Project: Measurement & Modeling

66732000 Manuelle Ndamtang
91242000 Antoine Hoebaer

1 Implementation of the project

1.1 Description of the tools

Failing to have physical machines to perform the simulation of a network, the used tools for the creation of the virtual machines are Virtualbox as hypervisor and Vagrant as tool to build and manage them. Therefore there are 3 virtual machines using Ubuntu 16.04 as Operating system:

- The server with 2Gb of RAM capacity and 2 cores
- 1 "simple" client with no latency.
- 1 client with the latency set to 10ms.
- 1 client with the latency set to 100ms.

Each client is configured to have 1Gb of RAM and 2 cores.

The project has been developed in Java and the configuration of the network is search that each virtual machine has 2 network adaptors. The first one is Nat-configured to enable the traffic through the Internet. And the second one, which is hostonly configured. It's used by the private network for the communication between the virtual machines.

1.2 Description of Processes

1.2.1 Server's Process

When we start the project, the first thread of the server load the database file into a variable and open the port waiting the requests. A fixed thread's pool has been configured with a fixed number of pool set to 3. For each connection, a worker is loaded treating the requests of each client.

1.2.2 Client's process

The client generated request randomly during the process. First, they load a regex file and then generated randomly the length of the request and the type of request(between 0 and 5). The choice of the regex is also randomly done and at the end, they send their request to the server.

1.2.3 Workload

The clients used requests stored into 2 text files. The first file was a list of keywords or part of sentences randomly picked from the dbdata file. The other -most used- one was a list of more general regexs (ex : `[ ][eoiua]*[a-z'_]ing[ ]`).

1.3 Monitoring Setup

To monitor the CPU-Usage during the experiment, we are using dstat. It reports the measurement of CPU-usage, network load into csv file in background. Concerning the requests's delay, the client record the leaving and returning time of the request and store the difference into a csv file.

2 Description of the modeling

The implemented model is the `M/M/1` queue. We have set the arrival rate λ to $1request/500ms$. The number of requests in the queuing station $E[Ns]$ has been set on the server to 5. The queue scheduling is a `First-Come-First-Served`. To test the model , we are using the improved version

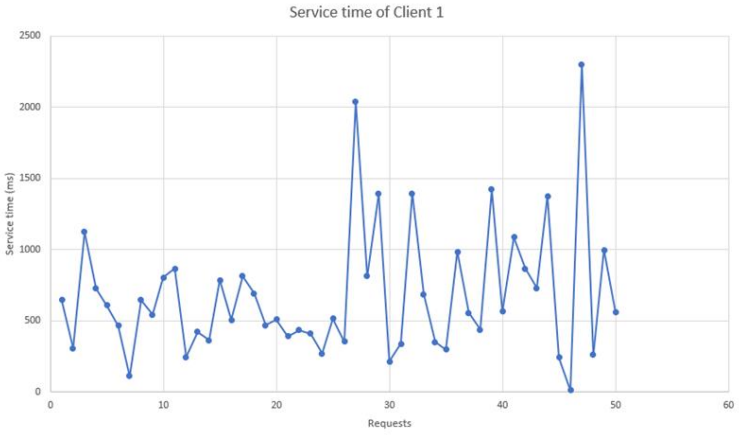
2.1 Theoretical study

Theoretically, that's means that:

- The arrival rate: $\lambda = 2requests/second$
- The average waiting time: $E[W] = 2.5s$
- The average service time: $E[S] = \frac{1}{\lambda} = 0.5s$
- The average response time: $E[R] = E[S] + E[W] = 3s$

2.2 Experimental measurement

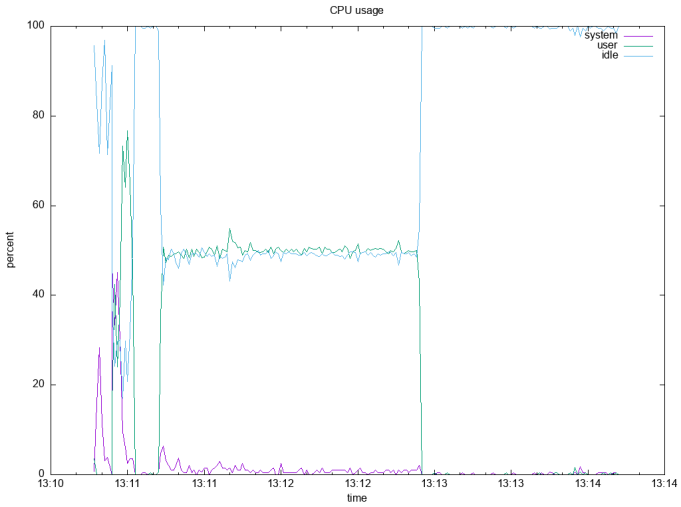
Considering the observed values, the information that we get that the average response time: $E[R] = 742ms$

