The Probem given

- *You have been provided two files, scenario1.txt and scenario2.txt which each contain simulated application logs in the following format:*

- *Column 1: # seconds since time t=0*
- *Column 2: # requests processed since last log*
- *Column 3: Mean response time for requests processed since last log*

- *The logging frequency is 1 per second, but there are no logs for seconds in which there were no requests. The data span two simulated weeks and were generated under an idealized/simplified model in which there is a single application server which processes requests sequentially using a single thread.*

- *If a request arrives while the server is busy, it waits in a queue until the server is free to process it. There is no limit to the size of the queue.*

- *For each scenario, please answer the following questions.*
- *Note that we define "week 2" to begin at second 626400 (6 am on the 8th day).*

- *1) How much has the mean response time (specifically, the mean of the response times for each individual request) changed from week 1 to week 2?*

- *2) Create a plot illustrating the probability distribution of the amount of server time it takes to process a request (excluding the time the request spends waiting in the queue). How would you describe the distribution?*

- *3) Propose a potential cause for the change in response times. Give both a qualitative answer as if you were explaining it to a client and a quantitative answer as if you were explaining it to a statistician. Create 1 or 2 plots to support and illustrate your argument.*

- *Your submission should include sufficient code and instructions to reproduce your analysis (any quantitative results as well as your plots).*

The Notations and Parameters

We define:

- n is the nth log-line
- t_n is the arrival time of the requests.

The problem is a bit unclear wether tn is the logging time-1s (log frequency) or the last logging time in the file. We take the most realistic scenario (logging-time-1s) where the server updates log every second. In consequence, we directly assume tn= logging time (equivalent problem with time shift of 1s).

- T_n is the inter-arrival time defined as :

$$T_{n+1} = t_{n+1} - t_n$$

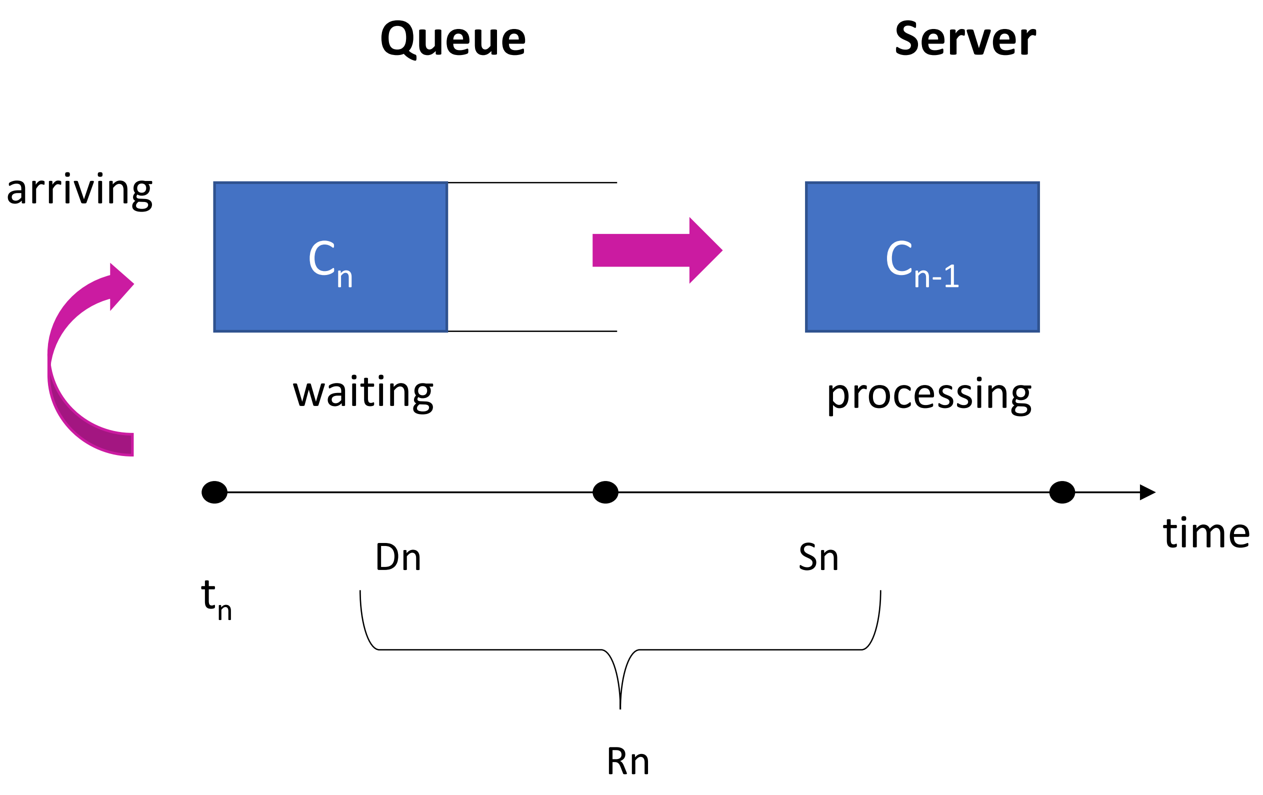
- C_n is the number of requests
- R_n is the total response time of the C_n requests
- D_n is the delay of C_n (i.e the waiting time)
- S_n is the service time (i.e the processing time)

