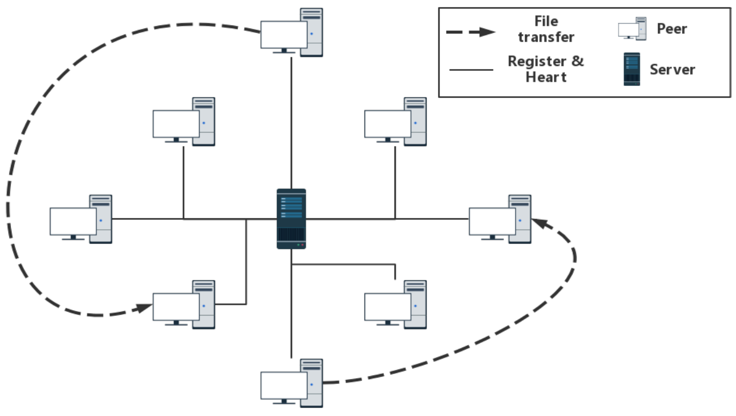
Design Doc for CS2510 project 1

Jiaye Zhu & Ziyi Huang

Design 1. A centralized P2P file sharing network

Topology



Design explanation

As shown in the topology diagram, this is a centralized P2P file sharing network. The central server is only responsible for the peer's registration/unregistration/reregistration(update) and file indexing & searching. File transfer are handled by those peers themselves.

Peer's registration/unregistration/reregistration(update) and file indexing & searching.

When a peer starts, it first connects to the central server to register itself as a valid peer for the whole P2P sharing network. Basically, send the hashes of the files' names it has to the server. Besides, the peer should send heartbeat to the central server. Peers should also report updates(diff) to the server for updating what files it has now so other peers could take advantage of it in the future.

When central server receives the request for registration/updating, it adds the peer's info to the hashtables used for file indexing & searching.

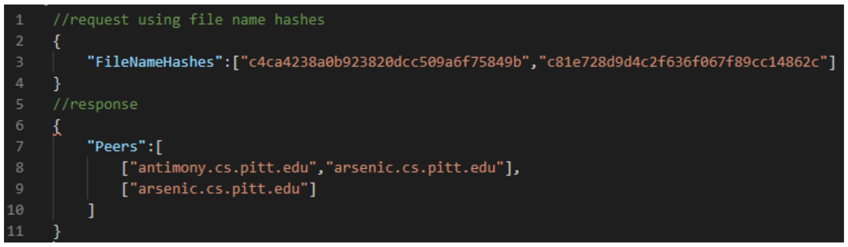
FileNameHash1	[peer1.IP, peer3.IP]
FileNameHash2	[peer2.IP, peer3.IP]

If there is enough RAM, the server should also maintain a hashtable to record which FileNameHashes are stored in which peer. That could save a lot of time for unregistration when the server failed to get heartbeat from a certain peer in some timeout T.

Peer2.IP	[FileNameHash2]
Peer3.IP	[FileNameHash1, FileNameHash2]

When a peer wants to download some files, it may send request to the server using the hashes of the file and get response of which peers have the files.

Example:



```
//request using file name hashes
{
    "FileNameHashes":["c4ca4238a0b923820dcc509a6f75849b","c81e728d9d4c2f636f067f89cc14862c"]
}
//response
{
    "Peers":[
        ["antimony.cs.pitt.edu","arsenic.cs.pitt.edu"],
        ["arsenic.cs.pitt.edu"]
    ]
}
```

File transfer

Once a peer gets response from the server, it will send requests to the peers that have the files it needs. A peer should maintain a hashtable of FileNameHashes and FilePath for the service to send particular files to other peers.

FileNameHash1	/path/to/the/file

For speeding up the downloading process and improve the availability of resources, a peer could ask for all the other peers that have the file it needs and ask them to send part of the file to it. After receiving all the parts, the peer could concatenate those parts to get the file. If some of the peers don't send anything after certain timeout T2, it asks those other peers for the lost parts.

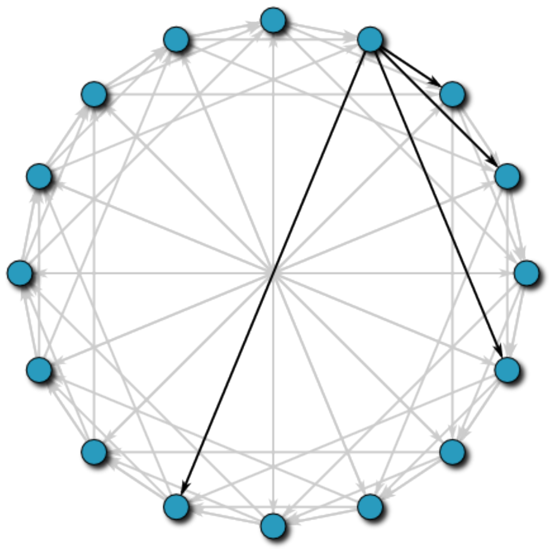
Tradeoffs and possible improvements

Having only one central server can make the program easy to develop and have a simple protocol but it dramatically decreases the reliability of the whole system. If the central server is down, the whole system is gone.

One possible improvement could be setting up server clusters instead of only one central server. It won't change the topology much and won't affect the protocol. However, synchronization inside the index server cluster will cost more time to develop the system.

