

2º Pratical Assignment - Report

Report – MIEIC

Course: Sistemas Distribuidos

Authors:

* Ana Rita Fonseca Santos –  [up201605240@fe.up.pt](mailto:up201707291@fe.up.pt)
* André Filipe da Silva Moutinho – [up201704889@fe.up.pt](mailto:up201704889@fe.up.pt)
* Eduardo João Santana Macedo – [up201703658@fe.up.pt](mailto:%75p%32%301%37%303%36%358@%66%65%2eu%70%2e%70t)
* Sérgio Bruno Rodrigues Dias – [up201704889@fe.up.pt](mailto:up201704889@fe.up.pt)

**Maio 2020**

**Academic Year 2019 / 2020**

**Índice**

[**1.** **Introduction** 3](#_Toc40824013)

[**2.** **Overview** 3](#_Toc40824014)

[**3.** **Protocols** 3](#_Toc40824015)

[**3.1.** **Backup** 4](#_Toc40824016)

[**3.2.** **Restore** 4](#_Toc40824017)

[**3.3.** **Delete** 4](#_Toc40824018)

[**4.** **Concurrency Implementation** 4](#_Toc40824019)

[**5.** **JSSE** 4](#_Toc40824020)

[**6.** **Scalability** 4](#_Toc40824021)

[**7.** **Fault-tolerance** 4](#_Toc40824022)

# **Introduction**

This report serves as a complement for the second practical assignment, having the purpose of specifying the design and implementation of the backup service. It also describes the extra features that were implemented to raise the project’s grade’s ceiling.

# **Overview**

The implemented solution allows for the backup, restore and deletion of files. There is also a state operation, which shows the client the storage status and the files that a give peer has initiated backup for.

JSSE was used for secure communication. Although the project members planned and attempted to use the more complex interface **SSLEngine**, the difficulties encountered in the process and shortage of time led to the decision of changing the communication mechanism halfway through the project development. Therefore, the final solution uses **SSLSocket** instead**.**

As for the scalability and fault-tolerance, those are ensured at the design level. The implemented design is decentralized and conforms with the Chord protocol/algorithm, which was incorporated to satisfy both requirements.

.

# **Protocols**

# **Backup**

Start here

# **Restore**

Start here

# **Delete**

Start here

# **Concurrency Implementation**

# **JSSE**

As mentioned earlier, this project does make use of JSSE for a safer communication. The security is ensured through the usage of Secure Sockets by the classes **SSLServerSocket**, **SSLServerSocketFactory**, **SSLSocket** and **SSLSocketFactory**, all from the javax.net.ssl package. Usages of this implementation can be seen in the file **Server.java**, which initializes the server and has loop which runs in the background waiting for new connections. Also, the file **MessageSender.java** has the functions used to connect and send messages to a peer.

# **Scalability**

Scalability of the application was ensured by implementing the Chord protocol/algorithm. This protocol revolves around a distributed hash table (DHT) to help in the resolution of unstructured names in the network. A DHT stores pairs (key, value) by assigning keys to different peers/computers in the network. A node only stores the values of the keys it is responsible for. Both values and peers are assigned an *m* bits identifier using consistent hashing, where 2m = maximum number of peers in the network.

Through the usage of this protocol, each peer only needs to keep track of around **log2 m**other peers, being **m** the amount of peers currently in the network. These peers’ information is stored in what’s called a **finger table**. The finger table helps in a way that querying for a key that is stored in the network is fast, since the number of nodes that must be contacted to find a successor in an *N*-node network is O(log N).

As mentioned earlier, implementing this protocol ensures scalability, since adding nodes to the network will have a low impact on the memory usage and the number of nodes required to query a key from the network.

Reference to important functions that implement chord (all in ChordNode.java):

* **join(InetSocketAddress)** : line;
* **notify\_successor()** : line;
* **notified(InetSocketAddress)** : line;
* **notified(InetSocketAddress)** : line;
* **find\_successor\_addr(long, Message)** : line

# **Fault-tolerance**

The design chosen was decentralized, which means that fault-tolerance was ensured by Chord’s fault-tolerance features. There are three threads running in the background that ensure the stability of the network:

* **FixFingersThread**: Every three seconds checks whether the peers in the finger table are all alive and updates the finger table in case of departure and join of nodes;
* **PredecessorThread**: Pings the predecessor and in case it doesn’t reply, sets it as null so that it can be updated later through the usage of a notify;
* **StabilizeThread**: Asks the successor for its predecessor and decides whether that predecessor should be the new successor, then notifies its successor, so that it can update its predecessor.

These threads are respectively implemented in the files: **FixFingersThread.java**, **PredecessorThread.java**, **StabilizeThread.java**, and only start running when the peer joins the network.