Introduction

As a requirement for our enhancements, we maintained full compatibility with the base protocol declared in the assignment document.

Enhancement for File Backup

The peers, before saving a chunk, save a temporary chunk with the received file id and chunk no. During the wait time required before sending the STORED command, they are listening for STORED commands from other peers, incrementing the known replication degree. If after this time the replication degree is not equal or above the desired one, it then sends the chunk. Due to the fact of the delay time being random in each peer, there is high probability of not having excess of stored chunks.

Enhancement for File Restore

First each peer starts a thread which listens in a hard-coded port in it's own IP. A peer which receives a request for storing a chunk, after waiting a random time, sends the chunk for the requester's IP. If this one is able to process the request, it must send a response for the IP of the peer which sent the CHUNK. If the requester did not sent a message, the receiver (of the request) presumes it does not implement this enhancement, so after half a second from it sending the CHUNK by unicast, it sends it to the MC.

Enhancement for File Deletion

The initiator-peer of this protocol call, after sending the DELETE command, expects to receive responses to it, in equal number to the file's desired replication degree, so, it puts the file's identifier and replication in a map, and in random time intervals, it sends another DELETE command. The peers which receive this must only send a response to the MC after doing the required actions.

Enhancement for PUTCHUNK

A peer whom receives a PUTCHUNK keeps on listening for STORED command even after it sends a chunk (if it does so). It then throws a thread which in time intervals of one minute verifies if the known replication degree of the saved chunk is below the desired one. If this is true, it sends a PUTCHUNK; if not, terminates itself.

Enhancement for Chunk Removal

Even though not being implemented, we though about improving the solution for the REMOVED command. It involves creating a random hash key (or similar), putting it in a second header line. Then, each peer that receives the REMOVED command must check if the unique (with high probability) identifier of the message has already been received. If not, proceed with the normal behaviour, and then send a response. The initiator-peer must then count the received messages in a separate thread, which in given time intervals re-send the REMOVED command with the original hash while the count is below the replication degree of the moment when the first command was sent.

Enhancement for Resolving Inconsistencies in Chunks Replication

This is another feature that could be implemented to make the distributed system more robust.

In order to prevent inconsistencies of the chunks replication degrees amongst peers (due to actions being taken while some of them are down), we thought about implementing a token, held by one and only one peer at a time, representing the one that has the correct information. So, from time to time, it sends the information of the various chunks, authenticating itself as the token owner. If after a given time, say five minutes, there is no message, the peers presume that the token owner has shut down, they wait a random time and after that send a message reclaiming the token if they didn't ear another peer sending the message. After this, the new token owner proceeds as described above.

Another Enhancements

→ Select interface to connect