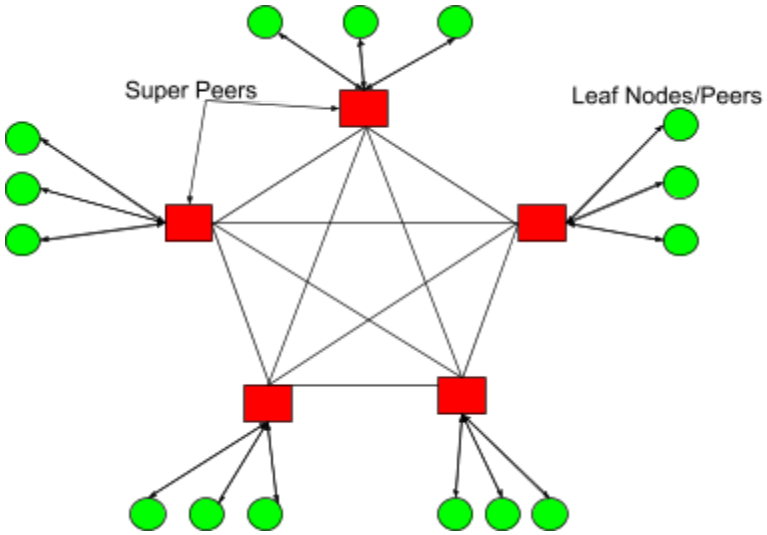
CS550 Programming Assignment 2 (PA#2)

A Hierarchical Gnutella-style P2P File Sharing System



[Figure 1]

Program Design

We use Java RMI (Remote Method Invocation) in our implementation. This is because of its usability for P2P and similar distribution models.

As explained below there are 4 steps to our implementation :

1) Instead of a central indexing server, we set up multiple super peers that have index registers to maintain all information for messages being passed/received.

These super-peers will receive "**query**" messages from it's Peer/Leaf Nodes. They will also receive multiple "**query-hit**" messages from fellow super-peers. The "**query**" and "**query-hit**" messages are forwarded/processed by the super-peers depending on the nature of the request that initiated them.

2) Once the super-peers are set up, we implement leaf-nodes in our architecture. These leaf-nodes are responsible for storing local data, sending or receiving multiple query hit messages These leaf nodes will also register their parent super-peers when being set up.

3) After setting up our super-peers and their leaf nodes, we first implement the All-to-All architecture as shown in Figure 1. In this topology, we connect all peers with their parent super-peers and the all the super-peers with each other.

4) Finally, we implement Linear Topology where the super peers are connected sequentially with the next super-peer. The queries and responses are sent sequentially in this case, unlike the All-To-All Topology.

Implementation Steps

Defining a Remote Interface :

We define a remote interface for the index server named "SuperPeerInterface" which extends the Java RMI RemoteException interface. We declare four remote methods in the interface. The first method is to register or delete the registered files stored in the index. The second is to search for a file. The third method is for sending queries. The fourth method is to check if our queries have been received or not at the intended destination.

All the methods are accessed remotely by super-peers and peers using Java RMI.

Implementing the Super-Peers :

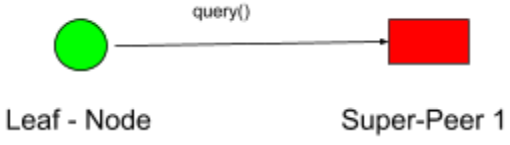
Each SuperPeer contains a registry similar to the Index Server in our previous project to do three things: create an instance of the remote object, bind the instance to a name in the registry and export the remote object. This registry is created using a hashmap data structure and will store all the information regarding its current leaf nodes along with an ID to identify itself.
The second method is a query method that will record all the requests from its leaf nodes or forwarded requests from other Super-peers. In this method, the super-peer will search its registry which is a HashMap table mentioned above using a search function. If the entry is found the details are extracted using the message ID. The super-peer then calls another method implemented in the Leaf Node to return the requested data. If no entry is found, the query is forwarded to other super peers or the next super peers based on the topology being used.
The third method is the queryHit method. This method is called after the super peer searches it's own registry for the requested file or another super peer forwards a query to the current super-peer. It is responsible for managing the TTL of a query (for how many hops is a query valid) and can send a reply to the leaf node who has initiated the request by looking at the registry table and calling its corresponding query hit method. This is done by obtaining the registry name of the leaf node and its ID.
A property file will contain the information regarding the super-peers and their leaf nodes. The user can modify this file to change the topology being implemented.
The data structure we use is a Multi-Values Map i.e. each key has an array of values. These values are the message ID, time to live(TTL), file name. requester's ID and requester's port number. Using these values the super-peers, as well as the leaf-nodes, communicate with each other.

