Design Document

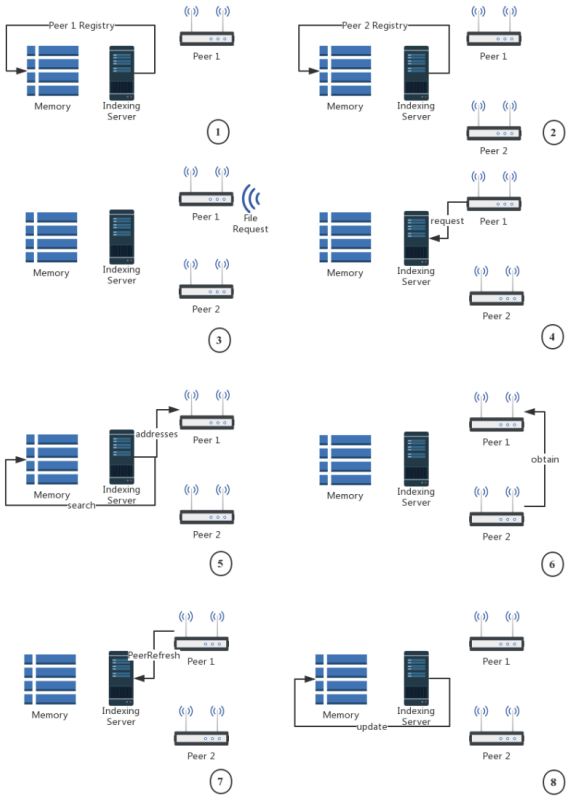
Fangzheng Guo
Zihang Zeng

Aim: Design a simple file sharing system.
Parts: Indexing server, Peers

1 Individual Indexing Server

1.1 Description

'Individual' means there is a separate indexing server in the system. Indexing server is a server that stores information of all peers including addresses and related files. Note that the indexing server only keeps the file names(the identity of files) and the complete files still be stored in each peer. During the requesting process, a peer will request the indexing server with the identity of the target file (names) first. Then, the indexing server will return the addresses of peers which contain the target file. In final step, the peer with inquiry will find the peer with target file based on their addresses, and then obtains the files. The following graph shows the structure of this approach.



Pic 1. An example flow chart for Individual Indexing Server

1.2 Functions

Here we introduce some important functions of this structure.

Indexing Server:

- function 1: Registry
  The inputs are peer id and [file names] corresponding to this unique peer. We use a hash table in Indexing Server to memorize the address of every peer (unique key) and its files names.

- function 2: Search
  The input is a file name, and the output is the addresses of all the peers holding a file with this name. We utilize a brute force search in our hash table.

- function 3: Update
  The input is an address and a file name. For updating purpose, the Server should add the name to the address' value in the hash table.

Peer:

- function 1: Request
  The input is a file name. With this function, the peer sends a request to Indexing Server for the file. The Server should return the addresses of other peers which hold this file.

- function 2: Obtain
  The input is a file name. Calling this function will let the peer send the selected file to its client.

- function 3: PeerRefresh
  Once a peer downloaded a new file, it should send its address and the file's name to the Indexing Server for updating.

1.3 Trade-Offs

1.3.1 Pros

Each peers can obtain any files easily by the stand-alone indexing server. They do not need to know the exact locations of each files because the indexing server stores all information about each peers with corresponding files. Therefore, this structure can add or remove a peer simply without complex operations.

