Serverless Distributed Application for Backing Up and File Sharing

Final Report

Mestrado Integrado em Engenharia Informática e Computação

Distributed Systems

Group 3, Class 4:
André Pires - ei12058
Filipe Batista - ei12068
Filipe Miranda - ei12021
João Bandeira - ei12022

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

June 6, 2015

Contents

1 Introduction

For this second assignment the group decided to develop a distributed application for backing up and file sharing, based on the first project developed in Distributed Systems.

The application presents a graphical user interface, requiring an authentication through login/register on the system or through "Login with Facebook".

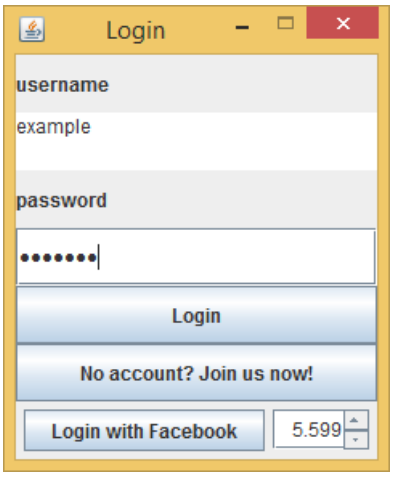

Figure 1: Login interface

After the authentication the user is capable of backing up files, restore, and share those with a group of friends.

In this report we will explore the project architecture and describe all of its functionalities. It will also address relevant issues to the project and present conclusions regarding the development of the project and the final project.

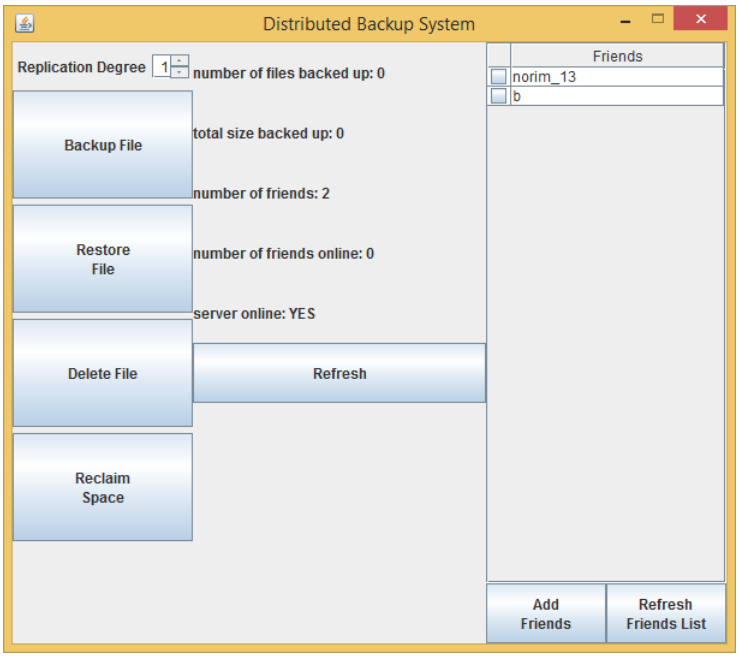

Figure 2: Main panel interface

2 Architecture

The application implemented is based on a server-peer and peer-peer interaction.

The server allows the authentication of the peers and provides a list of on-line peers, continuously checking, in a span of time, the users/peers on-line.

The user is required to log in into the application. The authentication can be done either using an application account or a Facebook Log In.

For an application account, the user must register, using an email and password. The information is then sent to the server where it is stored in the database, hashing sensitive information. After the registration, the user can Log In using the same credentials.

The user can also Log In into the application using a Facebook Account, where he/she must approve the application's access to Facebook's information like name, email and friends. Information of the user is then sent to the Server, where it is stored if there is no entry for the account on the database, or is validated.

Users are presented with a Friends List on the main panel. In the Friends List panel the user can add friends to his/her list, choosing from users registered in the system.

The peer allows the user to Back Up a chosen file with a selected replication degree. The file is divided in chunks and sent to the on-line peers, provided by the Server. The peer then stores the peers' id who saved chunks of the Backed Up file.

The user can also Restore a previously Backed Up file. Using the list of peers' ids who saved chunks of the file, the peer sends an IP request of those

peers, to the server and checks if they are on-line. Then, the peer requests, directly to a peer who has the chunk, a get chunk to recover the desired file.

For the reclaim function, the user first specifies the new folder size he/she wishes. The peer then selects chunks randomly to remove them until the new folder size is reached. For each chunk removed the peer sends a REMOVED message to the original user.

The peer has an always running thread called *ConnectionListenerPeer* whose function is to continuously read input from the user's port. This thread is able to read several types of messages, execute an action if necessary, and respond to the message.

This thread can parse ISONLINE messages, sent by the server to check if a peer is online, and send an OK message in response.

A message PUTCHUNK is parsed by the thread, where it is checked if the chunk in the message is already stored. If it is not, the chunk is stored in the receiver, if it has space in the backup folder. In the end, the thread sends a STORED message to the original transmitter.

In case of a DELETE message, the peer checks if it has chunks of the specified file to be deleted. If so, deletes them and sends an OK message, or a NOTOK message if it has not.

When a BACKUPFILE is received, the thread will try to send the file specified in the message to a peer.

