CS/CE/TE 6378: Project II

Instructor: Ravi Prakash

Assigned on: March 12, 2018
Due date and time: April 2, 2018, 11:59 pm

This is an individual project and you are required to demonstrate its operation to the instructor and/or the TA to get credit for the project.

1 Requirements

1. Source code must be in the C /C++ /Java programming language.
2. The program must run on UTD lab machines (`dc01, dc02, ..., dc45`).

2 Client-Server Model

In this project you are to emulate a file system and a set of clients accessing the files. You may require the knowledge of thread and/or socket programming and its APIs of the language you choose. It can be assumed that processes (server/client) are running on different machine (`dcXX`).

3 Description

Design a distributed system with three file servers, two clients and a metadata server (M-server) to emulate a distributed file system. One example of such network topology is shown in Figure. 1. Your program should be easily extensible for any number of servers and clients. The file system you are required to emulate has one directory and a number of

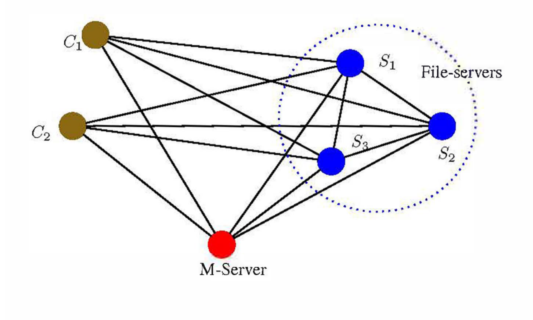

Figure 1: Network Topology

text files in that directory. A file in this file system can be of any size. However, a file is logically divided into chunks, with each chunk being at most 8192 bytes (8 kilobytes) in size. Chunks of files in your filesystem are actually stored as Linux files on the three servers. All the chunks of a given file need not be on the same server. In essence, a file in this filesystem is a concatenation of multiple Linux files. If the total size of a file in this file system is $8x + y$ kilobytes, where $y < 8$, then the file is made up of $x + 1$ chunks, where the first x chunks are of size 8 kilobytes each, while the last chunk is of size y kilobytes.

For example, a file named *file_x* may be of length 20 kilobytes, and be made up of three chunks: the first chunk of size 8 kilobytes stored in server S_1 with local file name ABC, the second chunk of size 8 kilobytes stored in server S_3 with local file name DEF, and the third chunk of size 4 kilobytes stored in server S_2 with local file name GHI.

In steady state, the M-server maintains the following metadata about files in your file system: file name, names of Linux files that correspond to the chunks of the file, which server hosts which chunk, when was a chunk to server mapping last updated.

Initially, the M-server does not have the chunk name to server mapping, nor does it have the corresponding time of last mapping update. Every 5 seconds, the servers send heartbeat messages to the M-server with the list of Linux files they store. The M-server uses these heartbeat messages to populate/update the metadata.

If the M-server does not receive a heartbeat message from a server for 15 seconds, the M-server assumes that the server is down and none of its chunks is available.

Clients wishing to create a file, read or append to a file in your file system send their request (with the name of the file) to the M-server. If a new file is to be created, the M-server randomly asks one of the servers to create the first chunk of the file, and adds an entry for that file in its directory. For read and append operations, based on the file name and the offset, the M-server determines the chunk, and the offset within that chunk where the operation has to be performed, and sends the information to the client. Then, the client directly contacts the corresponding server and performs the operations.

In effect, clients and servers communicate with each other to exchange data, while the clients and servers communicate with the M-server to exchange metadata.

You can assume that the maximize amount of data that can be appended at a time is 2048 bytes. If the current size of the last chunk of the file, where the append operation is to be performed, is S such that $8192 - S < appended\ data\ size$ then the rest of that chunk is padded with a null character, a new chunk is created and the append operation is performed there.

4 Operations

1. Your code should be able to support creation of new files, reading of existing files, and appends to the end of existing files.
2. If a server goes down, your M-server code should be able to detect its failure on missing three heartbeat messages and mark the corresponding chunks as unavailable. While those chunks are available, any attempt to read or append to them should return an error message.
3. When a failed server recovers, it should resume sending its heartbeat messages, and the M-server which should repopulate its metadata to indicate the availability fo the recovered server's chunks.

5 Submission Information

The submission should be through eLearning in the form of an archive consisting of:

1. File(s) containing the source code.
2. The README file, which describes how to run your program.

DO NOT submit unnecessary files.