LogBook

Samu Tamminen Paloma López Ryan McDyer
Diego Salmerón

First meeting

In order to desing our application, we have decided to use Java as programming language, and GitHub and Trello to share our thoughts.

Trying to decide how to implement our project, we first came up with the idea of having a master node, where to upload our files and share what we wanted. It would be the manager among the computers, taking each user's files and merging them in a common filesystem, then replicating it to every user's local machine.

After discussing it with the professor, we changed this vision for a decentralized system, where peers directly interchange the files.

Second meeting

We had agreed that all machines would act as peers. This carries the problem of synchronizing all files at the same time without a central node storing the "real" version of the filesystem.

We have agreed on the following file synchronization process:

1. Each node will inspect its own shared filesystem and return a description of it in JSON format
2. This JSON will be sent to all the other peers, and JSONs from them will be received.
3. Each node knowing what its peers have in their filesystems, they will start the operations in order to have the same version of the filesystem: Request the sending of files they don't have, erase those that other machines do not have, etc.

To achieve this, we add the following tasks to the TODO list:

- Implement a program that given a filesystem, returns its description in JSON format.

- Start studying about how to connect several machines over the network with the RPC/RMI middleware.

Third meeting

- The JSON description of a file system is already implemented, file changes are already detected inside a directory, taking also into account the subfolders that may exist inside of it (i.e. the algorithm is recursive).
- Now starting with the exchange of files between folders. Firstly trying to do it locally, with both directories in the same computer.
- Decided to keep always the newest file in case of conflict.
- A client and a server have been created using RMI, but we have problems with the interconection among computers. Able to do it between MACs but unable with Linux.

Fourth meeting

We still have problems with the communication among machines. In order to avoid losing time on that aspect, we started focusing on the web interface functionality. We have, then, three parallel working groups:

- One focused on the filesystem description and the interchanging protocol. Testing is done over local files at the moment
- Another focused on obtaining the connectivity among computers through RMI.
- The last one working on the web interface.

Web interface development

Here we will specify our first approach to the web interface functionality:

The web interface will be installed in a server running on only one of the nodes. Its objective is that users connecting from machines different from the one where its shared folder is can still be able to make changes to the shared filesystem.

In order to reach that, the node storing the server must initiate the filesystem convergence process when an access to the webpage is required, so as to offer the latest version of the filesystem. Then, the user will make changes through the interface and commit those changes if he wish. Finally, the node will start a new convergence process to store those changes on the shared filesystem.

Fifth meeting

The approach we took to the web interface in the last meeting was not the one required.

Web application and the filesystem application should run in different nodes and communicate between them over the network.

For this communication we are going to use the Spark framework.

Web interface development

We decided to use Java servlets and JSP as the technologies for implementing the web application and Tomcat 8 as the servlet container.

When initiated, the web app will request a filesystem node a JSON description of the filesystem, and will show it to the user by pages dynamically generated using JSP.

Every user action regarding a change in the filesystem will be stored in a JSON file describing those changes. When the user commits, this JSON will be sent to a filesystem node, which will make the changes on its local folder. The rest of peers, which are watching the others' shared folders, will initiate the convergence mechanism when the changes are detected.

Sixth meeting

In this session we came up with a problem our previous approach had: The JSON describing the files would be a problem when dealing with file additions: A link should be added for those cases, filesystem node would have to request the file to the web server, etc. These problems, and others that may arise, turns the JSON solution into an over-complicated way of working out the problem.

We thought of using simple HTTP requests for that: A GET request for deletions and a POST request for additions. We still have to think harder on this aspect.

We also decided to change when the web application sends reports to the filesystem node: Instead of waiting until the user commits a list of changes, as initially conceived, changes will be sent to the filesystem node as they are done.

Seventh meeting

We implemented Spark web server that serves information about filesystem for the web client and applies the changes made from the browser. After that we

reliazed that we do not need two servers; one Spark/Jetty implementation could read the data from filesystem and serve the HTML/JavaScript for the server.

Thymeleaf-templating library helped us to handle JSP rendering.

Diaspora development

After christmas we continued developing in our homes. For playing music files, we created extra routes on server that sniffs all music files on filesystem. That implementation is not the most efficient, if there are lots of files in the system. It could be improved by keeping track of the locations of music files in some data structure. That data structure could be updated every time the filesystem changes. Music songs are showed in the browser by JavaScript front-end made with React.js library.