added up together to form the final file.
6. Therefore, if the length of the downloaded file matches up with the file size that was received from the indexing server, the peer writes the file in its folder and sends an update_files_list to the indexing server.

Finally, the download function that is called by the threads was added 3 arguments apart from the peer node address and the file name:

- The starting position of the chunk (offset).
- The chunk size.
- The UID of the query message (to store the data from all the chunks in the dictionary).

Therefore, the download function stablishes the connection and sends the name of the file, the offset, and the chunk size. Then, it receives first, the md5 hash of the chunk and then, the actual chunk of the file.

It saves the file data (in bytes) in the files_pending dictionary with the timestamp (current time) when it was received and computes the hash to check the integrity of the download.

Some of the possible improvements and extensions could be:

- Implementing a thread that is constantly checking the files in the client folder, to notify the indexing server every time one file gets removed or added automatically.
- Implementing a login system for the peers.