LINFO2241-Project
Part 1: Measurement

Gorby Kabasele Ndonda, Viet-Hoang Tran

November 2021

Introduction

Imagine you have a company with customers using one of your products, a client-server application. When looking at the performance of such an application, one is typically interested in two problems:

- What response time do my customers experience? If the response time is high, what is the reason for this? Is the CPU or the disk in the server too slow? Or maybe the network connection? What about memory usage?
- Assuming I get new customers in the future, how many customers can my application handle before the response time becomes too high?

In this project, you will analyze the performance (e.g. response time) of a simple client-server application that you will implement in Java. The details of the application are given in the next section. We ask you to write two versions of the server, a simple version and an optimized version. The way how the server is optimized is left to you. The goal of this work is to measure the performance of the server in a systematic way and to show how the optimizations you have chosen impact that performance. You will later compare the measurement results with results obtained from a queuing station model.

Warning: In your experiments, the clients and the server **MUST** run on different physical computers with a network connection between them. Alternatively, you can have clients and the server as virtual machines on the same physical machine **ONLY** if you explain how you emulate the network (with tools like netem). If you are emulating network conditions, you have to choose parameters of the emulation like the latency. The value of those parameters depend on many things like how far (geographically) the two hosts are from each other. For example, the latency between two hosts in Belgium will be lower (e.g 10ms) than the latency between a host in Belgium and one in the United States (e.g 100ms).

Applications Specifications

The application that you have to implement allows clients to decrypt files. The application works as follow: A client holds an encrypted file that is protected with a password. The encryption is symmetric, meaning that same key is used

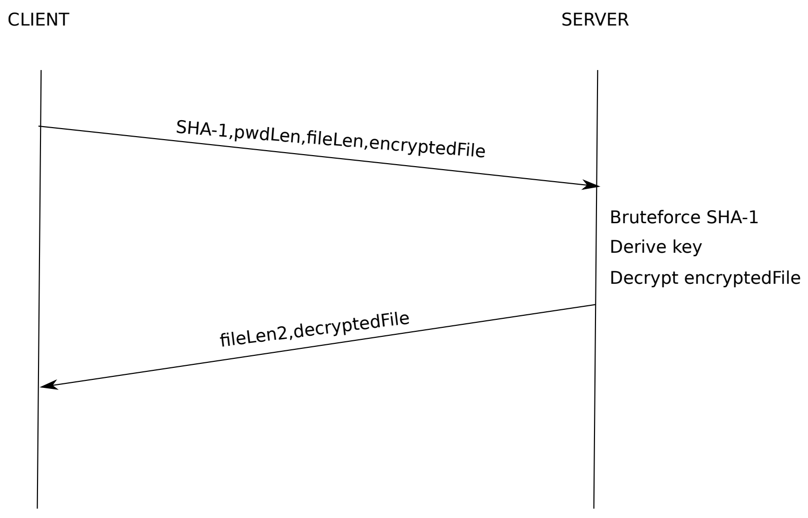

Figure 1: Exchange between a client and the server

for both the encryption and the decryption. The client does not have the password but he/she has the SHA-1 hash of the password and the length of the clear password. The client uses the application to decrypt the file by sending the encrypted file, the hash of the password and the length of the clear password. With this information, the server bruteforces the hash to find the password and then decrypts the file before sending it back to the client. Figure 1 shows an example of exchange between a client and the server. In this example:

- `SHA-1` is the hash of the password.
- `pwdLen` is a number representing the number of characters in the password that was hashed.
- `fileLen` is a number representing the length in byte of the file to decrypt.
- `encryptedFile` is the file to decrypt.
- `fileLen2` is a number representing in length in byte of the file once decrypted.
- `decrypted` is the file decrypted.

As mentioned the application must be implemented in Java, to help you, we provide you some code that you can use to implement both the client and the server. You are free to use the code as you see fit or change it completely as long as your implementation follows the specification. You are also provided a list of passwords and several files that you can use to create a workload. These are resources meant to help you, it is mandatory to used them.

The code examples and the test datasets can be found on Teams.

Task 1: Implementation

Write a simple server that accepts requests from a client through the network (similar to a web server). The server should open a socket on a certain TCP port and wait for incoming requests from the client. In Java, this can be implemented with just a few lines of code thanks to the java.net.ServerSocket class. See here for a short introduction:

- `http://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html`

Hint: Note that the constructor of the ServerSocket class allows you to define a so-called backlog. This is basically the maximum queuing capacity of the operating system's queue for incoming TCP connections. That could be important for task 3.

Your server should be multi-threaded, i.e., it should be able to process several requests at a time. The choice for the number (at least two) of threads used by the server is left to you. To achieve this, you should look at how to use thread pools in Java. In this project, you have to write two versions of the server:

- The basic version of the server perform a straight brute-force to find the password of the file.
- For the second version, we ask you to modify the basic version to improve its performance. You are allowed to modify the server as you see fit as long as it satisfies the specifications. To optimize your server, you must think about how to speed up the brute-force process

Task 2: Measurements

Measure the average response time of your system. For the measurements, we define the response time as the delay (as seen by the client) between sending the request and receiving the complete response. Obviously, the response time will depend on how many requests the server receives and how "difficult" the requests are. For example, requests with complex regular expressions need more CPU time on the server. And some requests might result in big responses with a lot of data sent back to the client. To do measurements, you should not send requests to the server by hand (that would take too much time for you). Instead, you should implement a small client application that sends requests at random times. For the inter-request time, choose some distribution. To send multiple requests to the server at the same time, you can either write a client application that opens multiple TCP connections, or the client only opens one TCP connection and you start the client application several times in parallel. The client should not wait for a response before sending the next request. It's perfectly possible that a request is sent while another request is still being processed. Try to go up to 50 to 100 client connections. Show measurement results for combinations of

- different request rate,
- different difficulties

Think about different difficulties and how they would affect the performance of the server in different ways (e.g more CPU intensive vs more network intensive). Choose reasonable values for the parameters: obviously, a request rate of 1 request/day does not produce interesting response time results. Show plots and discuss the results. What is the most important factor (CPU, network,. . .) in the response time, depending on the chosen parameters? Here are some tools to measure the network and CPU load on Linux:

- `http://www.binarytides.com/linux-commands-monitor-network/`
- `https://www.tecmint.com/command-line-tools-to-monitor-linux-performance/`

The measurement must be done for both versions of the server, you have to show that the optimized version is (hopefully) better than the normal one and explain why. We expect you to explain what modifications you have done to the basic version, why you chose to make the said modifications and what you did expect to improve by making these modifications.

Expected Output

You should hand in the source code of your implementation (no binaries) and a written report in a zip file. The deadline for the complete report (part 1 *and* part 2) is December 24. The complete report should have 4 to 5 pages (A4, 11pt, pdf format, no cover page). You can plan around 3 pages for the first part which should contain

- a description of the implementation, including the client
- a description about how you optimized the server and plots showing the impact of the optimization process
- a description of your measurement setup: The used hardware, network, and software to do the measurements
- a description of the workload you used in your experiments, i.e. how the clients generate the requests.

Don't forget to include your names, email addresses and NOMA. Keep your report concise. Don't waste space on long introductions, a table of contents, etc.

Evaluation Criteria for the first part

- Quality of code (comments, correctness, etc.)
- Quality of report (reasoning, language, readability, clearness of presentation)