For file sharing with friends the group implemented the following protocol:

1. The peer starts by requesting the last known IP of a friend
2. Sends a BACKUPFILE message to a friend using his IP. The message body contains the name and size of the file (in bytes). The BACKUPFILE message has the following format:

BACKUPFILE <Msize><MyID><CRLF><CRLF><Fname><Fsize>

3. The receiver (friend) after receiving the BACKUPFILE message, creates a thread (*FileShareReceiveThread*) in which it opens a socket, to receive the new file.
4. The receiver then sends an OK message, whose body contains the socket port opened in item 3.
5. Receives the data until the end of the file.
   (a) The file is sent line by line (each line starts by "LINE#").
   (b) When the peer receives a LINE, sends an OK response.
   (c) When the peer receives am "END#", it means that it received the hole file. and responds with an OK message.
   (d) In case of any error, the peer responds with a NOTOK message.

This thread can also parse GETCHUNK messages, where the peer responds to the transmitter a CHUNK message with the chunk specified on the original message.

Messages' Formats:

NOTRESPOND DELETE <FileID><UserID><CRLF><CRLF>
PUTCHUNK <BodyLength><FileID><ChunkNum><RepDegree><CLRF><Body><CLRF>
STORED 0 <FileID><ChunkNo><CLRF><CRLF>
REMOVED 0 <FileID><ChunkNo><CLRF><CRLF>
DELETE 0 <FileID><CLRF><CRLF>
GETCHUNK 0 <FileID><ChunkNo><CLRF><CRLF>
GETALLUSERS 0 <CLRF><CRLF>
GETONLINEUSERS 0 <CLRF><CRLF>
OK 0 <CLRF><CRLF>
NOTOK 0 <CLRF><CRLF>
ADDFRIENDS <BodyLength><UserID><CLRF><Body><CLRF>
BACKUPFILE <Msize><MyID><CRLF><CRLF><Fname><Fsize>

3 Implementation

The project was developed in the Eclipse IDE for Java and using SVN for subversion control, in a Windows environment.

For the implementation of this project it was used largely the Java language and SQL. Java was used for the implementation of servers, peers, protocols and user interface while SQL was used for the management of databases.

For an easier implementation of the graphic user interface, the group used the SWING library.

It was used the Gson library for the encoding in Json format of information sent in some messages.

For an easier manipulation of arrays the group used the Apache Commons library.

The application dispatches the protocol messages through a Secure Sockets Layer (SSL) from Java. The function *sendMessage* was implemented in both PeerNew.java(l.434) and Server.java (l.83), and allows establishing an encrypted link and sendg messages between a server and a peer or between peers.

4 Relevant issues

In terms of **security** we have implemented the following improvements (as proposed):

1. **Secure connections** - The application dispatches the protocol messages through a Secure Sockets Layer (SSL) from Java, and establishes an encrypted link when sending messages between a server and a peer or between peers (PeerNew.java(l.434) and Server.java (l.83)).

2. **Peers authentication** - Implemented a sqlite database system and used a browser with https requests for the Facebook login.

3. **Chunk confidentiality** - Chunks are read-only and the respective folder is hidden, so users can not easily access them.

```java
/**
 * Creates a physical chunk
 * @param chunk
 * @throws IOException
 */
public static void materializeChunk(Chunk chunk) throws
    IOException {
  String tempname = chunk.getFileId() + "_chunk_" +
      chunk.getChunkNo();
  File f = new File("files\\backups\\" + tempname);
  FileOutputStream out;
  out = new FileOutputStream(f);
  out.write(chunk.getByteArray());
  out.flush();
  out.close();
  //make chunk read-only
  f.setReadOnly();
}
```

```java
File dir4 = new File("files\\backups");
if(!dir4.exists()) {
    dir4.mkdir();
   FileManagement.hideFile("files\\backups");
}
```

4. **Possibility of choosing receivers** - This was implemented for file sharing only and not for backing up files.

In terms of **fault-tolerance**, we have implemented the following improvements:

1. If any message does not have a response, the same message is re-sent up to 3 times.
2. In the delete protocol, if a peer who has a chunk of the file which is being deleted does not respond, the initiator peer sends a NOTRESPOND message to the server. That being sad, when the peer who didn't respond logins, the server re-sends the lost message.

5 Conclusion

The group would like to point out, that despite having chosen to implement a similar program to the first project, the group had to re-make all of the functionalities.

The group member's shared tasks when implementing the program. It is difficult to delimit the actual worked performed for each member, because there isn't a specific task made by a single member.

Nevertheless, as requested, each member will talk about what he focused on.

- **André Pires** - I focused mainly on the Backup protocol and Delete protocol and respective GUI's, as well as "converting" the first project messages to this project. Also, created the process of checking online users, among other things. That being said, my work was 25% of the groups' work.

- **Filipe Gama** - I was responsible for the implementation of the "Restore" and "space reclaiming" protocols, including the respective GUI's. I helped with other things such as creating a state machine for reading the terminating sequence for messages, among others. That being said, my work was 25% of the groups' work.
- **Filipe Miranda** -
- **João Norim** -

6 Bibliographic References

SWING Documentation - http://docs.oracle.com/javase/7/docs/technotes/guides/swing/

Apache Commons Documentation - https://commons.apache.org/proper/commons-lang/javadocs/api-2.6/org/apache/commons/lang/ArrayUtils.html

Gson Documentation - http://google-gson.googlecode.com/svn/trunk/gson/docs/javadocs/index.html

Facebook Developers - https://developers.facebook.com/