Implementing the Leaf Nodes/Peers :
We implement the leaf nodes next. The leaf nodes are the primary requestors in our architecture and use two methods to request for files and reply to incoming requests. These methods are the same as those implemented in the super-peers namely Query and QueryHit.
A leaf node gets the registry details from the registry by lookup and calls the query method. Once the method is called a timeout period is set which is basically the lifetime of a request. Once the TTL expires, the request is no longer valid and gets discarded.
The QueryHit method of the leaf node also manages the TTL and matches the message ID of the request with the response received from the super peer as a validation. It then extracts all the details and stores them in a buffer. The buffer is implemented as an Array List data structure. The leaf node then asks the user to select the peerID from all the replies received to download the desired file.

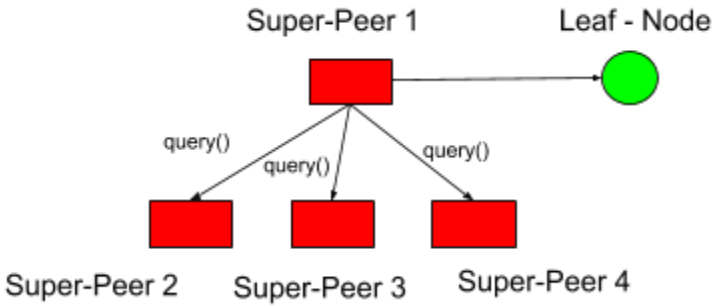
Flow Diagram

All-To-All Topology

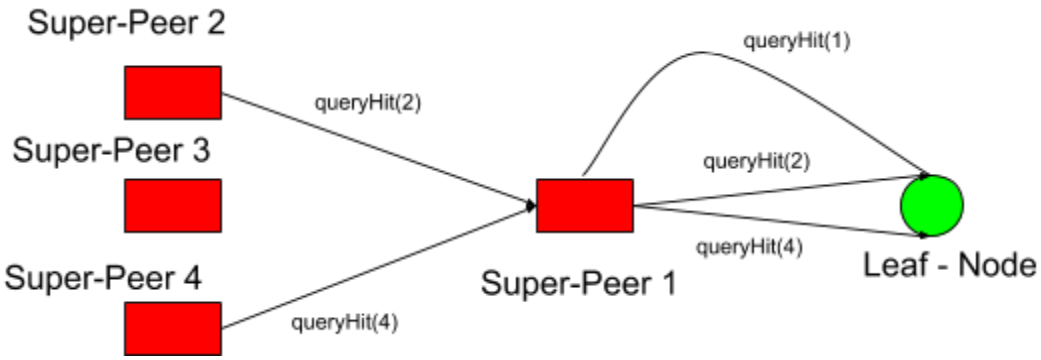
The leaf-node send a query() method call to the super-peer. This call will contain the message ID (msgID), time to live (TTL), file name(fname), requestor's peer ID and requestor's Port Name.



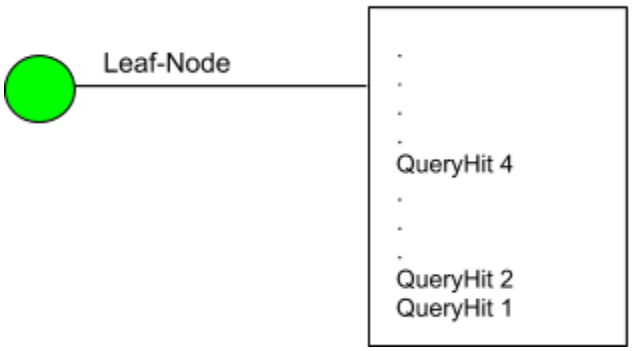
The super-peer then looks up the query in its registry. If no entry is found the query is broadcast to all other super-peers.