The parameters defined above obey to two logic formulas:

- $R_n = D_n + S_n$
- $D_{n+1} = (D_n + S_n - T_{n+1})_+ = (R_n - T_{n+1})_+$, n>0

Where $(x)_+$ means max(0,x). (if the server is free $D_n=0$)

<u>The steps of a requests package : FIFO type</u>

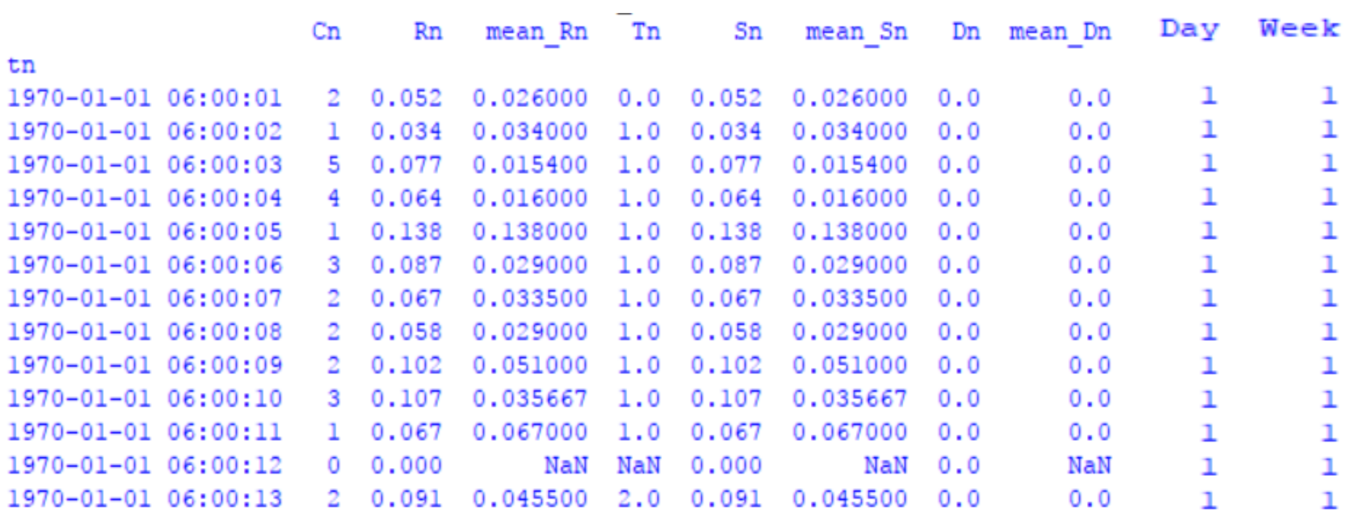


I. Exploratory Data Analysis for the mean reponse time

We first import the two scenarios and calculate the parameters defined above. For convenience, we initialise our time frame from 01/01/1970 (unix-time).

