Architecture & performance of computer systems

Rayan Ayouaz (03882000) rayan.ayouaz@student.uclouvain.be

December 2020

1 Implementation

The implementation is made in Java.

1.1 Server

The main thread of the server listen for incoming connections. There is another thread for handling request. Each time a connection is accepted the main thread calls the handler method of the class ClientThread. For clearer measurement of the performance we preferred this approach with only two threads.

To manages socket we used the class Server Socket which allows a maximum queue of 50 incoming connections.

As it can been seen on this piece of code, the handler construction takes on argument multiples array and a Boolean named fast which describe if we want to use the version of the server with improved performance or not. As in the trivial and slowest implementation of the server the sentences are stored in a 2xN array respectively follows the columns for their types and their content, the improved version is separated in 4 different arrays for each types. Since a requested regex will match most of the types one type only, it is more profitable to iterate only on the requested type.

```
1
2         if (fast) {
3             fastArray1 = initFastArray(1) ;
4             fastArray2 = initFastArray(2) ;
5             fastArray3 = initFastArray(3) ;
6             fastArray4 = initFastArray(4) ;
7
8         } else initArray();
```

For some reason that we don't fully understand it was important to initiate the four arrays in separated functions (instead of a function with a case statement) . Otherwise it seemed that the elements was too far away in memory to profit of intern improvement made by the jvm. At least, it is our main hypothesis.

The server can be launch in a terminal with the command line with one argument that can be 0 or 1 respectively for starting the server in slowest mode

and in fastest mode. For instance "java Server 1" will run the server with the improvement of performance. This improvement can be measured for one single request by placing a counter while all the lifetime an object ClientHanlder. By doing so we measure that on our computer used as a server the approximate average was 650ms for one request in fast mode and 710ms for one request in slow mode. As we will see later this small improvement can have a real effect on performance in models and reality.

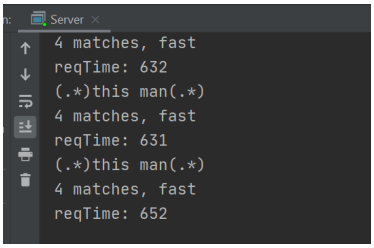

Figure 1: Request times in fast mode

1.2 Client

The client process several request by instanciate a new object RequestHandler, starting a new thread and initialize a new socket for each request that the server will treat as an independent client. We start each RequestHandler following a poisson distribution to better match with real conditions. The

The client can be launch with this series of argument, respectively the IP address of the server, the alpha parameter as Integer that represent seconds and the filename where the requests are stored seperated by newlines. For instance: " java Client 127.0.0.1 11 requests.txt"

where 11 is the alpha parameter (for ten seconds range).

We choose to estimate the alpha parameters into the range of 10 secondes because as our model takes almost one second to process a single request and since the time needed to process one request can't be greater than the arrival rate in queue theory (if we had choose an arrival rate of 2 per second for instance), it was important to choose a bigger and easy to understand range.

2 Server details

See Figure 2

3 Simulate the network

In order to not influence the performance of the server We run the server and the clients on different computers but on the same local network. To simulate dis-

