PA4 - DFS

New Attempt

Due Wednesday by 11:59pm **Points** 100 **Submitting** a file upload

Programming Assignment 4 - Distributed File System

Introduction

In this assignment, you will build, in C, a distributed file system for reliable and secure file storage.

A Distributed File System (DFS) is a client/server-based application that allows a client to store and retrieve files on multiple servers. One of the features of a DFS is that each file is divided into chunks and stored on different servers and can be reconstructed even if one of the servers is not responding.

In this assignment, a DFS client (DFC) uploads and downloads to and from some number of (4 for the following discussion) distributed file servers (DFS1, DFS2, DFS3 and DFS4). For our purposes, the DFS servers can all be running locally on a single machine with different port numbers, for example from 10001 to 10004.

When the DFC wants to upload a file to the DFS servers, it first splits the file into 4 equal length chunks P1, P2, P3, P4 (a small length difference is acceptable if the total length is not evenly divisable by 4). The DFC then groups the 4 chunks into 4 pairs such as (P1, P2), (P2, P3), (P3, P4), (P4, P1). Finally, the DFC uploads the pairs to the 4 DFS servers. The stored file now has (limited) redundancy - one failed server will not affect the integrity of the file.

How to choose which file chunks go where

In order to balance where chunks end up, you will need to hash the filename and then apply a modulus:

Let x = HASH(filename) % y, where Y is the number of DFS servers available.

The table below shows how to determine where chunks reside based on the value x, assuming y = 4:

x	DFS1	DFS2	DFS3	DFS4
0	(1,2)	(2,3)	(3,4)	(4,1)
1	(4,1)	(1,2)	(2,3)	(3,4)
2	(3,4)	(4,1)	(1,2)	(2,3)
3	(2,3)	(3,4)	(4,1)	(1,2)

You can use the md5sum command or your choice of an md5 hash library to compute MD5HASH().

Requirements for the DFC

The client needs to be invoked with the following syntax:

```
# ./dfc <command> [filename] ... [filename]
```

The configuration file ~/dfc.conf should contain the list of DFS server addresses and port numbers:

```
server dfs1 127.0.0.1:10001
server dfs2 127.0.0.1:10002
server dfs3 127.0.0.1:10003
server dfs4 127.0.0.1:10004
...
server dfsn 127.0.0.1:1000n # n number of servers
```

The DFC should provide for 3 commands ***list***, ***get*** and ***put***:

The ***list*** command inquires what files are stored on DFS servers, and should print the filenames available. The list command should also be able to identify if the file chunks on the available DFS servers are enough to reconstruct the original file. If pieces are not enough (means some servers are not available) then "[incomplete]" will be added to the end of the file.

The ***get*** command downloads all available chunks of a file from all available DFS servers. If the file can be reconstructed then write it to the current working directory. If not, then print the error ***<filename> is incomplete***, where <filename> is replaced by the actual filename.

The ***put*** command uploads the indicated file(s) into the DFS. If there are not enough servers to store the file reliably, your program should respond with ***<filename> put failed***.

Requirements for the DFS:

The DFS servers should be invoked by the follow commands:

```
# ./dfs ./dfs1 10001 &
# ./dfs ./dfs2 10002 &
# ./dfs ./dfs3 10003 &
# ./dfs ./dfs4 10004 &
```

Each DFS server should have its own directory named dfs1, dfs2, dfs3, or dfs4, under the server's current working directory.

A client must try for 1 second to connect to the server. If a DFS server does not respond in 1 second, we consider that server is not available.

Your DFS servers must handle multiple connections and service multiple DFCs concurrently.