CS/CE 6378: Project II

Instructor: Ravi Prakash

Assigned on: October 18, 2011
Due date and time: November 10, 2011, 11:59 pm

1 Requirements

1. This is an individual project and you are expected to demonstrate its operation to the instructor and/or the TA.
2. Source code must be in the C /C++ /Java programming language.
3. The program must run on UTD lab machines (`net01, net02, ....`).

2 Client-Server Model

In this project you are expected to use *client-server* model of computing. 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 (`netXX`).

3 Description

Design a distributed system with three 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. **??**. Your program should be easily extensible to any number of servers and clients. All the servers have a local copy of some file(s). All replicas of a file are

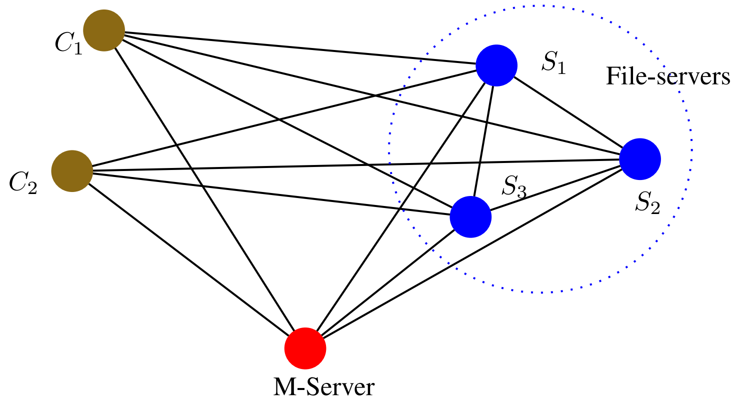

Figure 1: Network Topology

consistent, to begin with. The M-server has metadata of files and information about servers/clients. However, it should not participate in actual file operations (Create, Read or Write). You can assume that the file-servers and the the M-server will not crash. However, the program can expect client crashes.

The M-server must implement following services –

1. It has knowledge of all servers, clients and files hosted by the servers.

2. It keeps track of files along with current (to the best of its knowledge) location and status of files. The possible states of a file could be:

    a. CLOSED : The file is closed and not being used by any client.

    b. R_OPEN: The file is open in read mode by some client(s).

    c. W_OPEN: The file is open in write mode by some client.

3. The M-server must ensure that if a file is opened in the write mode by some client, other clients cannot open the same file in read or write mode. However, multiple clients should be allowed to open the same file in the read mode.

4. The M-server keeps track of multiple users (clients) of a file (in case of multiple read).

5. It receives periodic updates from the file-servers about their local files' status and updates its own metadata.

6. Upon receiving a request (Create|Read|Write) for permission from a client for a particular file, it should reply with either REJECT or GRANT permission to perform the corresponding operation. If it grants the permission then the following is done:

    (a) If a client is allowed to open a file in the read mode, the M-server randomly selects one of the file-servers where the read is to be performed and informs the client of its selection.

    (b) If a client is allowed to open a file in the write mode, the M-server communicates to the client the identities of all the file-servers maintaining replicas of that file. ...

7. On detecting a crashed client(s), if any, it instructs the server(s), on behalf of the crashed client, to close all files opened by the crashed client, and updates its local metadata.

The desired operations by a client are as follows:

1. A client must obtain GRANT permission from M-server before processing each request. In case the permission is granted, the client can use the SERVER location address (provided by M-server) for further communication. If the permission is denied (REJECT), it should abort the request.

2. It may send CREATE a file request, READ from a file request or WRITE to a file request to the SERVER upon having GRANT from M-server for corresponding request.

3. It performs READ/WRITE request in following three steps –

    I. Opens the file in read/write mode and sets the status of the file to R_OPEN/W_OPEN.

    II. Transfers data between client and server(s) and read/write the content from/to the given file.

    III. Closes the file and sets the file's status to CLOSED.

    So, any READ/WRITE operation results in three sets of request-response communications between client and server.

The desired operations by a server are as follows:

1. It locally populates metadata of files that it hosts, which also contains the current status of files (CLOSED|W_OPEN|R_OPEN).
2. It must send periodic updates of its metadata to M-server.
3. Upon request from a client, it must be able to CREATE a new file, WRITE to a local file and READ from a local file.
4. In case of CREATE, *all* copies of the file, across the servers, should be updated consistently.
5. In case of WRITE, since it has three operaions, each operation must be completed on *all* servers before begining of next step.
6. It must reply either SUCCESS or FAILURE to the requests from client along with requested data (if any).

As part of this project you have to determine how a server, on receiving the WRITE request communicates with the other servers to get the same write performed on all the copies. Your program must support the creation of new files, writes to the end of files, reads and writes at specific offsets from file beginning. It must also report an error if an attempt is made to READ that failed at the requested server for some reason. An error must be reported if there is an attempt to WRITE and at least one of the servers is unable to write to it. A client may crash during a request processing, for exmple, just after completion of step I or step II of READ/WRITE operation.

Your program must display informative log messages on console.

4 Submission Information

The submission should be through WebCT 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.