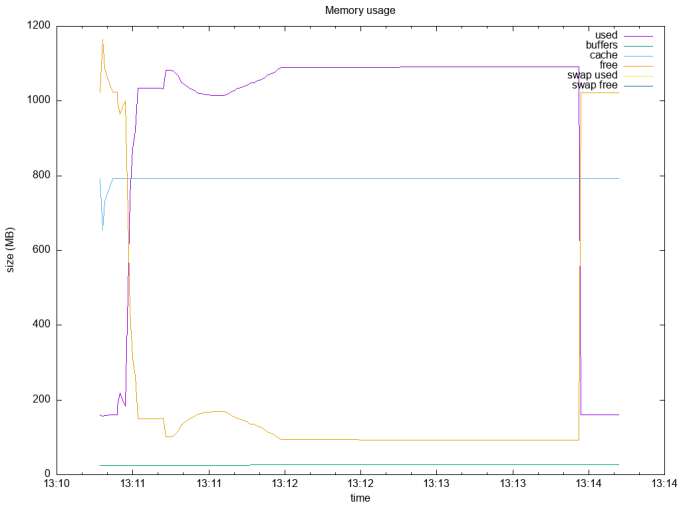


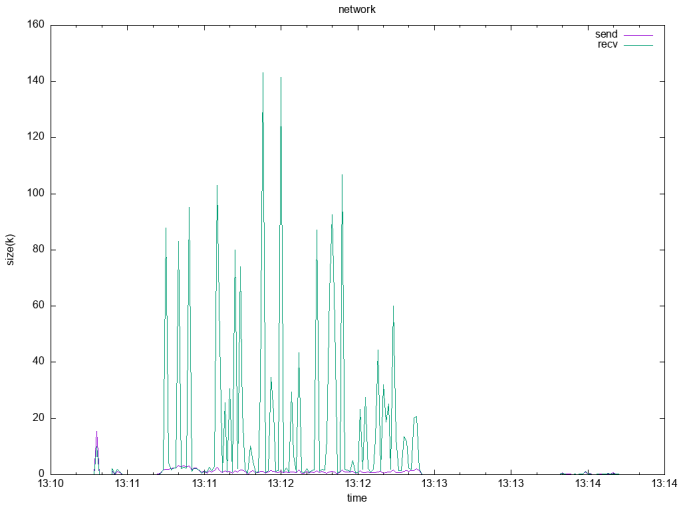
3 Study of the "simple" implementation

3.1 Server

According the the measurement, the CPU load is increasing once we launch the application and the client sending more and more request. We can explain because of the type of objects we are using and the amount the data sending between each machine(depending of how difficult are the requests). Considering the fact that we have fixed the threadpool to 3, we can assume that the server implementation is such that we have 3 services stations. That's why we are considering our server as one using `M/M/3` queue and always based on FCFS scheduling.