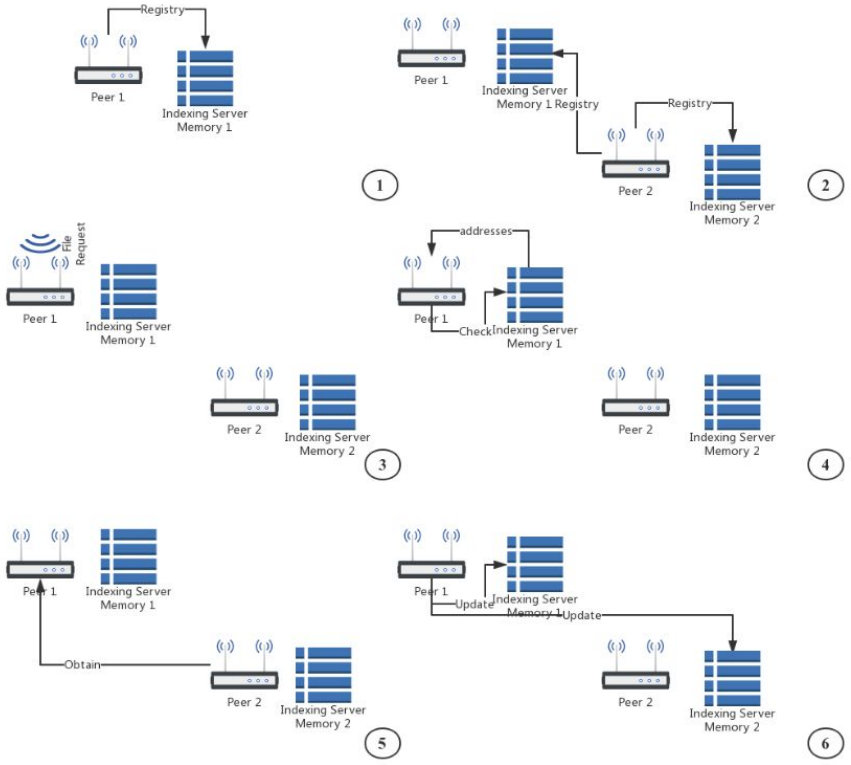
1.3.2 Cons

Obviously, building a stand-alone indexing server will waste more time in the requesting process. Each peers need to request the indexing server first to get the address of target peer so that it can obtain the file. That is, there are more exchanged messages and bytes during the request process, which means more time and more money.

2 Combined an Indexing Server with each Peer

2.1 Description

'Combined' means there is an indexing server as a part of each peer. Each peer stores the information of all the other peers. When a peer is requesting a file, it will visit its own server to get the target addresses. Then, it will find the peer with target files based on their addresses, and then obtains the files. The following graph shows the structure of this approach.



Pic 2. An example flow chart for Combined Server

2.2 Functions

Here are some functions of this structure.

- function 1: Registry
  Each peer should be registered in every Indexing Server, which means after creation, one peer should broadcast its identity to every existing peer, with its peer id and [file names]. We use hash tables to memorize the address of every peer (unique key) and its files names.

- function 2: Obtain
  The input is a file name. With this function, a peer will return the file if it hold this file. This function can be called by other peers which require the file.

- function 3: Request
  The input is a file name. With this function, the peer will search through its Indexing Server memory for the addresses of other peers which hold this file. Also, it can call the Obtain function of other peers according to the addresses.

- function 4: Update
  After receiving a new file, one peer should broadcast its address and the new file name to every peer. The Indexing Server part of every peer should add the names to the address' value in the hash table.

2.3 Trade-offs

2.3.1 Pros

Since each peer has a part as an indexing server, it can search for target addresses from its own memory. This will reduce the number of messages exchange and improve the system's consistency.