Design 2. A decentralized P2P file sharing network

Topology



Design explanation

We will use Chord to implement the decentralized file sharing network. There is no separate indexing server in this design, each node in this network holds a finger table and locates the desired file based on the information in the table. After the location is found, these nodes will communicate directly with each other to send and receive files.

Distributed Hash Table

Each file index is represented as a (K, V) pair, K means keyword, which can be a hash of the file name (or other description information about the file), V is the IP address of the node where the file is actually stored (or other description information about the node). All file index entries (all (K, V) pairs) form a large file index hash table, by entering the K value of the target file, all node addresses that stored the file can be found from this table. This large file hash table is then divided into many small pieces, and these small pieces of hash table are distributed to all participating nodes in the system according to specific rules, making each node responsible for maintaining one of them.

Chord Structure

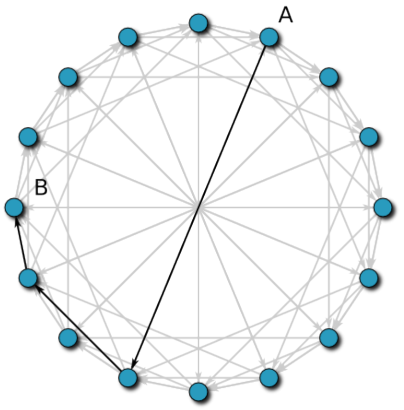
Chord chooses SHA-1 as the hash function, each item will be transferred to an integer, these integers will form an end-to-end ring, called the Chord Ring, and integers are arranged clockwise on the Chord Ring. All Nodes (IP address and Port of the machine) and Keys (resource identification) are then hashed to the Chord ring.

Clockwise, the node in front of the current node is called predecessor, the node at the back of the current node is called successor.

1. Each node will maintain a finger table, the length of the table is m, the ith data stored in the table is the $(n+2^{i-1}) \bmod 2^m$ successor ($1 <= i <= m$) of the current node.

2. Resources(Key) stored in the Node below, which means, along the Chord Ring, the first Node that satisfies hash (Node) > = hash (Key) is the successor of the Key.

3. When we are given a Key, the way we find the location of the resources is to find the successor of that Key.



File Searching

step 1. Check whether the hash of the Key is between node n and the first successor of n, if so, the first successor of n is the node we are seeking, end searching;

step 2. In the finger table of n, find the successor m that satisfies :

1). The distance between hash(Key) and hash(m) is the smallest;

2). hash(m) < hash(Key).

step 3. Forward the request to node m;

step 4. Repeat the steps above, till we find the node that contains the desired key.

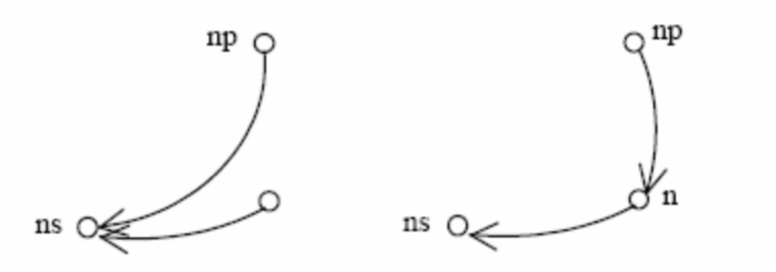
New Node Joining

The addition of new nodes requires the assistance of a known node, any node exists in the Chord network can serve this role.

Chord update nodes' successors and information in its table by periodically implementing stabilization requests to check whether the predecessor of each node's successor is the current node itself.

The joining process includes two stages: Join operation of the new node, which is the function join(n), and the process of being discovered by other nodes, which is the function stabilize().

Suppose np and ns are adjacent nodes in the Chord network, and n is the new node which should be located between np and ns after joining the network.



join(n): the n requests the wizard to find its successor (that is, ns) and initialize its own finger table and successor table. At this time, only n sets its own attribute, and other nodes do not know about the addition of new nodes.

stabilize(): All nodes are required to regularly check the predecessor of it's successor, and send notify messages to their direct successors. In our example, while np is implementing stabilize(), it will notice that the predecessor of its successor ns has been changed to n, so np will change its successor to n and send a notify to n, then n will modify n's predecessor to np, by doing so, the modification of ns, n, np have been finished.

Node failure or exit

All nodes on Chord will periodically check their predecessors and successors. If node n finds that its successor failed, the first available successor node in its successor table will be chosen to replace the failed node, the finger table will then be reconstructed according to the algorithm used when the node was added. After that, the failed node's predecessor will be notified, the predecessor will copy the successor table of n and adds n as the direct successor. Multiple stabilize() is required in order to spread the failure information to the whole Chord network.

File transfer

Once the node requesting the file finds the target node that containing the file, it send the file transfer request with the hash of the desired file directly to that target node, the target node then check the file location in its table and send the file to the node that sends the request.

Tradeoffs and possible improvements

By using Chord, we can locate the desired file quickly using hash. Since each node is acting as a indexing server, failure of some nodes will not affect the whole network and the network can recover easily from the failure

While encountering a node failure, Chord need to implement stabilize() multiple time in order to spread the failure information to the whole network. Although this approach can ultimately guarantee the integrity of the Chord network, its efficiency needs to be further studied when nodes are frequently in and out.