CS425

COMPUTER NETWORKS

PROJECT REPORT

PROJECT NO.11

WEB-BASED FILE HOSTING AND SYNCING SERVICE

(ShareU)

Team – 'J'

Group 20

Arbaz Khan (Y9128)

Mayank Dang (Y9332)

Group 14

Aman Kumar Singh (Y9066)

Pavan Sharma (Y9408)

Problem Statement

Make a web-based file hosting software (like Dropbox) that would enable users to store and share files and folders with others across the Internet using file synchronization. Also, if a user has uploaded a file to your web server, then any changes he makes to his file locally should automatically be synced with the file on the server whenever there is internet connectivity.

Implementation

The project is coded in Python Programming Language (v2.7.1). Sockets are used to transfer messages from one system to other. The crux of the project required the sending of files across a network using TCP stream sockets and syncing them on various remote locations.

Website

On top of the conventional method of syncing files by running the python program, we also offer website support. To create account in ShareU, the client has to first register on the website for an account. A conformation mail is sent to his email id, after which the client can sync his files under a folder placed in his local disk with the server and also download or upload any desired file through the website. Logging in to the website enables a client to upload, download or share any files from any external machine, without running his python program.

Directory Structure

Each client has a folder named “client”. *client.py* is the client-side python file that performs file synchronization and sharing (while looping continuously after connecting with the server). The client-side folder which contains all the files to share and sync with the server is “Sync”. It is also placed in the same folder. *client.py* contains the username of a particular user and that cannot be changed (inherent assumption).

The server has a folder “server” that contains the working file “server2.py” that is used to listen for incoming connections from various clients and fork a process to deal with each process. The folder “/users” has many folders, each being the username of client having all the files of that particular client. The folder “/logs” contains the logs pertaining to each client, named “<username>.txt”. The logs are used to store the paths of files that have been shared with the particular user.

client-server interaction:

The server python script is listening for incoming connections on a predefined port. The server forks another process (for dealing with a client) as soon as any client connection is received and keeps listening for any other connections.

Basic client-server syncing:

- The client-server interaction begins by the client sending its username, which is used by server to ensure that the client is registered with *ShareU*. The execution of client and server then proceed to the 'sync_ker_client' and 'sync_ker_server' functions.
- In these functions, the main syncing proceeds. A listing of all the directories and files in the "Sync" folder of client and the "users/[USERNAME]" folder in the server is done. Those file names are then sent one-by-one to the server. If the particular file is not found in the server, the server then receives the file from the client. Sending of file from the client side is done via the send_file function and receiving the file from the server side is done via the recv_file function.
- If the server already has the file, we need to send the file only if both the copies are different. For that, md5sum of the file is calculated on both sides and compared. The md5sum of a file is unique and changes with any modification in the file. The md5sum is calculated using UNIX command. **Hence, the syncing of the files is done only if the file copy on both sides is different. This prevents the need for any unnecessary transfer of files.**
- A directory name sent from the client to the server is created in the respective directory in server.
- We later delete all the extra files in the server which are not present in the client "Sync" folder.
- Files can also be uploaded via browser. Each time a file is uploaded in server straight from the web browser, we create an entry in the log file of the particular user (in "logs" folder). In each iteration, the log file is checked and if any path in the log refers to an existing file in the server directory of the same user, the file is sent to the client. The log entry is subsequently deleted.

Functions:

send_file (client_socket, data) and recv_file (client_socket, data)

This function is used to send the file whose path is given in 'data'. The file is opened using standard open() system call. Data is read (in binary format) and transmitted, where it is received by recv_file function on the other side in packets of 2KB, where it is written to the file. The end of file is relayed as a special escape sequence, closing the file as a result and ending the file transmission.

list_files (path, file_array, dir_array)

This function takes a path and lists all the files and directories in the path in file_array and dir_array lists, respectively.

***Sharing*:**

ShareU provides us the option of "sharing" of files and/or directories across users. This facility is particularly useful for sharing of files across users, friends or a group of people involved in a collaborative project. Any changes made to the shared folder's contents will appear instantaneously to everyone who is a member of the folder.

The traditional method of sharing involves the use of Databases to keep a track of the following things:

- Who is sharing files (the master user or the 'owner')?
- What are the users with which the files are being shared?
- Synchronization of the shared files on the client side of all the users (with whom the file is shared) with the 'owner' side and vice versa.
- A mechanism to allow only the 'owner' to delete the file and prevent the other users from deleting the file.

The above method has some drawbacks. The procedure is tedious and the overhead of maintaining a database on the server side is huge (a table entry for every shared file having many fields). To avoid this approach, we have implemented ***log files*** and ***symbolic links*** to share anything with a particular user. When user 'A' decides to share a folder or a directory, whose location is given by, say, 'path' with another user 'B', that 'path' is written in the log file of user 'B' and a link in the respective directory of 'B' pointing to the shared directory in the 'owner' is created. Entry in the log file of the user and creation of symbolic links is done by the *PHP* code of the website, while the managing work and file transfer and synchronization is done by Python code. Each location in the log file of a user can be:

- File uploaded via web browser (already in the server user directory).
- Directory created via web browser (already in the server user directory).
- A link to some directory of some other user (shared).
- A link to some file shared by some user with this user.

For the first two cases, we pass the information to the user, which creates the required directory in the corresponding location or copies the file, as explained in "Basic client-server syncing" above. The entry in the log file is deleted then.

