Universidade do Porto

Faculdade de Engenharia

FEUP

Distributed Backup Service

Enhancements report

by
José Martins - up201404189
Marcelo Ferreira - up201405323

Chunk Backup Subprotocol

As suggested in the project specifications, the protocol proposed can deplete the backup space rather rapidly, and cause too much activity on the nodes once that space is full.

In order to avoid this problem, “enhanced” peers observe the multicast control channel with the goal of counting the number of “STORED” confirmation messages of the respective chunk before sending their one. This way if the number of messages received is greater or equal than the desired replication degree, the peer discards the chunk and does not reply with a “STORED” message.

With this procedure the protocol guarantees the desired replication degree, improving massively the use of space in each peer.

File Restore Subprotocol

This protocol can be problematic if chunks are large and may not be desirable, because only one peer needs to receive the chunk, but we are using a multicast channel for sending the chunk.

To solve this problem, we stopped using the multicast channel as the communication method to get the chunk data. When a peer receives a “GETCHUNK” request, the chunk is sent directly to who has made the request using a TCP connection and then in order to avoid flooding the peer the application also keeps sending “CHUNK” messages to the restore channel but this time with an empty body, so that the other peers can still know when the chunk was sent and in this way prevent flooding the peer who is restoring the file with multiple messages containing the same content.

File Deletion Subprotocol

The problem presented on the project specification of this subprotocol happens if a peer that backs up some chunks of the file it's not running at the time the initiator peer sends a “DELETE” message for that file, in that case the space used by these chunks will never be reclaimed.

In order to solve this problem we added a confirmation message with the following structure:

“DELETED_CONFIRMATION <version> <peerID> <fileID> <CRLF><CRLF>”

Each “enhanced” peer sends using TCP this confirmation message to the peer who wants to delete the file, this confirmation message is sent when it receives a “DELETE” request and has successfully deleted all the chunks with the specified fileId in the request.

We are using TCP in order to assure that the message is received by the peer who has made the request.

With this in mind, the peer who has sent the “DELETE” request keeps tracking of which peers were storing chunks of that file and still have the chunk stored in it’s backup space, and then while the file is not fully deleted the peer continues to send “DELETE” messages in periods of 5 minutes, and also when the peer goes down and initiates after, it verifies if any of the files that it has backed up is still on a unfinished deletion process, sending afterwards another “DELETE” message, with the purpose of reaching the peers that still haven’t deleted the specified file, thus reclaiming the space that wouldn’t otherwise be reclaimed.

Reclaim Subprotocol

As suggested on the specification of the project this protocol have a problem that happens if the peer that initiates the chunk backup subprotocol fails before finishing it, in this case the replication degree of the file chunk may be lower than what is the desired by the user.

To solve this issue, it was added a flag to the metadata of the backed up file that stores if an error has ocurred while sending a “PUTCHUNK” message, in this case when the peer is initiated in the following time, it checks the occurrence of errors in any of it’s last backups repeating the backup protocol for all the files that have failed because of an unexpected shutdown of the server or because the replication degree wasn’t being bigger or equal to what was desired and the number of ”PUTCHUNK” request messages tries exceeded the defined maximum (in this case we are using a maximum of 5 tries).