Overview

I have implemented a Distributed File Server that consists of

- Distributed Transparent File Access
- Replication
- Lock Service

The file server operates in an upload/download manner.

The server will respond as expected to the standard test protocols described in lab 3. On the client side there only exists the desired ‘client proxy’ functionality. The client is capable of interacting with the server in a way that fulfils the lab. I believed that adding an extra layer was not needed and it would only use up another socket which would add to the cost of commands. However, for a more complex design that would have a presentation layer(GUI) I would see the requirement of the extra layer. I failed to add a fourth feature. During the initial stages I thought developing a security feature would be of interest to me but with hindsight a Directory Service should have been the second most important feature to implement. If I had implemented a Directory Service it would have made the other features much easier to implement as you will see.

Replication

Mix between Asynchronous and Synchronous Model. I have implemented a 'Primary Copy' architecture for replication. I do not cover the case in which a 'primary' server may be down or has left the network.

Client update request returns after the server it is writing to has been updated. However, each update is processed at all replicas in an ordered fashion directly after the initial update is made. This avoids the high cost of replication apparent to the client in a purely synchronous model (optimises response time). The client does not have to wait until it is replicated across all servers.

The way I have implemented replication means that a replica server can lay in the background if it remains unknown to the external clients but it could also serve frontend traffic if it is so wished to do so. That is a file server can both replicate data to its storage and invoke its data to be replicated on another file server. This functionality where a replica machine is open to clients of the DFS can easily be removed. Each server in my architecture assumes that a file system server is running on the same machine (this was for implementation and testing purposes. I used my own machine whilst developing the "Distributed File System"). However, each server does not have knowledge of what port each server is running on. A server acquires the port information by conducting a ‘netstat’ command. A request is sent to all listening ports that are returned from the CLI command. If a file system server receives the aforementioned request it responds with an acknowledgement that it is a server part of the DFS network and internally takes note of the port the request came from. (This work could have all been removed if I had implemented a directory service) Each client facing server is replicated with data that is held in any server in the architecture. (I used different folders to denote the contents of a file server.) When a write occurs on one server the file that was uploaded gets replicated (uploaded) to all other servers in the network.

Locking

(Partial) Consistency is guaranteed by the locking mechanism I have implemented. When a client acquires a lock. All other servers containing the file in question is notified that a lock has been

acquired on that file. This prevents a client writing to or acquiring a lock to that file through the use of another server in the network. The lock is released when the file is updated. Once the file is updated the file it is unlocked and the update is propagated to all other machines in the network and is thus unlocked in respect to that server as well. It only provides partial consistency because if another client attempts to acquire a lock on the same file on a different server it will be granted as there will be a lapse in time until each server is notified of the lock acquired. This could be overcome be having a centralised server in charge of controlling locks (a directory service). If one client acquired a lock on a file and a second client came looking to acquire the same lock it will not be blocked, only informed that the lock is not free. This is in contrast to conventional understanding of mutex locks. In mutex locking, the second client would be put to sleep until the lock is freed. However, this implementation allows the client to interact with the server whilst the file is still locked. There is an edge case that my implementation does not cover. If a client were to break their connection with a file server the lock would remain in place. This means that the file that the client locked will forever remain locked.