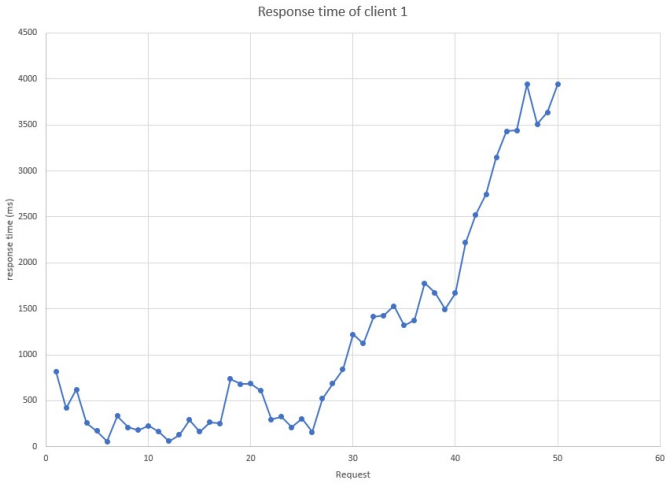


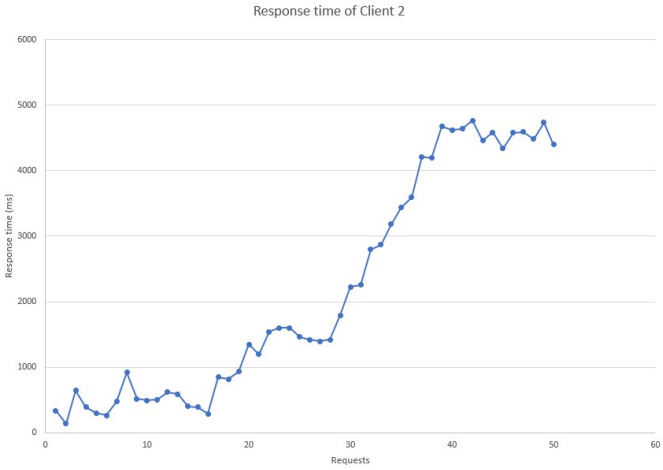


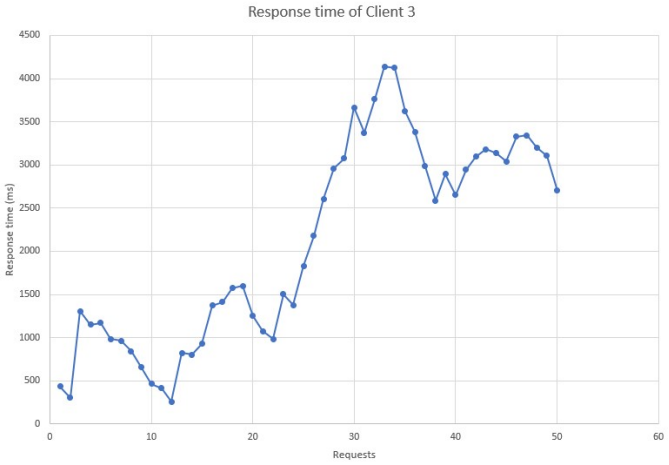


With this version, the network load become greater and greater due to the fact that InputStreamReader and BufferReader to read and send data. Their performance are limited for sending data that are big in term of size.

