Université Paris Sorbonne Nord
Institut Galilée

Master 1.
Distributed Java Programming
Project

In this project we will implement a simple distributed system for downloading files and use it to simulate the impact of failures in the communication channels.

Project Description

When a client contacts the server, it may ask for a list of files stored in the server. Then, using the identifier of one of those files, say f, the client may ask the server to transmit that file. Since file f can be potentially a "big" file, the server cannot sent the whole file in one shot. Instead, it sends the file in blocks of a given size B[1]. When the client has received all the blocks, it sends to the server the MD5 of the file downloaded. If the MD5 coincides with the one stored in the server, the server assumes that the client c has a "good" copy of the file, and the server stores the client c in its list of "trusted" clients for the file f.

Since the files we are dealing with are big, and the clients are impatient, a client may launch several requests to download, concurrently, different blocks of the file. The number of such concurrent downloads is a parameter D_c of the client. Note that, in this case, the client cannot know the order in which the blocks will be received.

The server runs in a machine with very limited capacity where many bad things may happen:

1. The number of requests that the server can handle in parallel is limited by a parameter C_s of the system. This means that, if there is already a number $n = C_s$ of requests being processed, the next request that arrives is queued and it has to wait.

[1]Variables in upper case denote parameters of the system.

2. Every T seconds, with probability P (two new parameters of your system), the server "unnexpectly" close one of the connections with a client currently downloading a block. This will allow us to simulate failures in the system.

To deal with the above mentioned problems, and to improve the throughput of the system, the server makes use of the clients of the system as follows. If a new request arrives, and the capacity C_s has been reached, the server tries to *delegate* the request to one of the trusted clients (if any). Since clients do not want to be bothered by serving other clients, the server must first contact the trusted client c' that, in turn, with probability P_c accepts the request. In that case, c' sends to the server a single-use token[2] that can be used by the client c to download a number N of blocks from c'. If the client c' refuses to help the server, then the server inevitably adds the request from c to the queue.

Some extra non-functional requirements

1. All the information needed to execute your programs (client and server) must be provided by *parameters* of your `main` classes. For instance, one should be able to launch a client as follows:

   ```
   java Client --file="id-file" --DC=4 ...
   ```

2. Use Java Logging [3] so that all your programs leave a log with information about the execution of the system.

3. Think very carefully how to synchronize the different threads of each class and the communication mechanisms. For this project, you **cannot** use RMI. Ask yourself questions such as: Which methods need to be *synchronized*? What happens if the expected message does not arrive? What if the MD5 of the downloaded file does not coincide with the one of the server? etc.

4. In order to avoid dealing with "big" files in your experiments, you can always test your code with "small" files and choose small blocks (e.g., choose the parameter B –the size of the blocks– to be $1KB$).

Deliverables and Rules

The project must be developed in groups of 3 or 4 students (no exceptions). Only one student per team must submit on Moodle a Zip file containing:

1. The source code, fully documented using Javadoc. Write in the header of the file a precise description of the purpose of the class and your design choices. For instance, mention that "*this class uses the protocol X in order to communicate Y and the data structure D is protected using the synchronized method M, etc*". You will loose **several points** if that information is not included.

[2] This means that c' accepts a download request from c only if c provides the token generated by c'.
[3] `https://docs.oracle.com/en/java/javase/11/core/java-logging-overview.html`

2. A detailed README file explaining how to compile and use the system. You must provide a main class `Test` that uses a `ProcessBuilder`[4] to launch several instances of the Java virtual machine (in the same computer) to run the server and some clients. I will use that class to test your code. For instance:

   ```
   java Test --clients=4 --DC=4 --P=0.3 ....
   ```

3. This project is a good opportunity to test that your code can be deployed in different machines. Hence, you have to add to your zip file the log files that resulted when you performed a test in the machines of the University[5]. I expect that you run at least 3 clients, all of them in different machines.

4. One of the objectives of the project is to evaluate how the system behaves in the presence of failures. Hence, you must test your system under different scenarios (number of clients) and parameters. For each scenario, run your system at least 10 times[6] and report the mean of:

   a) the time taken for all the clients to complete the download of a specific file;

   b) The number of times the server closed a connection;

   c) The number of times a client "helped" other client in their downloads; etc.

   Your zip file must contain a PDF with plots of the data obtained in your experiments. For instance, you can plot the mean of the time taken for all the clients to download a file while varying the number of clients considered; also, fixing the number of clients and varying one of the parameters; etc. The more experiments (and plots) you perform, the better.

5. The final score is **individual** and it heavily depends on the knowledge of the student about the code submitted. This knowledge will be assessed with technical questions during an "exam" on May 2 2025.

6. It is strictly forbidden to communicate with other groups by any means. Fragments of code slightly similar will be considered fraud and severely punished.

7. If you are planning to use extra packages, besides those included in the Java standard library, you have to send an email to `olarte@lipn.univ-paris13.fr`. Non authorized packages will not be accepted as part of the solution.

8. The deadline for sending the files on Moodle is April 30 at 8AM (no exceptions).

[4] `https://docs.oracle.com/javase/8/docs/api/java/lang/ProcessBuilder.html`

[5] The logs output by your programs must include the IP and port used by the client and server in all the requests.

[6] In fact, in order to have statistical significance, one should run the system more times in a Monte Carlo simulation (see e.g., `https://en.wikipedia.org/wiki/Monte_Carlo_method`). For time restrictions, we will run it only 10 times per experiment. Be aware of modifying the seeds of your random generators each time.