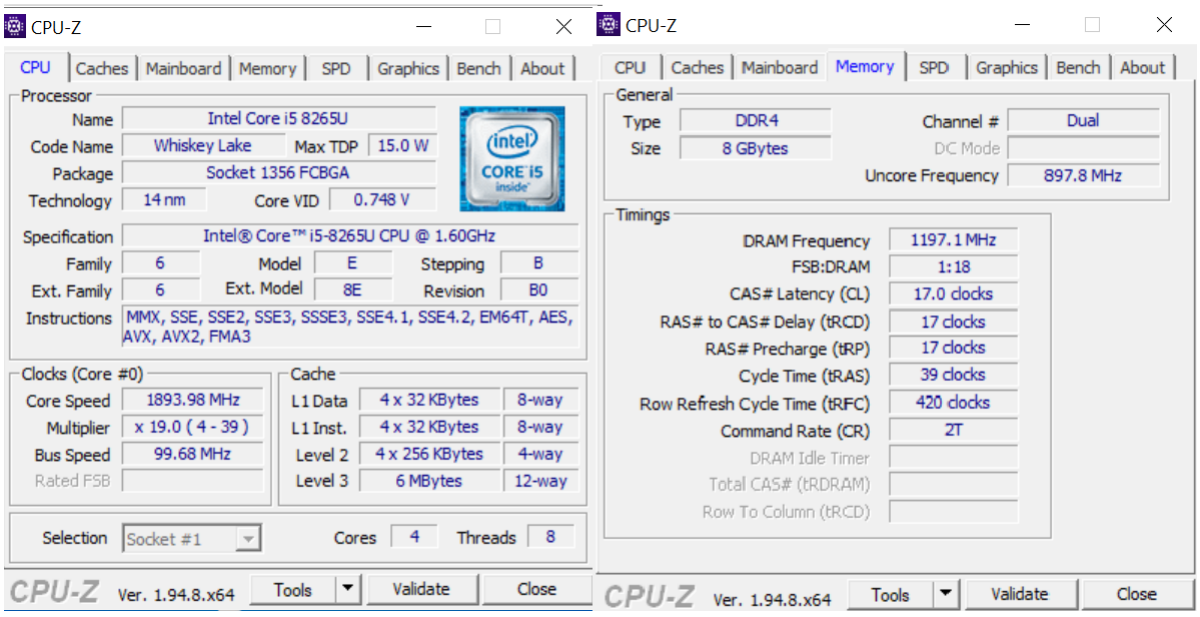

Figure 2: Characteristics of the server

tance we used a software named clumsy for windows. https://github.com/jagt/clumsy

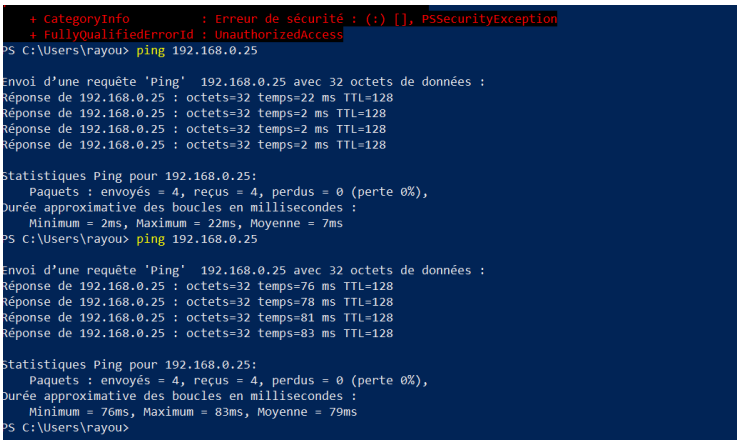

Figure 3: An adhoc delay of 50ms by using clumsy tool

4 Measurement

We measure the performance with three different file request with three different difficulties (from lowest to highest difficulty) two times. The first time the files contained 40 requests, the second time 70 requests since as we know the limit of socket accepting connection is limited by 50. We used so MM1 queue for model the first group and the second group (with respect to Kendall's notation). We also simulate these experience for different lags in network and different arrival rate. The following is a comparison between theses results and the models.

4.1 Models

Poisson range: 10s

4.2 Fast server

$\alpha = 10, 1/u = 0.065$. $E[X] = 1/(15.4 - 10) = 0.19/10s = 1.9s$
For 40 request: 1.9s * 40 = 76s For 70 request: 1.9s * 70 = 133s
$\alpha = 11, 1/u = 0.065$. $E[X] = 1/(15.4 - 11) = 0.22/10s = 2.2s$
For 40 request: 2.22s * 40 = 88.8s For 70 request: 2.22s * 70 = 155s
$\alpha = 12, 1/u = 0.065$. $E[X] = 1/(15.4-12) = 0.29/10s = 2.9s$ For 40 request: 2.9s * 40 = 116s For 70 request: 2.9s * 70 = 203s