3.2 Clients



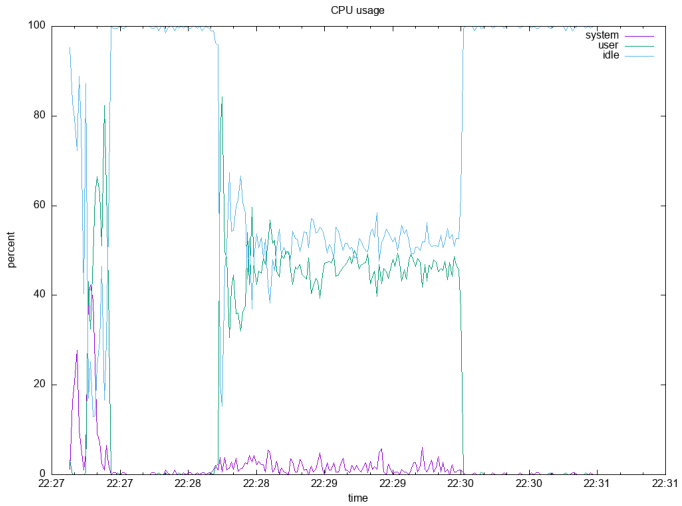


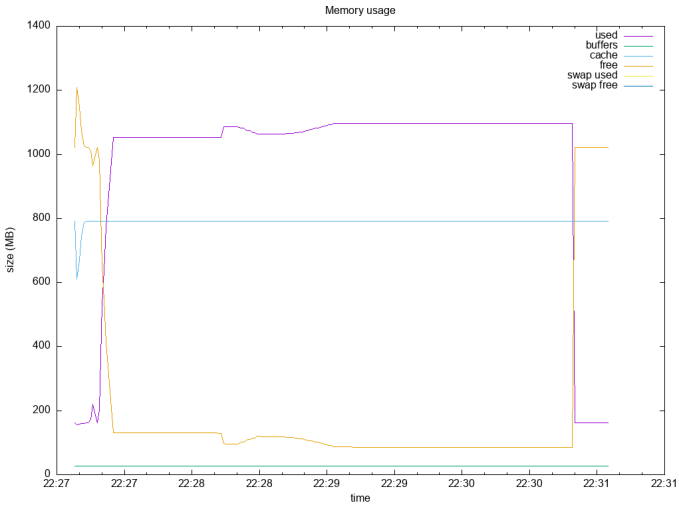


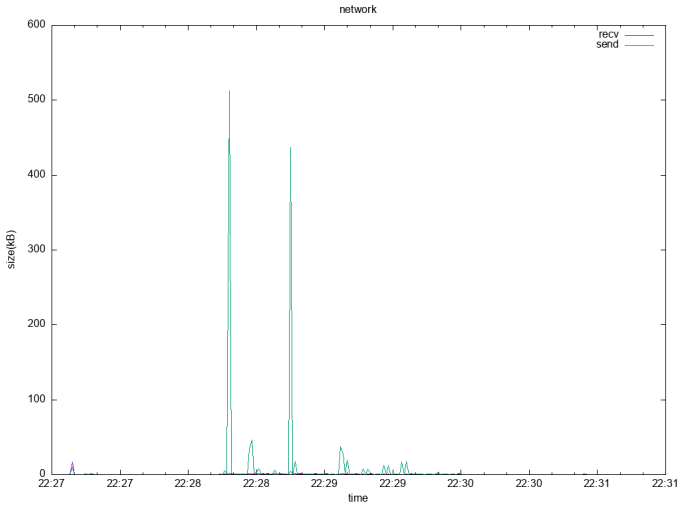
4 Study of the Improved implementation

4.1 Server

The upgraded server retrieved the requested data via an array of ArrayList of String instead of an 2 dimensions String array. And Instead of using BufferReader and InputStreamReader objets to read and send the data through the socket, we are using ObjectOutputStream and ObjectInputStream:

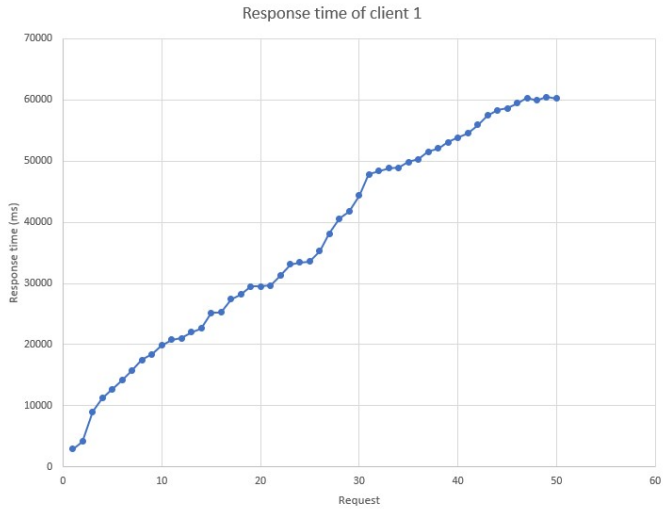


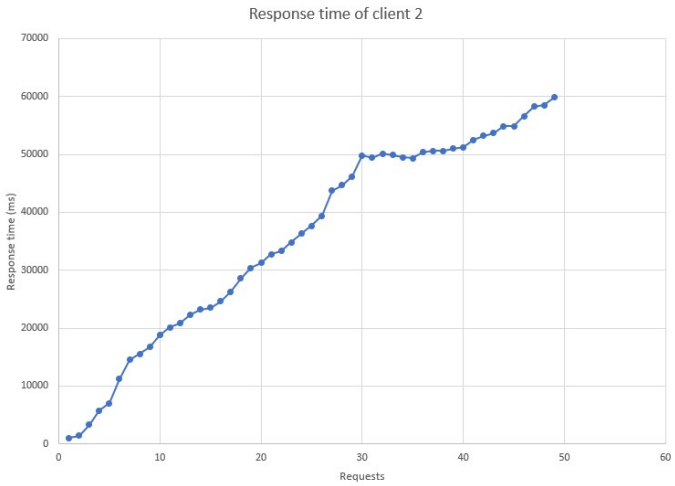


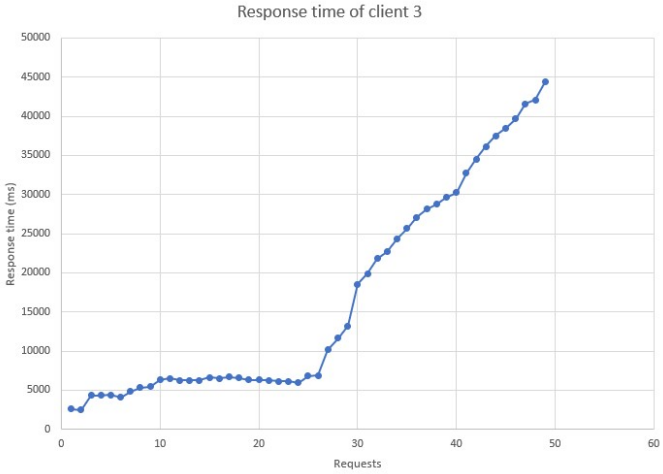


As you can the server manages better the packets band is better than the previous configuration. But there is no much difference between requests the memory and CPU-usage.

4.2 Clients


Response time of client 1
Response time (ms)
Request
0
10000
20000
30000
40000
50000
60000
70000
0
10
20
30
40
50
60


Response time of client 2
Response time (ms)
Requests
0
10000
20000
30000
40000
50000
60000
70000
0
10
20
30
40
50
60


Response time of client 3
Response time (ms)
Requests
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
0
10
20
30
40
50
60