P2P File Sharing System

Wenting Yang (wy77), Junfeng Zhi (jz399)

Kaifeng Yu (ky99), Fangting Ma (fm128)

Overview

Our project aims at building a decentralized peer-to-peer file-sharing system. The core theory we applied is the Chord lookup algorithm.

High-Level Design

Fig 1 shows the high-level design of our program. The user can send commands to the program through a terminal interface. The terminal receives and sends user commands to the node. A new node will go through a series of processes to join the Chord Ring and communicate with other nodes.

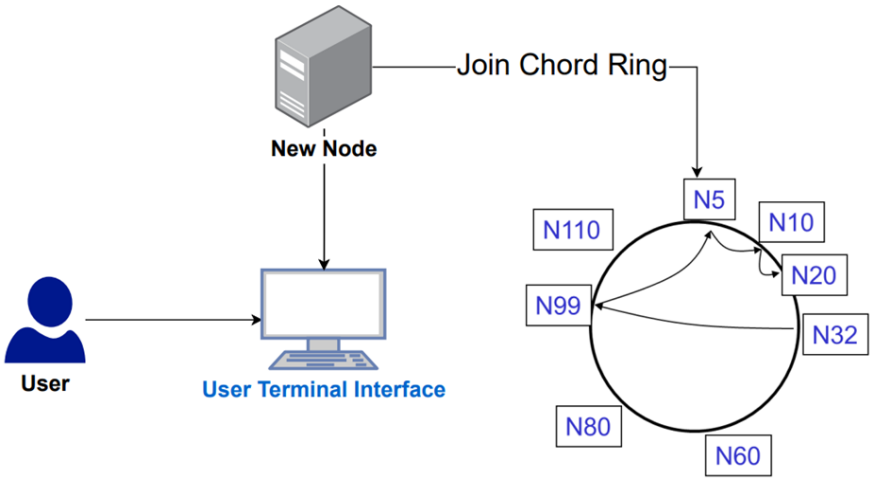

Fig 1. High-level Diagram

Usage

To invoke the program, use the script run.sh under the root directory. First, run:

```
chmod 777 run.sh
```

Then run:

```
./run.sh <Contact Node Hostname> <My Hostname>
```

- The `<Contact Node Hostname>` is a random node that is already in the chord ring. The new node will contact this random node to get access to

the chord ring. If the current node is the first node in the chord ring, give `0.0.0.0` as input here.

- The **`<My Hostname>`** is the hostname of the current node.

Once the script is run, the program will start to establish the node and join the chord ring. We can see some outputs in the terminal. Once it's finished, a terminal user interface will be up, and we can input the corresponding commands.

```
Welcome to the file sharing system! What can we help with you today?
Please choose a number from the following options:
1 Upload a file
2 Delete a file
3 Search if a file exists
4 Download a file
5 Node exit
```

Fig 2. User Terminal Interface

Infrastructure

The infrastructure of the program includes socket, which is the network interface for the file-sharing system. C++11 thread for multi-thread request handling. Google protocol buffer for the file transmitting interface between different nodes.

We design a Node class for the servers that are participating in file sharing. Each Node possesses a part of the distributed hash table, as well as some local information. A Node object stores the following data structures:

```
vector<pair<contactInfo_t, digest_t> > fingerTable; //Route table
unordered_map<digest_t, contactInfo_t> DHT;  //Distributed Hash Table
unordered_map<digest_t, string> localFiles;  // File hash --> File name
```

New Node Join

This is a challenging part of the project. Fig 2 shows a high-level concept of the new node join process. To be precise, the following are the list of operations a node and its peers have to perform to complete the new node join process.