2.3.2 Cons

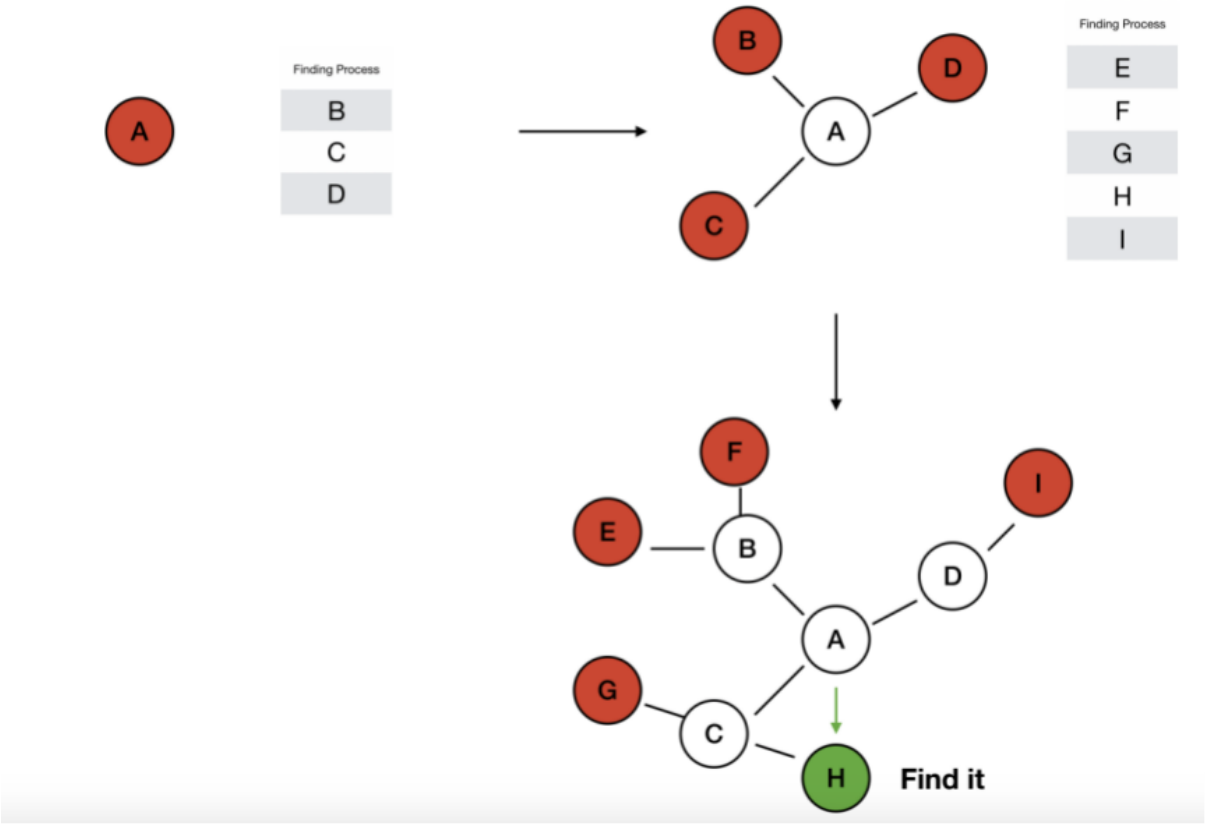
As a reason of fact, we need more space for each peer to store every client's information. Every time a new peer is created or there is a modification on one existing peer's file list, it needs to broadcast its status to every peer, which takes more time and also increases the probability of errors to happen.

3 Possible improvements and extensions

Based on an indexing server can be part of a peer, one possible improvement is that the indexing server part in each peer only stores the content of its adjacent peers. We create a neighbourhood for each peer based on the distance between others and itself. For example, if the distance between a tuple is smaller than a certain number, then we register themselves in each other's indexing server. Then, we will register a series of adjacent neighbours to the indexing servers in each peers. Note that we can set that each peer should have a certain number of neighbours so that its indexing server won't be empty or contain all the information of the graph.

Now assume that peer A is requesting a file F. It firstly checks its own indexing server to see whether F is its neighbour. If not, peer A will broadcast its request to its neighbours and the neighbours will check their own indexing server to see whether file F in their neighbours. If yes, it could return the address to peer A so that A can obtain F through this address. The above process will continue until file F is found.

The request process can be illustrated in the following graph.



The above graph shows that peer A search its indexing server first, which means it search its adjacent peers first. If there is no requested file in its adjacent peers, A will continue searching the indexing servers in its adjacent peers. This process stops when we find a peer with requested file. Then, A will visit this peer, obtaining requested file according to its address.