4.3 Slow server

$\alpha = 10, 1/u = 0.071$. $E[X] = 1/(14.08 - 10) = 0.24/10s = 2.5s$ For 40 requests: 2.5s * 40 = 100s For 70 request: 2.5s * 70 = 175s
$\alpha = 11, 1/u = 0.071$. $E[X] = 1/(14.08 - 11) = 0.32/10s = 3.2s$ For 40 requests: 3.2s * 40 = 128s For 70 request: 3.2s * 70 = 224s
$\alpha = 12, 1/u = 0.071$. $E[X] = 1/(14.08 - 12) = 0.48/10s = 4.8s$ For 40 requests: 4.8s * 40 = 192s For 70 request: 4.8s * 70 = 336s

4.4 Graphs

Figure 4 & 5

5 Discussion

As we can see on figure 4 and figure 5, increasing the alpha parameters tend to ameliorate the performances, for the fast server at least. This means than our server doesn't saturate and the request time rate may be too slow to observe some interesting features. However we see that the models exaggerate the obstruction of the queue: it can be seen on the slow models where the model value is higher than the observed values. Our estimations of the mean time taken for a request may be too big or it can also be some inside improvement made by the jvm which allows our server to not suffer from the queuing problems.

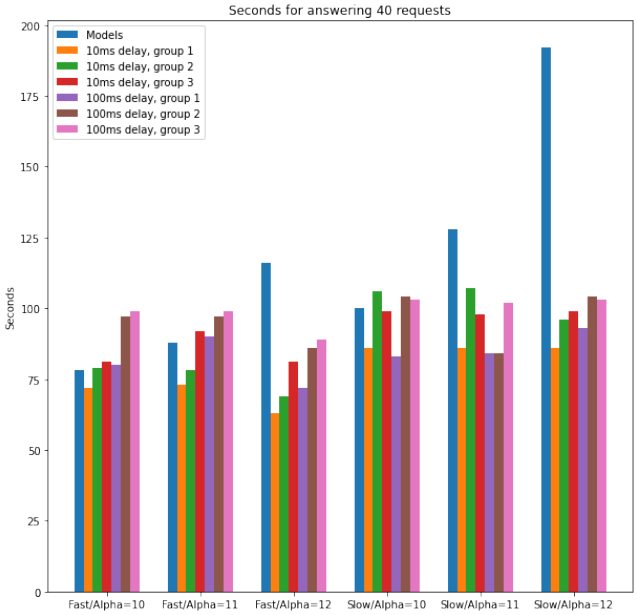

Figure 4: 40 requests

We can observe than slowing the network results obviously to a slowest server response. Finally we can clearly see an improvement between the two version of the server.

Another perspective would be too measure mean response for a request with a better a accuracy and model the "70 requests group" with a MM150 queue model as the buffer for incoming tcp connection is limited to 50 connection.

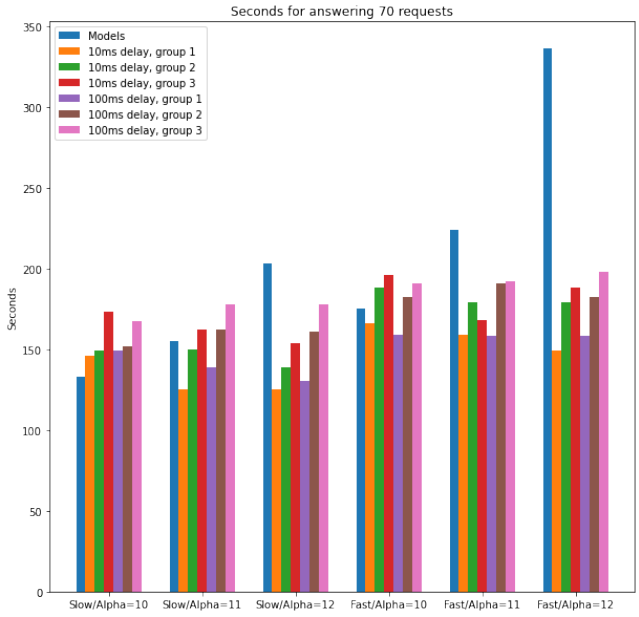

Figure 5: 70 requests