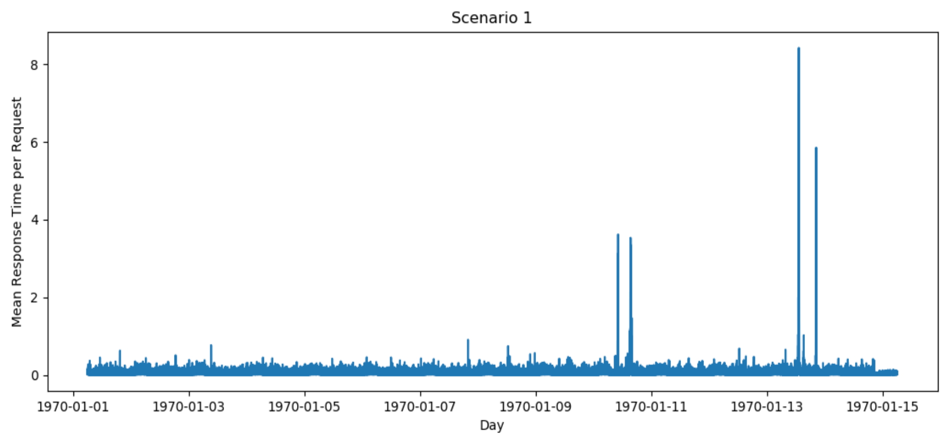
Moreover, In both scenarios, we can isolate about 10 negative data points for the service time Sn, which indicates that the logging frequency is not ideal. We treat them as missing values.

Here the head of the data (Scenario 1) to visualise the type of data available after calculating all the parameters :

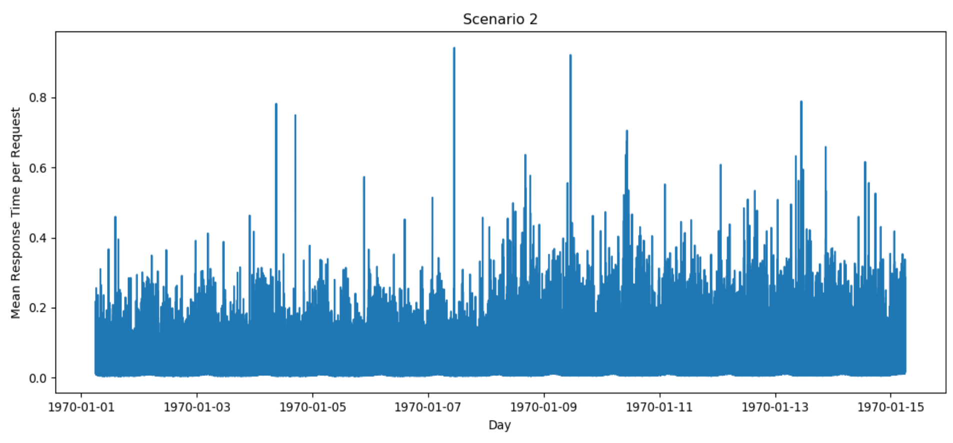


```
                     Cn     Rn   mean_Rn  T̄n     Sn   mean_Sn   Dn  mean_Dn   Day   Week
tn
1970-01-01 06:00:01   2  0.052  0.026000  0.0  0.052  0.026000  0.0      0.0     1      1
1970-01-01 06:00:02   1  0.034  0.034000  1.0  0.034  0.034000  0.0      0.0     1      1
1970-01-01 06:00:03   5  0.077  0.015400  1.0  0.077  0.015400  0.0      0.0     1      1
1970-01-01 06:00:04   4  0.064  0.016000  1.0  0.064  0.016000  0.0      0.0     1      1
1970-01-01 06:00:05   1  0.138  0.138000  1.0  0.138  0.138000  0.0      0.0     1      1
1970-01-01 06:00:06   3  0.087  0.029000  1.0  0.087  0.029000  0.0      0.0     1      1
1970-01-01 06:00:07   2  0.067  0.033500  1.0  0.067  0.033500  0.0      0.0     1      1
1970-01-01 06:00:08   2  0.058  0.029000  1.0  0.058  0.029000  0.0      0.0     1      1
1970-01-01 06:00:09   2  0.102  0.051000  1.0  0.102  0.051000  0.0      0.0     1      1
1970-01-01 06:00:10   3  0.107  0.035667  1.0  0.107  0.035667  0.0      0.0     1      1
1970-01-01 06:00:11   1  0.067  0.067000  1.0  0.067  0.067000  0.0      0.0     1      1
1970-01-01 06:00:12   0  0.000       NaN  NaN  0.000       NaN  0.0      NaN     1      1
1970-01-01 06:00:13   2  0.091  0.045500  2.0  0.091  0.045500  0.0      0.0     1      1
```

