Uppsala University

Computer Networks and Distributed Systems, 10c

Project, 1c

Authors:
Ashraf, Pouya
Billman, Linnar
Boström, Carl
Emrén, Krister

December 21, 2015

Contents

1 Summary

This project had the aim to explore how one may approach the task of creating a system that runs on a distributed system, and how to solve the many potential problems that may occur with that task.

2 Introduction

For this project we chose to implement a distributed buffer of sorts, which in turn relies on a distributed server polling mechanism, which ensures that the system will not fail when the designated server crashes/goes down (when the current server crashes, or loses connection with the other connected nodes, a different node will take the server responsibilities).

3 Original Design Idea

Initially, the project was envisioned as a distributed command line buffer, which also included a distributed server mechanism. The purpose of the distributed server mechanism was to ensure that the system would be able to recover by itself, in case the node currently acting as the server was to crash/get disconnected from the network. This was to be done by the server, which would with a fixed delay send the most recent information regarding the structure and hierarchy of the network to all connected nodes.

4 Revised Design

The revised design was simply a more detailed list of the original design idea.

Sever part:

- Has the responsibility to order the connected nodes by rank, so that the correct node will assume server responsibilities in case the current server goes down.
- Should always have the latest version of the material being edited, to be able to quickly distribute it to the other connected nodes.

- Decide whether to grant or deny a node access to the shared resource.
- Needs to keep track of which locked segment belongs to which node.
- Regularly checks to see if a node is still alive so that it can, if possible, unlock access to a locked resource.

Client part:

- Continuously listens to the server, to be able to quickly respond in a correct manner to different network events.
- Regularly saves the information from the server to update its local text file.
- Needs to ask permission from server to edit file at a given segment. If this is granted. The client may edit the file. When finished, the client will tell the server so that the server can update the information, distribute it to the other nodes, and ultimately unlock the resource.

5 The System

The main parts of the system are outlined below.

5.1 Node.java

This is the main part of the system. The Node class has two parts; one server part and one client part. This makes the node able to act not only as client, but also as server if need be (so that we don't have to instantiate a new object when we want to switch between client mode and server mode). When a node is created, it receives a rank depending on the number of existing nodes in the system (starting from zero). For example if there are 7 nodes in the system, a new connection will receive rank 7. The node with rank 0 acts as server. The new node also receives a list of socket information to all the other nodes. It is designed this way so that if the server were to die, the next node in the list (rank 1) would easily be able to take on the server responsibility. However, we did not have the time to implement the "server switch" (see section 6.3).

The client side of the node class is designed to communicate only with the node currently acting as the server, while the server side communicates with all the other nodes.

5.2 LockList.java

LockList is a class designed as a linked list, with the sole purpose of keeping track of which nodes have requested write access to which parts of the text. For instance, if node no.3 requests write access to a file at offset 450 bytes, and length of segment 100 bytes, other nodes will be unable to write information in that segment of the file (between offset 450 and 550 bytes) until node no.3 forfeits its privileges.

5.3 EditPane.java & EditWindow.java

EditPane and EditWindow are classes designed to display information to the user. This information consists of the file currently being edited, status/error messages and means to cancel the changes, or to submit them to the server. The graphical interface is entirely cross platform, since it is built on the Java Swing framework.

6 Results and Discussion

The resulting prototype is a distributed text editor system where one node acts as server and every client has a text window where you can highlight the part of the text you want to edit. Doing so locks the text region for your use only. When finished editing, you press the submit button to send the changes to the server, unlock the region, and update every client's text window. Each node has the capability to act as server, in this prototype however, only one node will act as server as the polling system is not yet implemented.

6.1 What we did right

What we did right was structure, planning and priority. We began the project by thoroughly planning all the parts of the system, and making clear priorities. We decided which parts and attributes of the system were more important than others. Even though we did not have the time to implement

everything we had initially planned, we feel content with the decisions we made throughout the project. Thanks to the planning we made we have also laid the ground work for some further development (see section 6.3).

6.2 What could have been implemented better

In the current prototype, the main part which could have been implemented in a better way is the LockList. This class was designed to save all the current locks in a linked list data structure to keep track of them to be able to update the information of the locks in case of text edits. If one node updated its text, the LockList was to update the offset of all the locks that occurred after the edited region so that they would still cover the correct part of the text. However, this did not work as intended and did not update the offsets correctly. Another small detail which could be implemented with more finesse is the case where the text window (or selection) is empty. In the current prototype one needs to have a minimum of one (1) character in the file to be able to edit it at all.

6.3 Possible Further Development

The first possible further development was the part we planned, but didn't have the time to implement; the polling system making it possible for another node in the network to assume server responsibilities in case the current server goes down/disconnects from the notwork. We have designed the system to be able to handle this. Each node has the capability to act as server, and each node has the capability to switch communication to the new server. However, the current prototype has no way of knowing when the server has died and the nodes aren't capable of doing the transition from client to server mode.

Other further developments are, for instance the design of the GUI/text editor, which could be more user friendly. It would also be nice to implement some kind of programming support for the text editor, e.g. syntax highlighting and automatic indenting (which in the end could be quite complex to implement, since different languages have different syntax, and use indentation in differing ways).

7 Conclusions

The current prototype is not as refined and advanced as we had hoped. With more time on our hands we might have been able to implement something closer to our initial plans/ideas, never the less, we are happy with our performance and are quite pleased with our product so far. We also feel that we have learned a lot throughout this project. We have touched the surface of the many possibilities of distributed systems.