**IN5020 Distributed Systems - Assignment 1**
Team members: Caroline Santos Alvær and Andreas Askim Vatne

**Implementation and Design**

The implementation of the client and server was mostly provided by the assignment text. However, there were certain design choices that were made by us.

*Cache*

The cache for both the server and the client was created using a LinkedHashMap. This gave us the necessary functionality to be able to update the cache accordingly and release the least recently used entries as necessary.

At the end of the program execution the client creates the server_cache.txt output file by calling upon the remote method getServerCaches in the proxy server. We decided this was the best solution seeing as the proxy server could easily bind to the stub of each server and access their individual caches, and return this as a list to the client.

*Client*

On the client side, we decided to place every remote method invocation in a runnable task and place every task within its own thread. The thread was then added to an array list of threads. To avoid that the main thread executes the calculation of the average times and creation of output files before the pool of threads were completed, we used the .join method on all threads created to ensure their completion before the main thread was allowed to resume.

*Server*

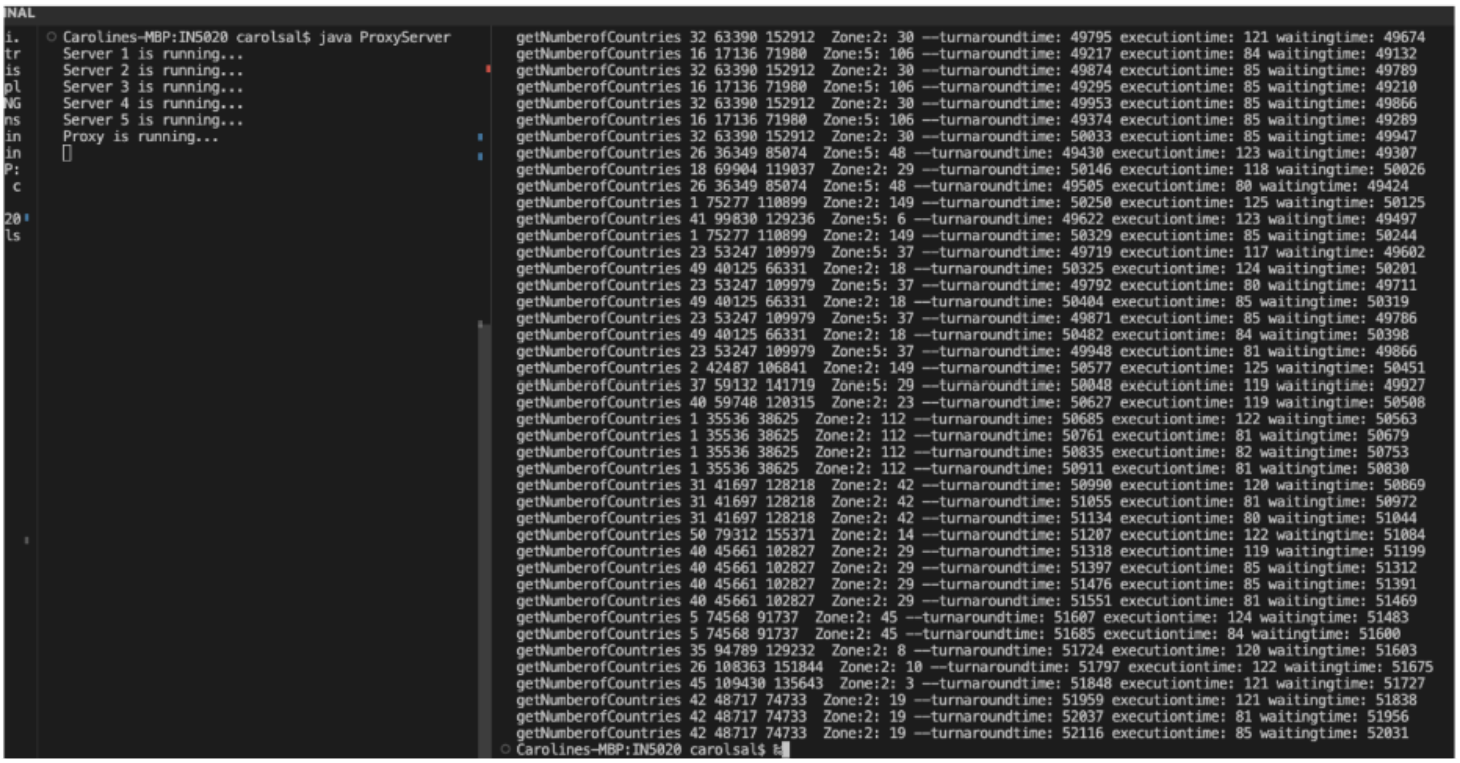
In the server and client class we implemented a simulate latency method to simulate the communication latency that would exist in a distributed network.

*ProxyServer*

The proxy was the part of the assignment where we were most free to do as we please regarding the design and implementation. In the proxy class we created an inner class called ServerNode with attributes to keep track of the workload of the different servers, as well as how many tasks the proxy had assigned to each server. The proxy server then started each server and binded each server to a name in the local registry.

To keep track of the workload of the servers without disturbing their execution of the client queries we implemented a thread pool with 5 threads that are reused. When a task to check the server workload is created, it is submitted to the executor object and one of the available threads handles the task asynchronously.

**Screenshots of client**




**User Guide**

The distributed application can be compiled and run in the following two ways:

**1 - Manually compile and run the application**

Compile: javac *.java
Run: java ProxyServer exercise_1_input.txt

**2 - Create a deployable jar file and run the application**

Create a directory called bin
Compile: javac -d bin *.java
Jar-file creation: jar cf assignment1.jar -C bin .
Extract Manifest file: jar xvf assignment1.jar META-INF/MANIFEST.MF
Add main class path to manifest file: Main-Class: ProxyServer
Save changes to manifest file: jar uvfm assignment1.jar META-INF/MANIFEST.MF
Run: java -jar assignment1.jar exercise_1_input.txt

**Division of workload**

Caroline and Andreas did 50/50 of the workload, co-wrote the whole code.