Let's first plot the mean response time per request against time to have an overview of our data :

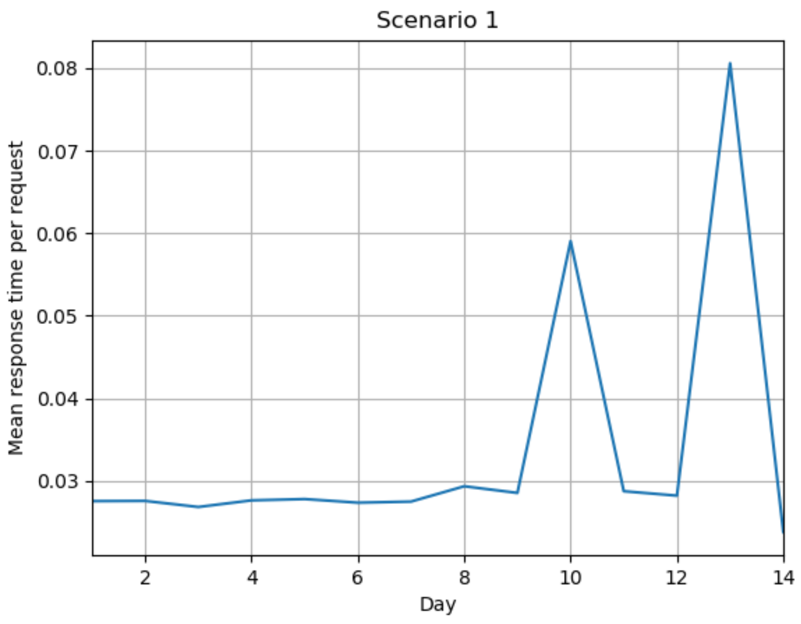


We can see spikes that prove there exist anormal long response times. We will detect these outliers and try to explain their origin.



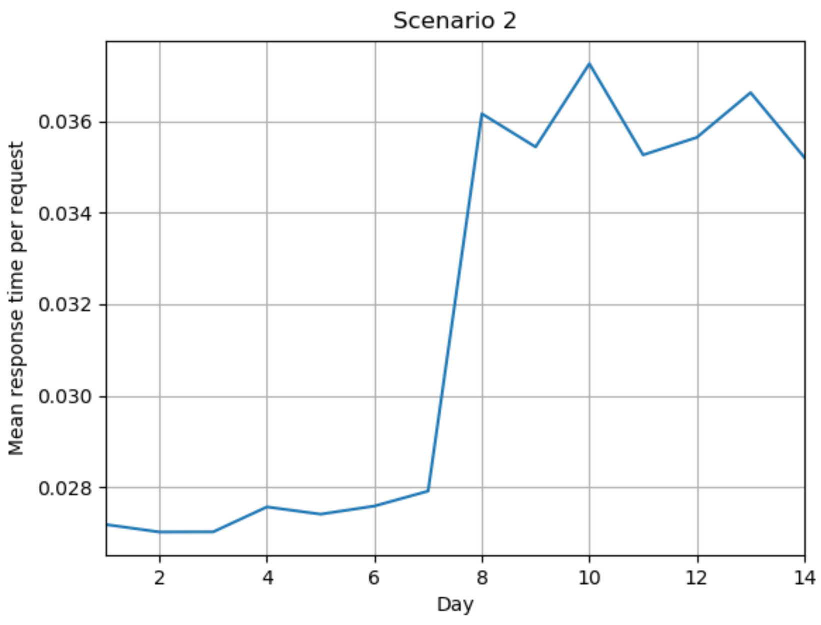
Scenario 2 seems to have a lot of variability but less outliers compared to Scenario 1.