Since the paths of the links are stored in the log files, the files which are pointed to by the links are sent to the user. Here, however, the entries of the links are also stored in a global dictionary, hashes. At the start of server2.py, a "share_log.pkl" file is created and a null dictionary is stored in it. Each time we need to access hashes to read (or write to) it, we first load the share_log.pkl file (in which we subsequently save the dictionary using the pickle library) and then perform the required operation and write to the share_log.pkl file again. Hence, share_log.pkl file contains the list of all the folders and

shared files. The keys of the dictionary are the filename of the links and the value pointed by the key is a two-membered array initialized by [hashsum, hashsum] for files and ['1','1'] for directories. Everytime the filename is received from the client, the corresponding hash entry's value[1] is set to hashsum (md5sum of the main file stored in server). Any change made to the shared file is detected as:

- If hashvalue[0] == hashvalue[1] == client_hashsum, then the file has not been changed anywhere. Nothing to be done in this case. Both copies are the same in this case.
- If hashvalue[0] == client_hashsum, then the server's copy is the recent one. In this case, the server sends the file back to the client. (since hashvalue[1] doesn't match, that is, the md5sum of the server-side file is changed)
- If hashvalue[1] == (-1), then the server's (owner's) copy has been deleted. Here, the server signals the client to delete the file in its own directory too!
- Else, the client's file is retrieved.

Each time a shared file is synchronized (server sending file to client or client sending file to server), its hashvalue[0] and hashvalue[1] are set as client_hashsum so as to deny the client any future need of resynchronization (Till the server copy gets changed, resulting a change in hashvalue[1] field of the dictionary entry).

The above setting ensures that any change by any user with whom the file is shared is immediately relayed to the owner copy of the file on the server, which is then updated to the client side of the 'owner' user (by adding an entry of the owner in hashes dictionary and checking the above algorithm again). Any change in the master file is also relayed to all the clients, with whom the file has been shared, ensuring **perfect share-sync algorithm**. All this has been possible due to symbolic links.

Deletion of files is also handled perfectly. If the client deletes its file (which was shared to some other users), the main 'owner' file gets deleted. This means that the symbolic links to this file become dangling. The hashvalue[1] of all these dangling pointers is set to -1, thereby signaling the client to delete its copy of shared file.

On the other hand, if a client deletes its file, the pointer is simply deleted, since the client doesn't have permission to delete the file in the owner's directory. This is ensured by the inherent nature of symbolic links, that is, if we delete the symbolic links, the main target file is not touched at all. Hence, our decision of using symbolic links was a good one.

Data reliability:

To provide the users with data reliability, we have used the concept of multiple servers. This option provides the client an option to fall back to an alternate server (without any connection loss) in case of main server failure. When the main server (that is, server2.py) starts, it forks a process as_client.py which acts as a client and tries to connect to a **"backup server".**

Using the “Backup server, we transfers all the files (which include the data and log files of every client registered) in the current folder to another server, which sits on another machine, and is listening for connections. Hence, whatever data is residing on the main server, is copied regularly to the alternate server via as_client.py interaction with server2.py of the backup machine server. The port number and IP of the backup server are known only to as_client.py, which ensures that no other client can connect to the backup server.

Every client is, hence, provided with two IP’s and port numbers – one of the main server and other of the backup server. In case the first server goes down, we do NOT terminate client program. Instead, the client then connects to the alternate server and uses that server for file synchronization. This is done by the process “server.py” situated in the outer directory in the backup machine, spawned by the backup server itself for listening to any diverted clients. Hence, the backup machine server2.py runs two processes: server2.py (saving data of the main server) and server.py (listening for clients in case of failure of the main server). The syncing can be continued further with this server.py, without the client even knowing about any fault whatsoever.

This provides very high reliability, because the data is not lost and the probability of both servers going faulty before the first one gets back online is very minute.

php implementation:

- Mail function

We wrote a new mail function ‘Email’ in php which does not use ‘mail’ function of php to make our website portable to any domain. It uses smtp.cc.iitk.ac.in as its smtp mail server to send mails by the account ‘akrsingh@iitk.ac.in’ to newly created accounts’ email addresses and is used to verify them. SMTP server commands: [HELO or EHLO, AUTH LOGIN, MAIL FROM , RCPT TO ,DATA] etc.

- Authentication

We use username-password pair to authenticate users. We use mysqlserver.cse.iitk.ac.in to store our users’ information such as name, username, password and email. Email is verified by sending a code (last five characters generated by md5 of the email address) to the provided mail address and then testing it during registration of new account. SESSION variables at server side are used and cookies are not stored on user’s computer, to recognize current logged in user identity.

- Sharing

Sharing between users is allowed. Folders and Files can be shared across accounts. We have implemented sharing using Symbolic links. If a file owned by A is shared with B, then a symbolic link in

the root directory of B is created with target of A's file and B can only download the file. If a Folder is shared, then B can upload files in the link as well too.

- Actions allowed at website

Users can create folders, upload files, download files and share files and folders from website side. Users, however, are not shown other users' root directories (a user cannot share her root directory with anyone) for security reasons.

- Look

We have tried to give the website a professional look. Designing is done by using HTML, CSS and CSS-3. We have also given some interactive features like auto-complete using J-query.

Important features:

- A file hosting service that enables users to store and share files and folders with others across the Internet using file synchronization.
- The ability to sync files by using python terminal script.
- The syncing of the files is done only if the files on both sides are different (by checking the **md5sum** of the files). This prevents the need for any unnecessary transfer of files between computers.
- The use of **log files** and **symbolic links** to implement sharing of files and folders across users.
- A **backup server** that stores data of all the clients and their logs that the clients can fall back to, in case of any physical breakdown of the client.
- A perfect **share-sync algorithm** that ensures that the version of shared file saved on the server and the Sync directories of clients is the one that was last updated by the owner or any user (shared ones).
- The added implementation of a **website** from which the users can download, upload and share files with each other. To use the facility of ShareU, the client has to first register on the site, which involves sending confirmation code and downloading the software archive from the site.