CSCI4430 (Spring 2020)

Assignment 2: A Multi-Client Multi-Server File Transfer Application

Due on March 26, 2020, 23:59:59

1 Introduction

In this assignment, you will extend the *MYFTP* application that you developed in Assignment 1 with fault tolerance. The idea is that instead of hosting a single server, we now have multiple servers that store a file. Even if some (but not all) servers are unavailable, each file remains available through the file reconstruction of the remaining servers.

2 Design Details

Design overview. To support fault tolerance, we employ *erasure coding*, in which we configure two parameters n and k, where $n > k$. Erasure coding transforms every k fixed-size blocks into n coded blocks of the same size. The fault tolerance property ensures that any k out of the n coded blocks can recover the original k blocks.

To implement the features of erasure coding, we use the Intel Storage Acceleration Library `ISA-L` (https://github.com/intel/isa-l). The tutorials will give you additional details on how to install and use ISA-L.

Figure 1 shows the architecture. We divide a file into different blocks of the same size; the last block is padded with zeroes to fill the block size. We partition the blocks into groups of k blocks each; the last group may contain fewer than k blocks. We transform each group of k blocks into n coded blocks, where we call the n coded blocks a *coding group*; for the last coding group, if it contains fewer than k blocks, we add extra zero blocks to form k blocks and perform coding. Thus, a file may be composed of multiple coding groups. The block size is a configurable parameter (in bytes).

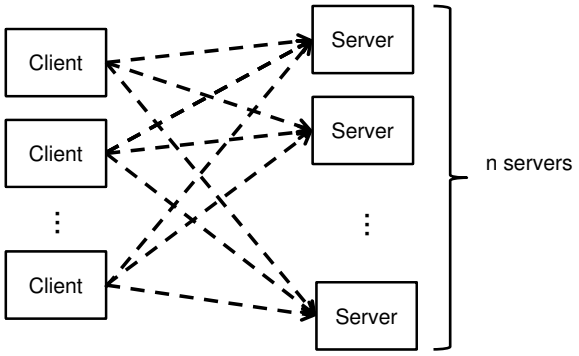

Figure 1: Multi-client multi-server architecture.

Upload/download files. To upload a file to severs, a client distributes the n coded blocks of each coding group to n servers. To download a file, a client contacts k servers to reconstruct the file (note that the file can be initially uploaded by the same or a different client). It is possible that the servers may not be available during reads. The client may contact the *first* k available servers to retrieve the blocks to decode the file.

List files. A client can request *any* available server to list the currently stored files. All servers are supposed to have the same view of the list of files being stored.

Multiple client/server support. Each server still uses multi-threading based on the `pthread` library (like Assignment 1) to support multiple clients. However, each client will use I/O multiplexing with `select()` to support the connections from multiple servers.

2.1 Assumptions

We make the following assumptions to make our life easier.

- All servers are available during an upload; by available, we mean that the server can be successfully connected. If some servers are unavailable when an upload command is issued, the upload operation simply aborts.
- We assume that $n \leq 5$ and $1 < k < n$. All clients and servers are using the same k and n in the same deployment.
- Each server supports at most 10 active clients at the same time.
- The block size is in units of bytes. We assume that it's always a power of 2. The maximum block size is $2^{20} = 1048576$ bytes (i.e., 1 MiB).
- The maximum file size is 2^{30} bytes (i.e., 1 GiB). A file could be in ASCII or binary format.
- You may extend the protocol messages from Assignment 1 for this assignment. However, you don't need to follow the exact format of the protocol messages. You are free to design additional data structures to keep track of the blocks for each file. In this assignment, we only focus on whether the functionalities are supported.
- We will only test your programs in the virtual machines (installed with Linux). You don't need to test your programs on the Unix platform nor the cross-platform issue.

2.2 Interfaces

The server uses the following command-line interface:

```
server> ./myftpserver serverconfig.txt
```

The configuration file `serverconfig.txt` contains the following lines: (i) n, (ii) k, (iii) block size (in bytes), (iv) the ID (from 1 to n) of the server in a coding group, and (v) port number. An example of `serverconfig.txt` is as follows:

```
5
2
3
4096
13457
```

The client uses the following command-line interface:

```
client> ./myftpclient clientconfig.txt <list|get|put> <file>
```

Note that the `file` argument only exists for the get and put commands, and you can define your own output format for any error message. The configuration file `clientconfig.txt` contains the following lines: (i) n, (ii) k, (iii) block size, and (iv) list of server IP addresses and ports. An example of `clientconfig.txt` is as follows:

```
5
2
4096
192.168.0.1:13567
192.168.0.1:13568
192.168.0.2:13567
192.168.0.3:13567
192.168.0.3:13568
```

3 Milestones

The new MYFTP program should support similar functionalities stated in Assignment 1's specification. We clarify all milestones here.

- A client can list files in the repository directory on the server side.
- A client can upload a file (either an ASCII or binary file) to the servers. If the file doesn't exist locally, or not all servers are available, then the client displays an error message.
- A client can download a file (either an ASCII or binary file) from any k out of n available servers. If the file doesn't exist or fewer than k servers are available, the client displays an error message. Note that the downloaded file may be initially uploaded by a different client.
- Each client and server can reside in different machines.
- Each server supports a general number of active client connections (up to 10) using multi-threading. Each client also supports simultaneous server connections, but using I/O multiplexing.

4 Submission Guidelines

You *must* at least submit the following files, though you may submit additional files that are needed:

- *myftp.h*: the header file that includes all definitions shared by the client and the server (e.g., the message header)
- *myftp.c*: the implementation of common functions shared by both the client and the server
- *myftpclient.c*: the client program
- *myftpserver.c*: the server program
- *Makefile*: a makefile to compile both client and server programs

Your programs must be implemented in C/C++. They must be compilable on the virtual machines. Please refer to the course website for the submission instructions. Have fun! :)