Programming Assignment 3

**Instructions**

* Due by 11:59pm of 03/25/2024.
* Late penalty: **10%** penalty for **each day late**.
* This is an individual assignment, although you may work in groups to brainstorm on possible solutions; your code implementation, report, and evaluation must be your own work.
* Upload your assignment on the **Blackboard** with the following name:

**Section\_LastName\_FirstName\_PA3.zip**

* Please do **NOT** email your **assignment** to the **instructor** or **TA**!
* All the submitted document files should be **PDF/Doc format** files. The **handwritten file is not accepted**.

Overview

In PA2, you implemented a simple centralized file-sharing system. 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.

Detailed Description of Assignment

In this assignment, you need to design a hierarchical peer-to-peer (P2P) system.

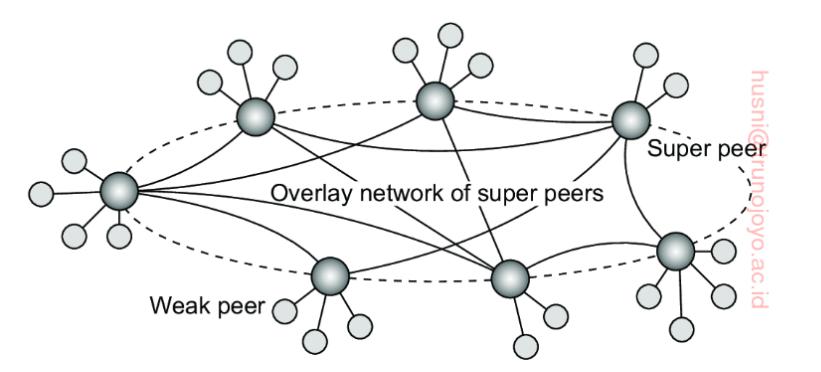


Figure 1. A hierarchical organization of nodes into a super-peer network

As the topology shown Figure1, **a super-peer** acts as a proxy or a broker in the peer-to-peer network. **A weak peer** keeps only connection to a super-peer. Each super-peer is like an indexing server in PA2. To keep things simple, we assume that the structure of the P2P network is **static**. This means that your network is initialized statistically at startup time.

The **weak peer node** performs **at least** the following tasks:

* Listens for connections from a port which can be specified either by Configuration file or Passed Arguments
* Hosts files in its directory, the file directory can be specified through Configuration file or Passed Arguments
* Its associated super-peer is specified either by Configuration file or Passed Arguments.
* As a server, it needs to register all the files it holds to its associated super-peer when it starts up. (You could consider the super-peer as an indexing server)
* As a server, if the node gets an exit signal, it must unregister from its super-peer.
* It should support an automatic update mechanism. If the peer node downloads a new file or deletes an existing file on its directory, it should update this information to its super-peer in time.
* The maximum number of weak peer-node attached to super-peer node is **5**.
* As a server, it could accept download requests and then send the requested file to the request node. The server keeps a record of download requests, even if no file is found for download request.
* When having initialized the P2P network, the peer node, acting as a client, searches for files by issuing a *query* request to its super-peer to check where the file is located in. If the file exists in other peer nodes, then a *queryhit message* is sent from the super-peer. After receiving a *queryhit message*, the client could download the file directly from the peer node.
* **Peer Node output**: The node maintains a log of the nodes that it connects to and connected to it, the files it sends to/gets from which node, the amount of data sent, the time it took to download a file and possible errors faced.

The **super-peer node** performs **at least** the following tasks:

* It maintains its connected peers and neighbor super-peers either by Configuration file or Passed Arguments
* It provides the following operations for a weak peer node to register/unregister with it and upload their files
  + Register a weak peer node.
  + Unregister a weak peer node.
  + List the files information’s that files in peer nodes: file size.
  + Update its files:
    - Add a new file
    - Delete an existed file
* It supports to receive the query message from one of its connected weak peer nodes and then broadcast it to all its **neighbor** **super-peers** in order to check where the request file is located in.
  + Each query message needs to carry a globally unique message ID with it.
  + The *queryhit* message must carry the same message ID as the corresponding query message in order to be propagate back correctly.
  + The super-peer initially checks if this file exists in other weak peer nodes connected to the super-peer itself. If the file exists in other weak peer nodes, then a *queryhit* message is sent to the request weak peer node.
  + The *queryhit* message must carry the same message ID as the corresponding query message in order to be propagate back correctly.
  + The super-peer forwards the query message to all of its neighbor super-peers through broadcast/back-propagation method no matter the request file hit or miss in itself.
  + Each neighbor super-peer checks if the specified file exists in its connected peers and responds with a *queryhit* message if the file exists. Similarly, the *queryhit* message will be propagated back to the original sender by the reversed path of the *query* message.

