Design Document

Table of Contents

1 Requirements 2

1.1 Decentralized Indexing Server 2

1.1.1 Bootstrap 2

1.1.2 Server thread 2

1.1.3 Distributed hash table 3

1.1.4 Client thread 3

1.2 Hashing Functions 3

1.3 Replications 4

1.3.1 Registration 4

1.3.2 Search/Lookup 4

1.3.3 Obtain/Download 4

1.3.4 Data Structures 4

1.3.5 API’s 4

1.4 Code flow 5

1.4.1 Insert (Register) 5

1.4.2 Search (Lookup) 5

1.4.3 Obtain (Download) 6

# Requirements

## Decentralized Indexing Server

Aim of this project is to build a decentralized system to index and replicate files. We would be designing this system over 8 peers who would act as a server as a client.

### Bootstrap

When the peer starts, it will read a global configuration file to read the IP: PORT combination where it should listen as a server.

We have defined the following config file to bootstrap the network

**server\_config:**

Configuration file containing IP: PORT for the servers.

Example in the following file, IP: PORT of first server would be 127.0.0.1 1231:

nikatari$ cat server\_config

127.0.0.1 1231

127.0.0.1 1232

127.0.0.1 1233

127.0.0.1 1234

127.0.0.1 1235

127.0.0.1 1236

127.0.0.1 1237

127.0.0.1 1238

127.0.0.1 1231

This config file is used whenever we start the peer:

./peer <peer\_id> : For peer id “i”, “(i+1)”th entry will be read from the file.

Say, If peer id is 0, first entry from server\_config will be read from the file and the respective ip and port will be used for initializing the server.

### Server thread

This thread will maintain a hash table of 1M entries. The hash table for this assignment is a simple array of 1M.

* **Server will run a select() linux call to listen to the current open client fd’s.**
* **Clients will establish socket connection to a peer server will be established only once i.e.**

**during the first request. After that, the socket will be kept alive till the time peer/server shuts**

**down.**

### Distributed hash table

Every peer will maintain a hash table of 12500 entries in this code.

Note this can be adjusted by modifying the following macro:

#define MAX\_HASH\_ENTRIES 12500

**Key to the distributed hashtables will be the file name and value will be the server’s that have the file (This will include the registering server’s id as well as replica’s id):**

### Client thread

This thread will present the users with following options:

1. \*\*\* Enter 1 for inserting an entry \*\*\*
2. \*\*\* Enter 2 for searching peers \*\*\*
3. \*\*\* Enter 3 for obtaining an entry \*\*\*
4. \*\*\* Enter 4 to exit \*\*\*

**Along with sending the request to the primary server, these API’s would send a request to the replica as well to insert, search and obtaining the entry.**

**While retrieving, primary server is down, secondary server would be contacted. An error would be returned if the secondary server could not be reached as well.**

## Hashing Functions

Client will call insert/search/obtain API’s. These API’s will internally calculate the server to whom which the request should be sent. This is done via the following API :

**int server\_compute\_hash (char \*key)** {

int hash = 0;

        char \*temp = strdup(key);

        while (\*temp != '\0') {

                hash = hash + \*temp;

                temp++;

        }

        temp = NULL;

        return hash % NUM\_SERVERS;

}

This function will calculate the sum of the ASCII equivalent of all the characters and then mod it with the number of servers. The final result shall then be the server to which the request should be sent.

## Replications

### Registration

Whenever a server is registering a file, a replica will be calculated using following formula and they key value pair as well as the contents of the file will be sent to the replica:

(atoi(index\_of\_the\_peer) + 1) % NUM\_SERVERS);

### Search/Lookup

If the primary server calculated while registration is not available to respond to the request, request will be routed to the secondary server or the replica. If replica isn’t available as well, API will error out.

### Obtain/Download

If the primary server calculated while registration is not available to respond to the request, request will be routed to the secondary server or the replica. If replica isn’t available as well, API will error out.

### Data Structures

|  |  |
| --- | --- |
| /\* Data structure to send data to server from client \*/  struct client\_data {  char \*ip;  char \*port;  char \*(\*hash\_table)[2];  };  /\* FD's for primary and secondary replica's  \* First column will have primary's fd and  \* second column will have secondary's fd  \*/  int server\_fds[NUM\_SERVERS][1][2]; | /\* Data structure used internally by server to spawn a thread to Put/Get/Delete \*/  struct server\_data {  int client\_fd;  char \*msg;  char \*(\*hash\_table)[2];  };  /\* Lock for all hash operations \*/  pthread\_mutex\_t lock = PTHREAD\_MUTEX\_INITIALIZER; |

### API’s

/\* Populate the servers after reading from server\_config \*/

int populate\_servers (char \*servers[NUM\_SERVERS][2]);

/\* Function used to determine the server to which request should be sent \*/

int server\_compute\_hash (char \*key);

/\* Function to start a peer as a server \*/

void \*server (char \*ip, char \*port, char \*hash\_table[12500][2]);

/\* Function to insert an entry to the hash table \*/

char \*put (char \*key ,char \*value, char \*dir, char \*servers[NUM\_SERVERS][2]);

/\* Function to get an entry from the hash table \*/

char \*get (char \*key, char \*value, char \*servers[NUM\_SERVERS][2]);

/\* Function to delete an entry from the hash table \*/

char \*obtain(int server\_num, char \*file\_name, char \*servers[NUM\_SERVERS][2]);

## Code flow

This section will cover how the API’s to register, search or obtain a file work.

Whenever a peer starts, an interactive menu will be displayed to choose from the following options:

1. Insert 2. Search 3. Obtain

### Insert (Register)

In option 1 in the interactive menu,

1. User will be asked to enter the **file names, their index and the directory in which the file exits to register the file.**
2. This data will be collated the sent to the respective server, calculated using the function server\_compute\_hash.
3. **If the primary server is not available, request will be sent to the secondary server or the replica.**

char \*put (char \*key ,char \*value, char \*dir, char \*servers[NUM\_SERVERS][2]);

**Inputs:** Key (File names), Value (Server’s index), Dir (Directory for the files), Servers (list of servers calculated during bootstrap).

**Outputs:** Success/Error message

### Search (Lookup)

In option 2 in the interactive menu,

1. User will be asked to **enter their index (this index will be used to determine the replica in case primary doesn’t respond to the search request) and the file name they are looking for.**
2. This function will return the id’s of server’s which have the file.
3. **If the primary server is not available, request will be sent to the secondary server or the replica.**

char \*get (char \*key, char \*value, char \*servers[NUM\_SERVERS][2]);

**Inputs:** Key (File name), Value (Server’s index), Servers (list of servers calculated during bootstrap).

**Outputs:** List of peers that have the file.

### Obtain (Download)

In option 3 in the interactive menu,

1. User will be asked to **enter the file name and the server id** they want to use to download the file from.
2. This function will return the contents of the file.
3. **If the primary server is not available, request will be sent to the secondary server or the replica.**

char \*obtain(int server\_num, char \*file\_name, char \*servers[NUM\_SERVERS][2]);

**Inputs**: Server\_num (Server from where file has to be downloaded), File\_name (The file to be downloaded), Servers (list of servers calculated during bootstrap).

**Outputs:** List of peers that have the file.

End of Document