Usage

To run, compile and run all the executables with a command line argument corresponding to the chunk size (in bytes). Run a metadata server, one or more clients, and one or more data servers, all on separate terminals. Run the metadata server first as ./m_server.out <CHUNK SIZE>. Then run 3 or more data servers with the same CHUNK SIZE and 1 or more client with the same CHUNK SIZE. In order to ensure that the queue is not filled with a few large messages, CHUNK SIZE must not exceed 512 bytes.

All terminals must have the same current working directory - the message queue id is generated by hashing this directory path string. We have chosen to use only 1 message queue for communication between all the processes as well as a single message format that is passed between the processes. They message type field of each message in the message queue denotes the target pid of the process that this message was intended for. For messages directed to the metadata server, we use message type 1 instead.

Implementation

A single message format is used for all messages. In addition to the data field, it also includes 3 fields for addresses (used in add chunk), 2 for paths (the second is used for move and copy) and one each for sender, request type, status, error message, and chunk ID. Not all of these are used in every message.

On starting, the metadata server creates the shared message queue and starts poling it for relevant messages. It then handles each request as it comes. To end, a SIGINT (Ctrl+C) should be sent. This deletes the message queue, so that the same key can be reused.

The metadata server does not maintain a true hierarchy (for the file structure) - it simply maintains a hashmap of file names, which may or may not normally be valid directory names. For example, "qwe/@ 4qw" would not normally be a valid name, but it is directly hashed here, thus it does not raise errors.

When a new data server is started, it notifies the metadata server and mentions its pid, which is stored. It also creates a directory, to store all the chunks that are sent to it. The data server can take requests to store a chunk, copy a chunk, move a chunk, run a command on a chunk, or list all the chunks that it has.

Data servers do not store multiple copies of the same chunk, if such requests arise (this may occur if there are less than 3 data servers). The data server responds with a status message after every communication to denote whether the process executed successfully or not. Furthermore the data server also prints whatever operations it is performing on its own terminal.

When we run the client, it does no special communication at the start. It starts communicating with the metadata and data servers after the user has specified an operation to be done. The operations are described in detail in the next section.

Operations

The syntax of the operations are given below:

ADD FILE syntax: 1 <file path>

ADD CHUNK syntax: 2 <file path><machine file path><chunk number>

COPY syntax: 3 <source><destination>

MOVE syntax: 4 <source><destination>

REMOVE syntax: 5 <file path>

COMMAND syntax: 6 <command><d-server-pid><chunk name>

LS DATA syntax: 7 <d-server-pid>

LS FILE syntax: 8 <file path>

ADD FILE: Metadata server adds the file to the hashmap. The metadata server then sends a message back to the client to tell whether the operation was successful or not. Usage Example: 1 /home/user/hello.txt

ADD CHUNK: The client asks the metadata server for a chunk ID number and a list of 3 data servers to send the chunk to. Then, to each server in the resulting list, it sends a store request with the <chunk number>th chunk of the file specified by <machine file path>(which is local to the client). For example, if <chunk number>is 1 and we are trying to send our local file hello.txt, it would send the first CHUNK SIZE bytes of the file, if <chunk number>was 2, it would send the next CHUNK SIZE bytes. Data servers reply with a confirmation. Usage Example: 2 /home/gfs_user/hello.txt /home/client_user/Documents/hello.txt 1

COPY: The metadata server adds a file at the destination and populates it with duplicate chunk IDs for each chunk. Then, for every chunk, it selects 3 data servers, and instructs each current holder of the chunk to make a store request to a new holder, under the new ID. Usage Example: 3 /home/user/hello.txt /home/user/hello2.txt

MOVE: The metadata server updates its hash table without moving any chunks. Usage Example: 4 /home/user/hello2.txt /home/user/hello3.txt

REMOVE: The metadata server removes the entry for <file path>from its hash table and sends a remove request for all the data servers for each of the chunks. Usage Example: 5 /home/user/hello3.txt

COMMAND: The client sends a request to the data server whose pid is <d-server-pid>to run the command <command>on the chunk specified by <chunk name>(note that chunk name is just the chunk id number, not internal file name). The data server forks, runs the command on the specified chunk and then sends the result back to the client. It is useful to first run LS FILE to get the pids of the data servers that the chunks of a file are stored in. Usage Example: 6 wc -l -w 123456 3

LS DATA: The client sends a message to the data server whose pid is <d-server-pid>. The data server forks, runs the ls command and returns the result (internal file list) back to the client. Usage Example: 7 123456

LS FILE: The metadata server provides the list of chunk ids associated with the file and the data servers that each of these chunks are stored at. Usage Example: 8 /home/user/hello.txt