Project 1 Report

Distributed Backup Service

Integrated Master's Degree in Informatics and Computing Engineering

SDIS 2020/2021

Class 3 Group 3:

Prof. Pedro Souto
Prof. Hélder Castro

Bernardo Ramalho - up201704334@fe.up.pt
João Rosário - up201806334@fe.up.pt

Concurrent Execution of Protocols

For each Peer to be able to execute various protocols at the same time, each Peer has an object of the *java.util.concurrent.ScheduledThreadPoolExecutor* class. This object is responsible for executing threads, immediately or with some delay (useful in some protocols). Every channel is runned in a different thread and the interpretation of the message is done in a different thread from the one that runs the channel. This makes it so the Peer can not only receive messages in all the channels at the same time but can also execute various protocols at the same time. We also use threads in some protocol specific tasks.

To prevent errors when accessing the same data, in the Peer.PeerFolder class, we use an object of the *java.util.concurrent.ConcurrentHashMap* class which is thread-safe. We use this, for example, as an object that saves the perceived replication degree of each chunk, which is used in all the protocols.

Chunk Backup Protocol Enhancement

The main goal of this enhancement is to prevent the depletion of the backup space and reduce the activity on the nodes once that space is full.

The implementation of this enhancement is rather easy to understand. It consists of, before storing the chunk, the peer verifies the perceived replication degree of the chunk. The chunk is only stored if the perceived replication degree is less than the desired replication degree. When a chunk is saved, the peer sends the “STORED” message, which lets other peers update their perceived replication degree of the chunk.

This only happens if the peer is using the enhanced version, if not, the peer simply saves the chunk without caring about the perceived replication degree.

Chunk Restore Protocol Enhancement

The main goal of this enhancement is to prevent the chunk data from being sent over to a multicast channel which makes no sense since any peer that isn’t restoring the file is also receiving the chunk data. In order to implement this enhancement we made so that the peer that is going to receive the chunks acts as a server, and all the others are the “clients” which send chunk information to the server socket. The server

also is capable of handling multiple clients sending chunks at the same time.

By doing as stated above we get a clean implementation which also is capable of interoperating with other peers who don't use tcp, for example, a peer with no enhancements is sending the chunks through the multicast socket and the enhanced peer's are sending other chunks through the tcp channel.

Also as a final note the enhancement also waits between 0 and 400ms to send other chunks so that it doesn't send the same chunk as other peers but sometimes this interval is too short and thus multiple peers send the same chunk.

File Deletion Protocol Enhancement

The main goal of this enhancement is to, in case a peer who backs up some chunks of the file is not running at the time the initiator peer sends a DELETE message for that file, recover the space used by these chunks.

The way we solved this was by implementing two new messages. The message "VERIFYFILES" and the message "DELETETHISCHUNK". The message "VERIFYFILES" is sent by the peer that wasn't running at the time when the DELETE message was sent. When an enhanced peer receives this message he will check all the chunks that were deleted. Then, it will send a "DELETETHISCHUNK" message, which contains the name of the chunk that was deleted, for each deleted chunk.

In order for a peer to know which chunks it has deleted, we added a new ArrayList where each peer will save the name of the chunks that it deletes. This way, when a peer receives a "VERIFYFILES" message, it only has to loop through that array and send each value.