* + Each query message needs to carry a time-to-live (TTL) value which is decreased at each hop from one super-peer to another super-peer. A query message from S is broadcasted to all its neighbor super-peers and relayed by each receiver until its TTL value decreased to 0.
  + Each super-peer also keeps track of the message IDs and its upstream peers, that is where the messages are sent from. You could use a map to store the [message ID, upstream peer ID] pairs. To prevent query messages from being forwarded infinitely many times
* **Indexing server output**: Log all operations done to the Indexing server node. It must track the nodes that connect to it, and what operation they do.

Evaluation

1. Deploying at least 3 super-peers and each super-peer is connected to 1-3 peer nodes. They can be setup on the same machine or different machines. Each peer node has its file directory.
   1. Ensure you can transfer one file properly.
   2. Ensure multiple peer nodes could simultaneously upload and download files.
2. Evaluate the system performance for two topologies for super-peers and then compare their performance. The tested two topologies are tree topology and all-to-all topology. You could initialize the topology by assigning the neighbors for each super-peer.



Figure 2. Two topologies for super-peer network

* 1. Measuring the average response time per client query request when varying the number of clients (eg, the number of peer nodes send the query request) according to the tree topology as shown in Figure 2(a).
     1. Deploy at least 9 super-peers and each super-peer is connected to 1-3 peer nodes. Different peer nodes could hold repeat files.
     2. Measuring the average query response time seen by a client, since there may be multiple results for each query request. Using your own technique to decide when the last query result should come back. For example, you could define a cutoff time, and wait until that time and then compute the result.
     3. Repeat this measurement 300 times and get the average.
     4. Do the same calculation by changing the number of clients issuing queries, such as 1, 2, 4, 8 and so on.
     5. Graphing your result and justify your conclusion.
  2. Measuring the average response time per client query request when varying the number of clients (eg, the number of peer nodes send the query request) according to the all-to-all topology as shown in Figure 2(b).
     1. Performing the same experiment as sequence a by only changing the topology to all-to-all topology.
     2. Graphing your result and justify your conclusion.
  3. Compare the system performance got in sequence a and b under the same system size, graph the comparing result and justify your conclusion.

Submission Information

When you have finished implementing the complete assignment as described above, you should submit your solution to the blackboard. Each program must work correctly and be well documented. You should hand in:

1. **Source Code (25 points)**: You must hand in all your source code, including with in-line documentation.
2. **Makefile/Ant/requirements (5 points)**: You must use Makefile or Ant to automate your programming assignment compilation or a requirements file to download any dependencies. If using external libraries (mainly for compiled languages) put them in a folder marked external. **Note:** The external library cannot do the heavy lifting of the assignment tasks for you. If in doubt reach out to the TA.
3. **Deployment scripts (10 points)**: You must provide a deployment script for the different nodes.
4. **Readme (10 points)**: A detailed manual describing how the program works. The manual should be able to instruct users other than the developer to run the program step by step.
5. **Compiles Correctly (10 points)**: Your code must be compiled in a Linux environment.
6. **Output files & Performance Evaluation** (25 points): A copy of the output generated on running your programs for each of the evaluations. For (2) in evals, provide the output for each level scaled. Write a report with your findings along with the graphs.
7. **Design Doc (15 points)**: You must write about how your program was designed, what tradeoffs you made, etc. Also describe possible improvements and extensions to your program (and sketch how they might be made). Separate from report.
8. Please structure your assignment root folder as follows:
   1. Code: for your source code and make files and deployment scripts. Your file structure in here is up to you but please make sure it is clean.
   2. Docs: for all written documents, report, readme, design doc, etc.
   3. Out: for all output files from your peer nodes and super-peer nodes, named indicating which specific evaluation.
   4. Misc: for other files not mentioned specifically.
9. Please put all of the above into **one** .zip file, and then upload it to the **blackboard**. The name of .zip should follow this **format**: “Section\_LastName\_FirstName\_PA3.zip”.

Submission checklist:

* Your source codes
* Makefile or equivalent specified above
* Deployment scripts
* Readme
* Your Evaluation output
* The output files of your peer nodes and Indexing server node as specified above
* A Design Document