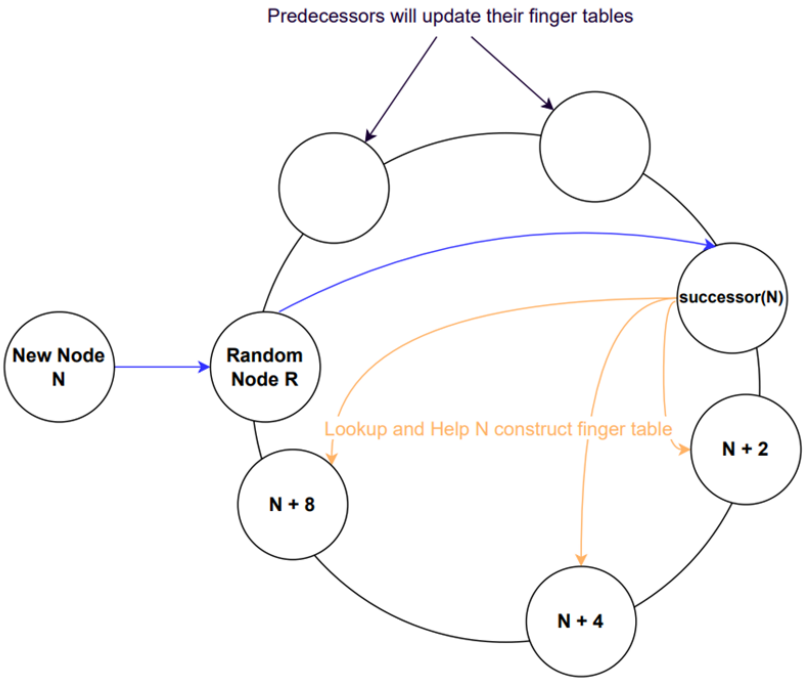

Fig 3. Process of New Node Joining Chord Ring

1. The new node generates its id in the chord ring using SHA-1 hash (its id is the hash of its hostname:port)
2. The new node contacts a random node R that is already in the chord ring. This is called the **entryNode**
3. EntryNode receives the join request from the new node. It will lookup in the chord ring for the node that is immediately after the new node (called the **successor)**
4. The successor is responsible for constructing the routing table for the new node. Successor will send a RouteTableInit packet to the new node. RouteTableInit contains N route table entries, each pointing to successor(n+2^i-1), from i=1 to i=log(2^N)=N, where N is the length of the hash table(hence also the length of the route table, since the log table length is log(num of nodes))
5. Update the route tables of the nodes precedes the new node. The program would search backwards to find successor(new_node's n - 2^(log2N - 1)). Start from this node, traverse its route table to make modifications, then return the first item in the node's route table (i.e. its first successor), then recursively update the route tables of the next node … Until we reach the new_node.
6. The update method is: If New_Node is a more precise successor(n+2^i-1), we update the ith item in the route table. Say the current node is N, we traverse N's route table, say the current item in the route table is M, if the clockwise distance from new_node to N >=

clockwise distance from N to N + hash_delta, and the clockwise distance from New_Node to N <= clockwise distance from M to N, we update the route table.

7. The successor of New_Node sends a part of its distributed hash table (that the New_Node should now be responsible for) to the New_Node. We put this step at the last to ensure all the route tables are properly updated before we hand some files to the New_Node.

File Lookup

In our p2p file-sharing system, the user could choose to look up whether a file exists in the system or not.
To do the lookup, a node first checks whether it is the successor of the file id. If it is, the node will check its distributed hash table to gather information about the file and return it immediately. If it is not, the node will send out a LookupFileRequest and wait for the successor of the file id to send back a LookupFileResponse with the necessary information about the targeted file. For nodes that receive a LookupFileRequest, it will first check if it is the successor of the file id. If it is, the node will check its distributed hash table to gather information about the file and send back a LookupFileResponse to the node which initiates the LookupFileRequest. If it is not, the node will look into its finger table to figure out the next hop, and then forward the LookupFileRequest to the next hop.

Download

To download a file, our file-sharing system will first perform a lookup to check if the file exists in the file-sharing system. If the file does exist and if it locates on another node, the requesting node will send out a DownloadRequest to the node that actually stores the file, which we call it the “**owner node**”. When the owner node receives the DownloadRequest, it will establish a socket connection with the requesting node and send over the file.

Add and Delete File

The file that needs to be added should be placed into the shared_file directory. The node will check if the file exists under the directory and then send the file information to the successor of the file’s hash. It will tell the successor that “I have this file with hash value xxx. You should remember this in your distributed hash table. If anyone is interested in this file, you can tell them I have this file.

Delete file works similarly. The node will first check if it is the owner of this file (only the owner has permission to delete a file). Then it will delete the file in its local storage and send a delete request to the successor of the file, which will remove this information in its distributed hash table.