Final Report on Serverless Distributed Backup Service Project

Distributed Systems

Filipa Manita Santos Durão, up201606640
Guilherme José Ferreira do Couto Fonseca da Silva, up201603647

Introduction

Having finished the proposed project, the group would like to inform that all the base protocols of the backup service were implemented. The service was implemented in order to support concurrent execution (described in the first section of this report). As for the enhancements requested, the group implemented the **Babkup** and the **Restore** enhancements. These enhancements both implement TCP communication protocol and both are described further on this report.

1. Concurrency

In order to achieve better run times of the system, the group implemented Concurrent Execution.

There are always three threads running: the **backup** thread, the **restore** thread and the **control** thread.

The **backup** thread only receives messages of the type **PUTCHUNK**. These messages send a Chunk to be stored on the other peers.

The **restore** thread only receives messages of the type **CHUNK**. This message is a response to the message GETCHUNK. As specified in the protocol description, the GETCHUNK message requests a particular chunk of data. The CHUNK message sends that requested chunk of data to the peer. This CHUNK message, however, can be of two different kinds, the **base protocol** kind and the **enhancement kind**. The base protocol kind follows the message structure requested in the project specification. As for the enhancement kind, the message to be sent using TCP, also has the fields *hostname* and *port*, to identify the destination of the message. The **TCP enhancement** will be explained further on this report.

The **control** thread receives all other messages needed for the program execution. They will now be specified and briefly described.

- The **DELETE** message, that identifies chunks that must be eliminated from all the peers of the backup system due to the elimination of the original file from the source peer.
- The **GETCHUNK** message, that requests a particular chunk of data from the backup system.
- The **REMOVED** message, that is sent by a peer when it deletes a certain chunk. This message identifies the chunk deleted so the other peers can check if the replication degree is maintained.
- The **STORED** message, that is sent by a peer when it stores a certain chunk.

- The **CANSTORE** message, that is sent by a peer when a PUTCHUNK message is received and the **TCP enhancement** is being used. The CANSTORE informs that the peer has the capacity to store the chunk and returns its *hostname* and *port*, so that the chunk can correctly arrive the destination.

Every time a message is received that demands access to the disk, a delay needs to be done or a TCP connection is established, an **Operation** is scheduled to perform that action. An Operation is a *runnable* that fulfills a certain purpose.

There are several types of Operations:

- **Message Transmission:** has two Operations, the **SendMessageOperation** and the **RetransmitMessageOperation**.
- **File Manipulation:** has two operations, the **SaveFileOperation** and the **DeleteFileOperation**.
- **Message Answers:** has two operations, the **GetChunkOperation** and the **PutChunkOperation**. These operations access the disk to get data and then call the SendMessageOperation.
- **TCP:** has four operations, the **putchunkTCPOperation** and the **StoredTCPOperation**, related to the backup enhancement, and the **getChunkTCPOperation** and the **getTCPMessageOperation**, related to the restore enhancement.
- **Others:** these are Operations that do not fit in the previous categories but are still important to mention. It has two, the **UnsubscribeOperation** and the **LatchedUnsubscribeOperation**. They unsubscribe to a type of message.

For the execution of the Operations, it was used a **ScheduledThreadPoolExecutor**. Not all Operations are needed all the time, so everytime one Operation needs to be executed, a thread is "picked" from the ThreadPool and the Operation is executed in that thread.

Each thread (connection) has a subscription table. There are multiple kinds of subscriptions, but they are all related (class - subclass kind of structure). There is the **Operation Subscription**, that represents the name of the message. Then there is a subclass from Operation Subscription, the **FileId Subscription**, that is an Operation Subscription to a specific file. Finally there is a **Chunk Subscription**, that is a **FileId Subscription** to a specific chunk.

Finally it is important to add that the concurrency in this system only supports the **simultaneous** backup or restore of a **maximum of 7 chunks** for a given file, to avoid network congestion.

2. Enhancement 1: Backup

The enhancement suggested for the **backup** protocol was a scheme to control the number of copies of the file in the backup system. Even though the User can define the number the **replication degree** of the file, that is the minimum number of copies to be saved in the system, meaning, all peers in the system can save the same file, largely surpassing the desired replication degree. This is obviously a enormous waste of disk space, and can even stop some peers from backing up their files while there are others overly-repeated.

For the resolution of this problem, the group used **TCP** communication protocol and instead of saving the file chunks in all peers, each peer that receives the PUTCHUNK message will check if it has enough space to keep the chunk. If then the peer verifies that unites all conditions to save the chunk, it will send the message **CANSTORE**, in which it will share its *hostname* and its *port*, so the chunk can be redirected specifically to it. Upon receiving this message the initiator peer will establish a TCP connection and proceed to send the chunk through it, then waits for the other peer to confirm the success of the operation.

The **CANSTORE** message structure is the following:

`CANSTORE <Version> <SenderId> <FileId> <ChunkNo> <CRLF> <Hostname> <Port> <CRLF><CRLF>`

After the peer has stored the chunk, it will broadcast a **STORED** message, just like in the base protocol.

3. Enhancement 2: Restore

The enhancement suggested for the **restore** protocol was a structure to stop the multicast channel from becoming flooded with CHUNK messages. A GETCHUNK message comes from a specific peer, arrives to all other peers and one of them that contains the requested chunk of information will answer with the CHUNK message. But why flood the channel with that CHUNK message when only one of the peers is it recipient? The answer was to create some kind of scheme that allowed the message to go directly from the source to the destination.

Similarly to the **backup** enhancement, this enhancement implements **TCP** communication protocol for the transmission of the messages. As in the regular restore protocol, the requesting peer sends a GETCHUNK message and awaits for a response. The difference lies on the response received, since in this enhancement the other peers with the requested chunk will not answer with the chunk but with a CHUNK message with their *hostname* and *port* (structure specified below). When that message arrives to the requesting peer, it will then proceed to establish a TCP connection with the peer of which *hostname* and *port* were received, and the chunk will be transmitted through that communication channel.

The **CHUNK** message structure is the following:

`CHUNK <Version> <SenderId> <FileId> <ChunkNo> <CRLF> <Hostname> <Port> <CRLF><CRLF>`