The super-peers which get a response from their leaf-nodes for this query respond the super-peer who broadcast the request with a queryHit() call. This call contains the message ID (msgID), time to live (TTL), file name (fname) and an array with the remote leaf-node details (resultArray). The requesting super-peer then forwards this queryHit() responses to the requesting leaf nodes, We can see here that since the leaf nodes of super-peer 3 did not have the requested information, no queryHit() response was sent from there.



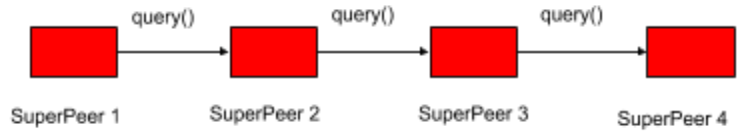
Once the requesting leaf node receives the response, it stores them in a buffer as mentioned previously. It will then ask the user from which source does it want to download the file.



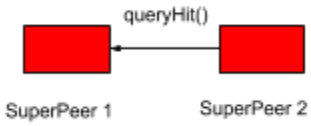
Once the user mentions the source from where the file needs to be downloaded, the leaf node retrieves the file thus ending our operation.

Linear Topology

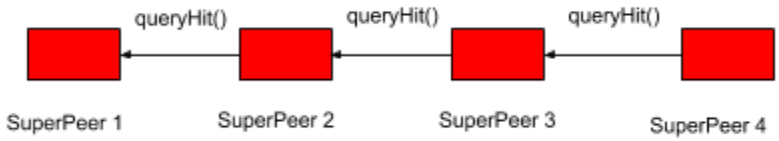
The implementation is the same as All-to-All Topology, with the only difference being that the super-peer can broadcast a request only to its adjacent super-peer. The flow diagrams are given below.
This figure shows the query() call being forwarded to the adjacent super-peers.



Once the file is found the details are forwarded back to the requesting super-peer using the queryHit() call.



The queryHit() calls are linearly sent to the requesting super-peer



This super-peer then sends all the received queryHit() calls to the requesting leaf nodes.

Computing Average Response time for sequential and concurrent requests :
We compute the average response time as well as the response time for multiple concurrent searches in a separate class “AvgRespFileSearch“. The response time is calculated as :

(end time - start time) / (no. of requests)

In the implementation, a leaf node connects to the desired leaf node using a mesh fo super-peers in the network in order to download a file from it, The method to search a file is called sequentially 200 times and the average response time for this process is calculated and displayed. The screenshots for the same have been provided in the performance evaluation section.
We also measure the response time when multiple peers make simultaneous requests to the super-peers. We run the different number of threads concurrently in our every iteration to see the variation on response time with variation in the number of peers.

Trade-offs and Improvements :

The first trade-off that we can see is that as the number of super-peers increase the latency of the network increases. Also, for each request, we will need a higher TTL failing which the requests might expire before reaching all the leaf nodes or before a response can be received. The increased super-peers will also consume more bandwidth. For the linear topology, an added trade-off will be having a single point of failure in the network.

There are a few improvements which can be implemented in our architecture:
Whenever the query method is called, a timer is set (to 5 seconds in our case). The query is terminated once this timer reaches 0. Only the responses received within this period are considered. This is a major drawback since even after the query being killed, leaf nodes will continue to follow up on this request and send a queryHit() response if the file is found.
We can overcome this drawback by having the parent super-peer of the requesting leaf node broadcast a stop search message to other super-peers indicating that the request has been terminated.
Another drawback is that we need a property file containing the information for all super-peers and peers along with the topology being implemented, We can develop a way to do this using the command line interface to reduce the number of user inputs.