Distributed Backup Service for Internet

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Assignment 2

Final Report

29-05-2020

Bruno Micaelo	201706044
Mariana Neto	201606791
Vítor Gonçalves	201703917
Vítor Ventuzelos	201706403

Overview

The implementation developed expands upon the previous project, containing the same main protocols, namely backup, restore, delete, reclaim and state, and relies on RMI communication to provide an interface to the user.

The Chord design is followed, to allow for a decentralized system, and to ensure the scalability and fault-tolerance provided by it.

Multi-threading is heavily used to ensure concurrency.

The syntax of the application is: *java Peer <protocol_version> <peerID> <srvc_access_point> <port> <ipAddress_of_other> <port_of_other>*, where *protocol_version* is a string that specifies the version of the implementation to be used (currently only supports 1.0), *peerID* is an integer that identifies the peer, *srvc_access_point* is a string that specifies the RMI access point for the user interface, *port* is an integer that specifies the port to be used by the peer to connect with other peers, and *ipAddress_of_other* and *port_of_other* are respectively a string and an integer specifying a peer already in the Chord network to which the new peer will connect.

To facilitate testing, a few simple scripts were made, and can be found in the scripts folder. To work properly, these must be run in the src folder.

- *clean.sh* deletes all data saved by the implementation (backed up chunks and restored files)
- *compile.sh* deletes all .class files within the src folder and compiles the Peer application and the TestApp application
- *runPeers.sh* opens five separate xterms, initiating *rmiregistry* on one and Peers on the remaining four. The access point of these peers is “access_peer_1” through “access_peer_4” and they use ports 8000 through 8003. Note: in case your machine does not have xterm, an alternative that uses gnome-terminal was also developed (*runPeersGnome.sh*).

Protocols

Interface

The implementation reuses the TestApp interface from the first project, which connects to the main service via RMI and is implemented in TestApp.java.

The call syntax is the following: *java TestApp <peer_ap> <sub_protocol> <opnd_1> <opnd_2>*, where *peer_ap* is a string identifying the peer's RMI access point, *sub_protocol* is a string specifying the requested protocol, and *opnd_1* and *opnd_2* vary depending to the protocol. On each protocol, its specific syntax will be presented.

Additionally, the Remote Interface used can be found in the PeerInterface.java file, and includes 5 methods, one for each of the implemented protocols.

Backup

Syntax: *java TestApp <peer_ap> BACKUP <path_of_file> <replication_degree>*

The Backup protocol stores the specified file on the service, divided in chunks. The desired replication degree is not guaranteed if there aren't enough peers, or enough storage space, in which case the user is notified.

The bulk of its implementation is in mainThreads/BackupMainThread.java.

Restore

Syntax: *java TestApp <peer_ap> RESTORE <path_of_file>*

The Restore protocol downloads a previously backed up file from the service and places it in /peer_disk/peer<peerID>/restore/.

It is important to note that a peer can't restore files backed up by other peers.

The bulk of its implementation is in mainThreads/RestoreMainThread.java.

Delete

Syntax: *java TestApp <peer_ap> DELETE <path_of_file>*

The Delete protocol deletes all chunks in the service belonging to a specified file previously backed up by the peer.

The bulk of its implementation is in mainThreads/DeleteMainThread.java.

Reclaim

Syntax: *java TestApp <peer_ap> RECLAIM <max_disk_space>*

The Reclaim protocol changes the disk space provided by the peer for the backup service to *max_disk_space*. If backed up chunks have to be eliminated in this process, chunks of files with higher replication degrees will be picked first and other peers will be notified, allowing them to maintain the replication degree by backing up the chunks on other peers.

The bulk of its implementation is in mainThreads/ReclaimMainThread.java.

State

Syntax: *java TestApp <peer_ap> STATE*

The State protocol prints the state of the peer on the screen, listing files backed up by the peer and and chunks of files from other peers contained in storage.

The bulk of its implementation can be found in data/PeerDatabase.java, on lines 579 to 661.

Chord

Syntax: *java TestApp <peer_ap> CHORD*

Protocol that allows to know the status of the Chord, with information related to its predecessor, successor, list of fingers and list of successors.

The bulk of its implementation can be found in chord/Chord.java, on lines 468 to 496.

Concurrency Design

In order to guarantee concurrency in our project, we resorted to an implementation based on threads, so as to ensure we can execute multiple requests without having any concurrency issues. For each request that is received a Runnable object that processes the task is instantiated, and in the case of the BACKUP and RESTORE, a reference is saved in a Map, to allow further notifications when the relevant messages are being processed. This can be found in the Peer class (Peer.java), where upon calling a protocol via RMI, a thread is opened to run that protocol, leaving the Peer object available to run other protocols concurrently (the thread objects that run the protocols can be found in the mainThreads folder).

Scalability

To ensure the scalability of the system, we followed the Chord design (its implementation can be found on the three files within the chord folder. Thanks to this design, Peers may join the network freely, up to the set limit, and the maximum number of Peers can be increased by changing the value of the M variable (found on chord/chord.java line 34).

Fault-Tolerance

To achieve **fault-tolerance**, given that we have implemented a **decentralized design**, we have introduced the techniques suggested in the **Chord paper** related to this topic.

It was necessary to create a new structure in the Chord class, an **ArrayList with N NodeInfo entries**, which we call successors_list.

The value of N was obtained as **N = log (2 ^ M).**

Regarding its implementation, we follow, as I mentioned above, the Chord paper but also a presentation we found, which will be left at the end of this section. It was necessary to create a new function that did not exist in the initial pseudo-code provided, **updateSuccessorsList()**, which is called within the **stabilize()** function, so it will be called periodically. This list aims to store the first **N** active successors in the Chord and thus, in case my successor fails I have a list that I can use to obtain a new successor and this way the Chord ring is **unlikely to be corrupted.**

Within the **updateSuccessorsList()** function, which can be found in chord/Chord.java on lines 338 through 392, we iterate over the list, which will initially have **N NodeInfo's** equal to the node. At each iteration, we ask the node in position i for its **successor**, and **update** the entry **i + 1** of the list of successors with the **NodeInfo returned**. If no answer arrives within a period of 3 seconds or the answer is null, this node is deleted from the list and we add **"my node" at the end of the list**.

It is important to mention that whenever **position 0 of the list of fingers is updated, position 0 of the list of successors is also updated with the same node**, the same happens in the opposite case. In this way, we guarantee that the successor is always the same in both lists.

In addition to the creation of this function and its call at the beginning of each call to the stabilize function, there was also a small change in another Chord function,

closest_preceding_node. In this function, in addition to **iterating over the finger table**, we also iterate over the **successors list.** Therefore, we have the possibility to obtain two nodes close to the key passed as an argument. If **none of the nodes is null**, we check the **closest value** and this is the one r**eturned**.

Consulted resources:

- https://dl.acm.org/doi/pdf/10.1109/TNET.2002.808407?download=true
- https://www.kth.se/social/upload/51647996f276545db53654c0/3-chord.pdf
- first project of both groups