CS 6378: Programming Project III

Instructor: Ravi Prakash

Assigned on: November 12, 2019
Due date: December 3, 2019

This is an group project to be performed in groups of two. You get to pick your partner for the project. Both members of the group are required to jointly 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`).
3. This project is a client-server model. You are to emulate a file system and a set of clients accessing the files.

2 Description

1. There are five file servers, S1 through S5, two clients, C1 and C2, and a metadata server (M-server). Each of the clients is connected to all file servers and the metadata server. All of file servers are connected to each other, and the metadata server is connected to all file servers.
2. The file system you are required to emulate has one directory and a number of 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 4096 bytes (4 kilobytes) in size.
3. Chunks of files in your filesystem are actually stored as Linux files on the five 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 4x+y kilobytes, where y ¡ 4, then the file is made up of x + 1 chunks, where the first x chunks are of size 4 kilobytes each, while the last chunk is of size y kilobytes.
    - For example, a file named X.txt with a length of 10 kilobytes should be divided into three chunks, and these three chunks might be stored in different servers.
4. For each chunk, three replicas are maintained at three different servers. When a new chunk is to be created, the metadata server selects three chunk/file servers at random and asks each one of them to create a copy of the chunk.
5. 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 servers host replicas of each chunk, which server hosts which chunk, when was a chunk to server mapping last updated.
6. At first, 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 file 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.

7. 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 three of the servers to create replicas of 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.

8. Any append to a chunk is performed to all *live* replicas of the chunk. Use **two-phase commit protocol**, with the appending client as the coordinator and the relevant chunk/file servers as cohorts, to perform writes. Assume there is no server failure during an append operation.

9. If multiple appends to the same chunk are submitted concurrently by different clients, the M-server determines the order in which the appends are performed on all the chunks. Once a client completes an append to a chunk it informs the M-server. Only then does the M-server permit another client to perform an append to the same chunk.

10. A read from a chunk can be performed on any one of the current replicas of the chunk.

11. A recovering chunk/file server may have missed appends to its chunks while it was down. In such situations, the recovering chunk/file server, with the help of the M-server, determines which of its chunks are out of date, and synchronizes those chunks with their latest version by copying the missing appends from one of the current replicas.

12. Only after all chunks of a recovering chunk/file server are up-to-date does that chunk/file server resume participating in append and read operations.

13. You can assume that the maximum amount of data that can be appended at a time is 1048 bytes. If the current size of the last chunk of the file, where the append operation is to be performed, is S such that 4096-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.

3 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 for the recovered servers chunks.

4 Submission Information

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

1. File(s) containing the source code. Your source code must have the following, otherwise you will lose points:
   (a) Proper comments indicating what is being done.
   (b) Error checking for all function and system calls.

2. The README file, which describes how to run your program.