CS 6378: Programming Project II

Instructor: Ravi Prakash

Assigned on: March 12, 2022
Due date: March 31, 2022

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.

In this project, you are required to implement the Ricart-Agrawala algorithm for distributed mutual exclusion, with the optimization proposed by Roucairol and Carvalho, in a client-server model.

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. You will need to know thread and/or socket programming and its APIs for the language you choose. It can be assumed that each process (server/client) is running on a single machine (`dcXY`). Please get familiar with basic UNIX commands to run your program on *dcXY*.

2 Description

1. There are three servers in the system, numbered from zero to two.
2. There are five clients in the system, numbered from zero to four.
3. Assume that each file is replicated on all the servers, and all replicas of a file are consistent in the beginning. To simulate separate file systems for each server, create a separate sub-directory for each server within your home directory. Initially, all the sub-directories are identical in terms of the list of files they contain, and the contents of the corresponding files.
4. A client can perform WRITE operations on files. The client should randomly select one of the three servers and send the WRITE request with the appropriate information (as described below) to that server. The server, acting as the proxy for the client, then broadcasts the WRITE request to all other servers. The WRITE operation to a specified file is performed as a critical section execution by the proxy server using the Ricart-Agrawala algorithm.
5. Once the WRITE is performed (i.e., the proxy server exits the critical section), the proxy server sends a message to the client informing it of the completion of the WRITE.
6. A client issues only one WRITE request at a time.
7. Different servers (acting as proxies for different clients) can concurrently perform WRITE on different files.
8. In order to satisfy the requirements mentioned above, you must implement Ricart-Agrawala algorithm for distributed mutual exclusion between the three servers, with the optimization proposed by Roucairol and Carvalho, so that no two servers are allowed to update the same file concurrently. The write operation on files can be seen as *critical section* executions.

9. The operations supported by servers are as follows:
   (a) ENQUIRY: A request from a client for information about the list of hosted files.
   (b) WRITE: A request from a client to append a string to a given file.

   The servers must reply to the clients with appropriate messages after receiving each request.
10. Assume that the set of files does not change during the program's execution. Also, assume that no server failure occurs during the execution of the program.
11. The client must be able to do the following:
    (a) Gather information about the hosted files by querying the servers and keep the metadata for future.
    (b) Send request to a randomly chosen server to append a string $\langle client\_id, timestamp \rangle$ to a file f_i during WRITE operation. *Timestamp* is the value of the clients, local clock when the WRITE request is generated. This append must be performed on all replicas of f_i
12. Display appropriate log messages to the console or to a file.

    Keep in mind to close all open sockets when your program exits/terminates.

3 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.

4 Additional Information

Hopefully, the following paragraphs will help you understand the utility of this exercise.

In a very simple form you are simulating the implementation of a replicated file system. Consider a cloud-based file system like Google Docs. There are a very large number of users (clients) of such a service. To ensure that clients do not experience any interruption in their service, Google would maintain replicas of its file system in multiple servers located in different parts of the Internet. Google may have valid reasons to not disclose to its clients the exact number of such servers and the protocol implemented among them for replica consistency. So, when a client wishes to use Google Docs, the Domain Name Service may provide the client with the address of one of the servers. The client submits its request to that server which acts as a proxy for the client, and ensures that the client's request is satisfied. This is precisely what this project is trying to simulate. Please note that while I am using Google Docs as an example, I am not claiming that this is how Google has implemented its service.

Also, remember that while one client may be modifying a file through its proxy server, another client may be concurrently modifying another file through the same or another proxy server. Consequently, the mutual exclusion algorithm should be implemented at the granularity of files, and not at the level of the entire file system.