To have a more precise view, we compute the daily mean of the Response Time per Request:



For the Scenario 1, we can notice:

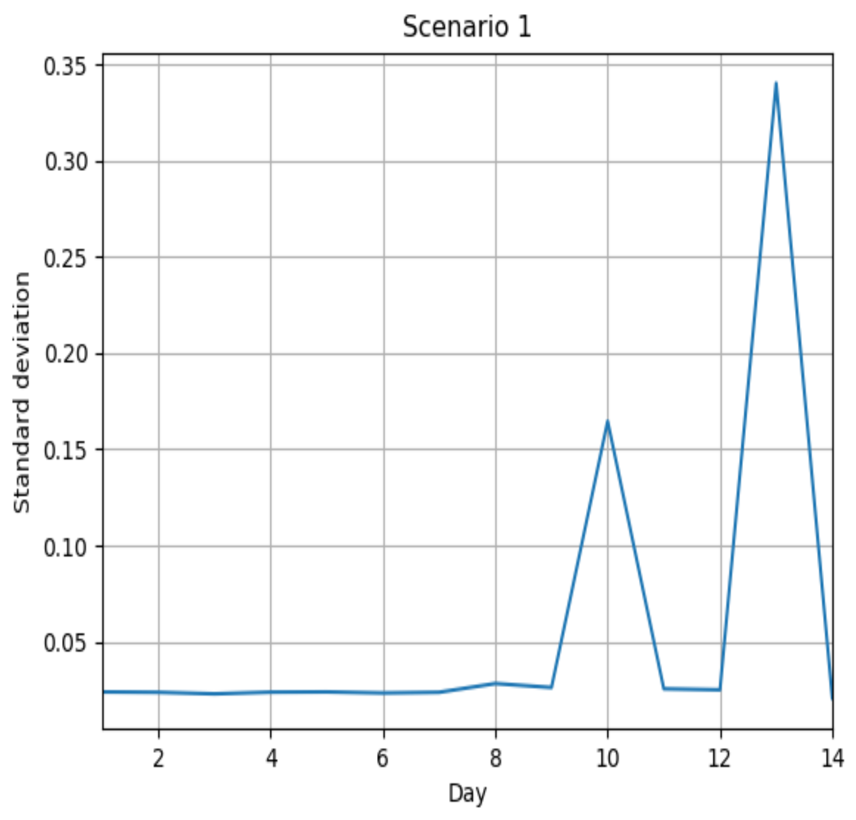
- The mean response time increases from the first week (**27,5 ms**) to the second week (**39,8 ms**).
- The mean of Day 14 is lower than the average (20,9 ms).
- Also, the sparsity is higher in the second week, as day 10 (60 ms) and day 13 (80 ms) have a much bigger mean than the other days.

