Thomas almestad (thomaalm)
Home exam 1 report in5570

Files

- Typeobjects.m
  - Describing the typeobjects used in the program, including
    - PeerType: Typeobject for the Peer class
    - ServerType: Typeobject for the Server
    - HashType: Typeobject for the hash algorithm, enabling “loose coupling”
- Hash.m
  - Hash implementation using FNV-1a from https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Vo_hash_function#FNV-1a_hash
- Server.m
  - Nopester server implementation, keeps file indexes and acts as a mediator between peers/clients
- Peer.m
  - Nopester client/peer implementation, contains a list of files and has an interface for interacting with the server
- Nopester.m
  - Program to showcase the peer to server use-cases.
- Makefile
  - Used to easily compile and run the program locally and on planetlab.
  - Unfortunately planetlab does not have *make* installed, so it only works locally.
  - Has 5 functions:
    - Compile
    - Run
    - Runp: runs on planetlab
    - Setup [host]: Creates a node, if *host* is specificed, it connects to said host
    - Setupp [host]: Same as Setup, but for planetlab using *emx32*
- outputs/
  - Contains text output for the 3 test cases and the output from the initial dump

Program explanation

Server

The server is the mediator between the peers in the program. It has two indexes, one linking file hashes to a list of file names, and one to link file hases to their respective locations (peers). The indexes were implemented using *Directory* in emerald, which is the equivalent to hashmaps in *Java*, or records in other languages. The indexes are updated when peers notify the server of adding, removing or updating files.

Server functions

- addFile (fileName,fileHash,peer)
    - Called when a peer adds a file
    - Updates hash->name and hash->peer index with new file
- removeFile (fileName,fileHash,peer)
    - Called when a peer wishes to remove a file
    - Removes or updates indexes of said file
- updateFile (oldName,newName,oldFileHash,newFileHash,peer)
    - Updates old file to the new one
    - Same as removing a file, and then adding a new one
- getFileMatches (query) -> ([peers])
    - Searches the hash->name index for a name that matches the query
    - If a match is found, it returns the list of peers from the hash->peer index
- (the following functions are only used in the nopester test program)
- getIndex_hash_name
    - Returns the hash->name index
- getIndex_hash_peers
    - Returns the hash->peers index

Peer

A peer is the client of the program structure. It owns a list of files that is synced with the server, so every other client can access its files. The file index is implemented with Directory, and is indexed as such: fileName -> fileContent

Peer functions

- addFile (fileName,fileContent)
    - Adds the file to the local peer index, and adds to hash to the server
- removeFile (fileName)
    - Removes the file with said name, and notifies the server that the file no longer exists.
- updateFile (oldName,newName,newFileContent)
    - Removes the old file
    - Adds the new file

    - Calls the server.updateFile function
- getFile (fileName)->(fileContent)
    - Called by other peers, usually after it has asked the server for getFileMatches which returns a list of peers
    - Returns the fileContent from the local file index
- findFiles (query)->([peers])
    - Same as the server.getFileMatches but for peers to use individually
    - Basically a client wrapper for the server function
- (the following function is only used in the nopester test program)
- getName
    - Returns the given peer name
    - Identifier used by the test program

Testing

The test program *nopester.m* first generates 5 nodes, and assigns each node 3 files, where the first file is overlapping with the previous node's last file. As described by this statement:

```
for i:Integer<-0 while i<number_of_peers by i<-i+1
     const file<- files.getSlice[i*2,3]
     ...
```

The peer and file names have the pattern *peer + [p_num] + file + [f_num]*
E.g. the first peer will have the files *peer0file0, peer0file1, peer0file2*.

The peers are then dsitributed among the remaining available nodes, after the server has been relocated.

In this program I am performing the following tests:

- Removing a file
- Updating a file
- Downloading a file

Which should test every part of the program.

More specifically, the setup part of the program creates the peers and gives it 3 files each. It thus tests the *addFile* functionality on both the client and the server. This is shown in the first dump *init.txt*.

When downloading a file, I am testing *findFiles, getFile* and *addFile* on the client, and *getFileMatches* and *addFile*, on the server.
In this test case, peer0 tries to search for the file *peer2file1* which peer3 has locally. The server then searches its indexes and gives a list of peers which matches the query. The test program then goes through the list of peers, and then adds the file *peer2file1-downloaded* to peer0. *Test_download.txt* should reflect this.

When removing a file, I am testing the *removeFile* functionality on both the client and the server.
In this test, peer1 tries to remove the file *peer1file1*. It deletes it from its local index, and notifies the server. The server then removes the file in its indexes. This is reflected in the *test_remove.txt* file.

When updating a file, I am testing the *updateFile, addFile* and *removeFile* functionality on both the client and the server.
In this test, peer2 updates its file *peer2file0* to *peer2file0-updated* with the new filecontent *asdasd test test update hehe.* This is reflected in the *test_update.txt* file.