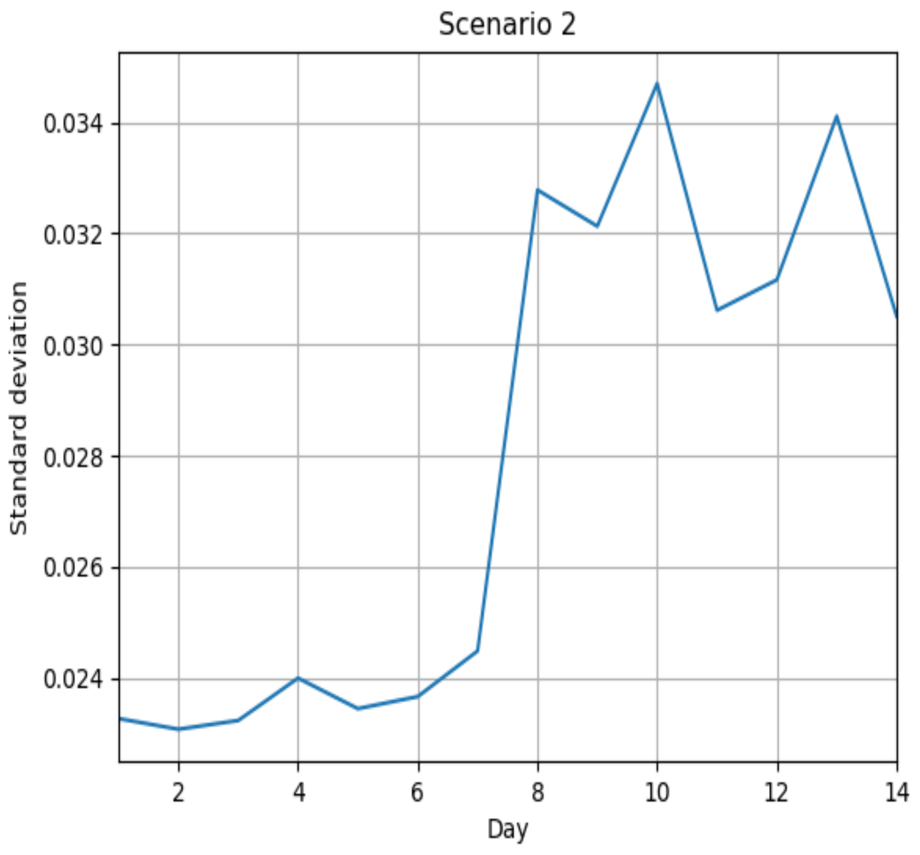


For the second scenario :

- The mean response time increases almost every day during the first week (overall mean = **27,4 ms**)
- The mean level shifts upward during the second Week (**36 ms**) and introduces daily small fluctuations.

We want to measure the variability :





The plot of the standard deviation, which measures how far is the data from its mean, confirms the increase in the second week (from day 8) in both scenarios.

The variability in Senario 1 is high in absolute value, while in Scenario 2,the fluctuations are more visible in comparison to the other days.

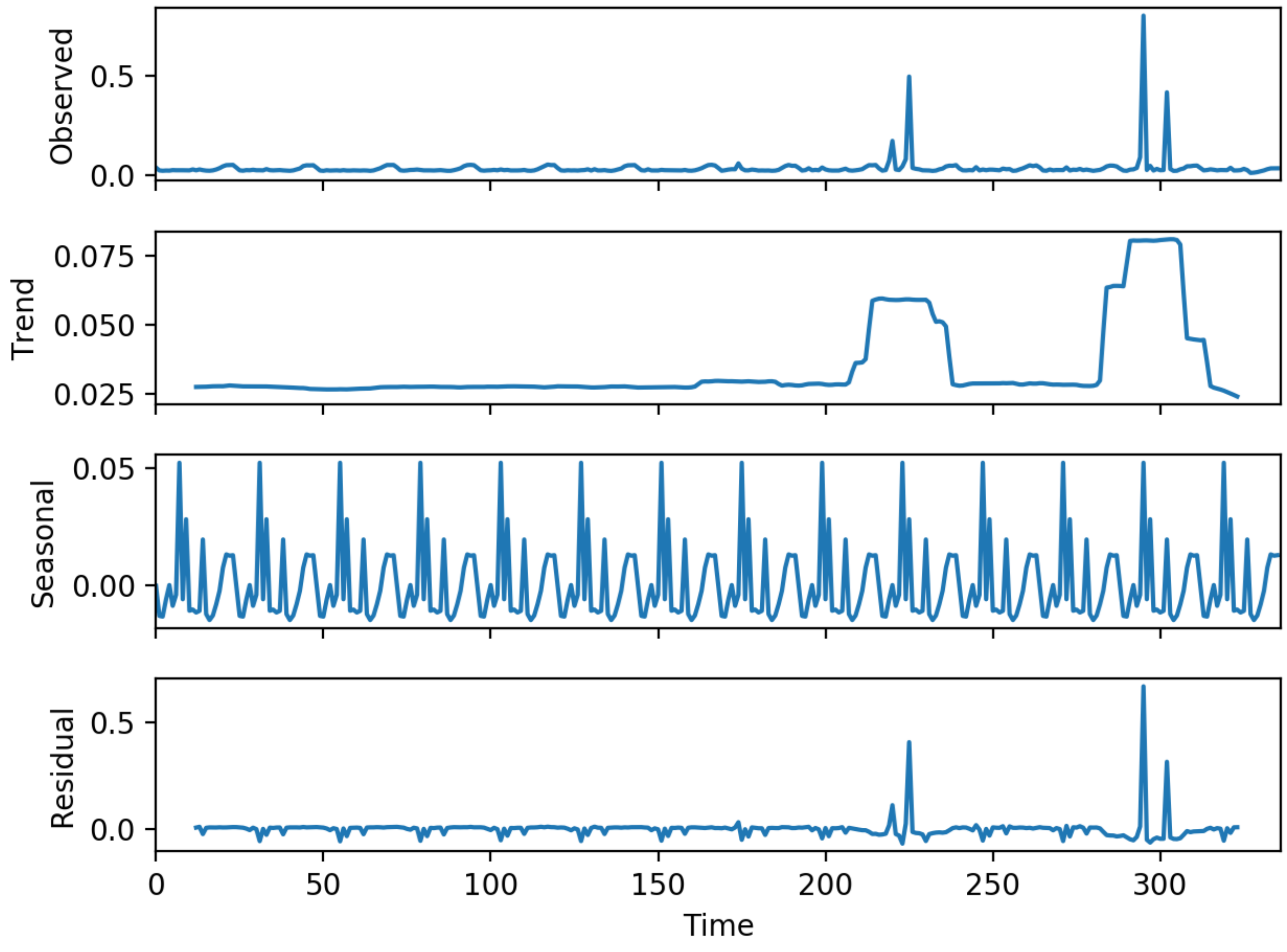
A better understanding of the mean Response time data is to decompose the time series into its trend and seasonal components. We use the following additive model :

$$Mean\_R_n = T_n + S_n + E_n \text{, where } E_n \text{ is white noise}$$

Python assumes a time series is collected at regular intervals, without gaps in the data. So we have to handle missing log times with 0 requests for the analysis.

We fill the missing values with the k-nearest neighbours mean, but other methods could be applied (splines, polynomials…)

<u>Scenario 1 Time Series Decomposition</u>

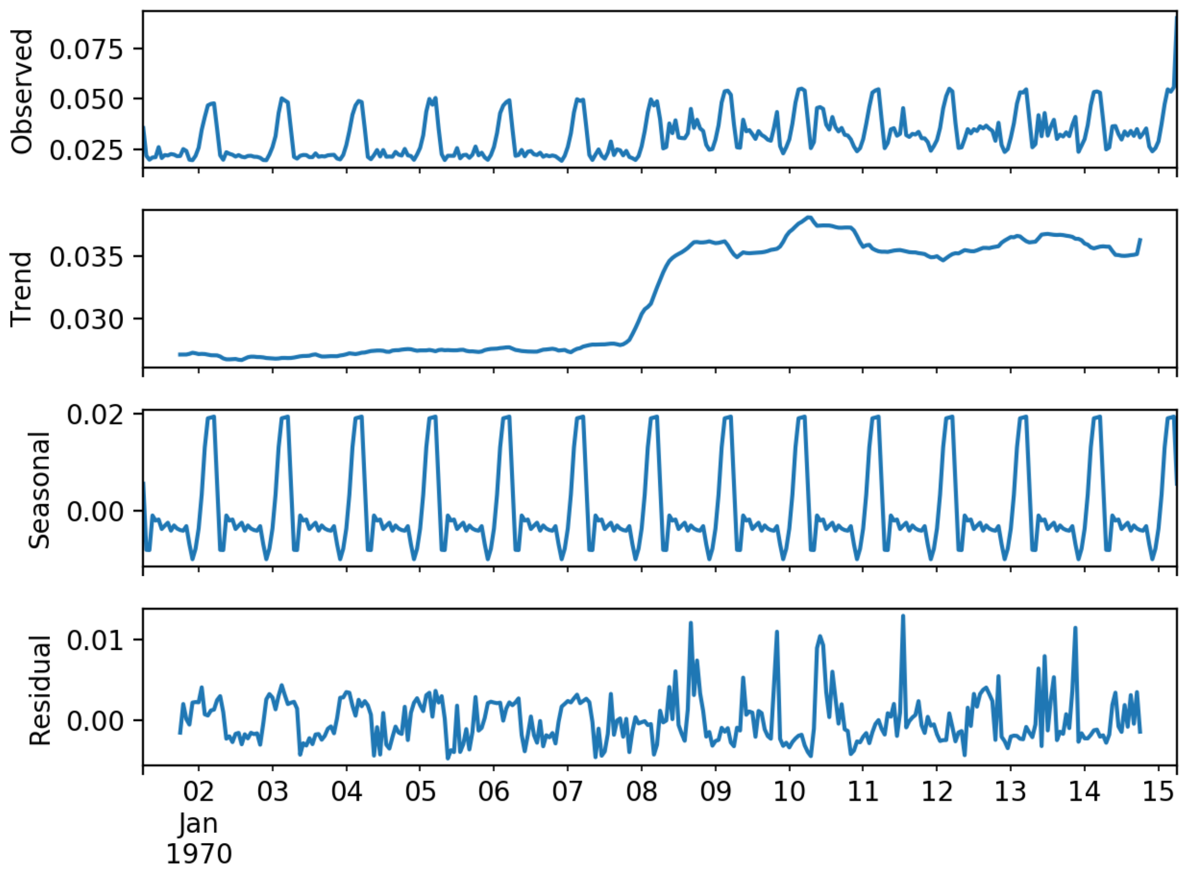


For Scenario 1, the slight upward trend shows the weekly increase of the mean response time (shift at 168 hours = 1 week) and anormal long behaviour for day 10 and day 13.

The seasonal component confirms the daily period of the response time. Spikes are visible during the daytime which could be due to high traffic.

We apply the same decomposition model for the Scenario 2 :

Scenario 2 Time Series Decomposition



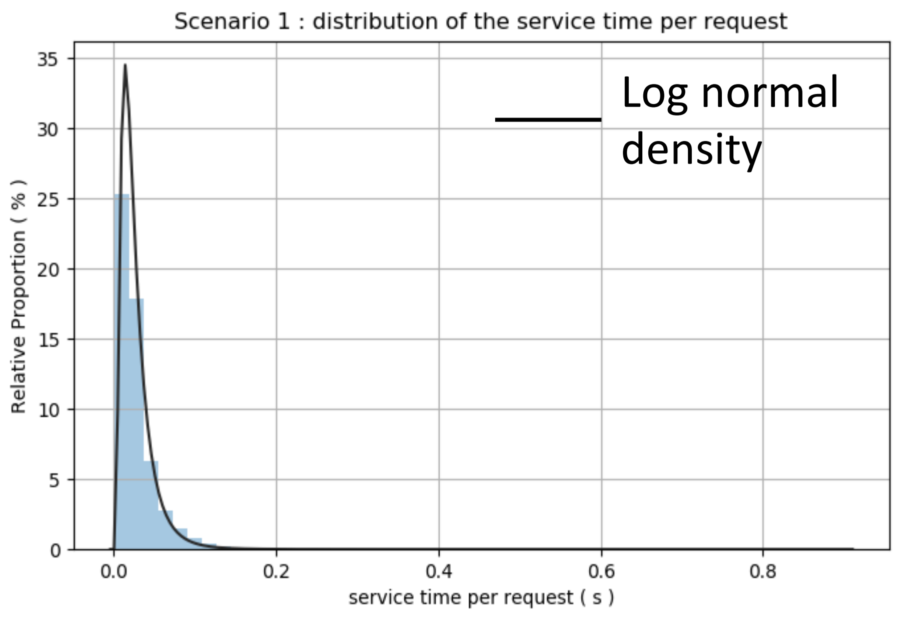
For Scenario 2, we have an increasing trend since the first week.

The mean response time shifts for the second week.

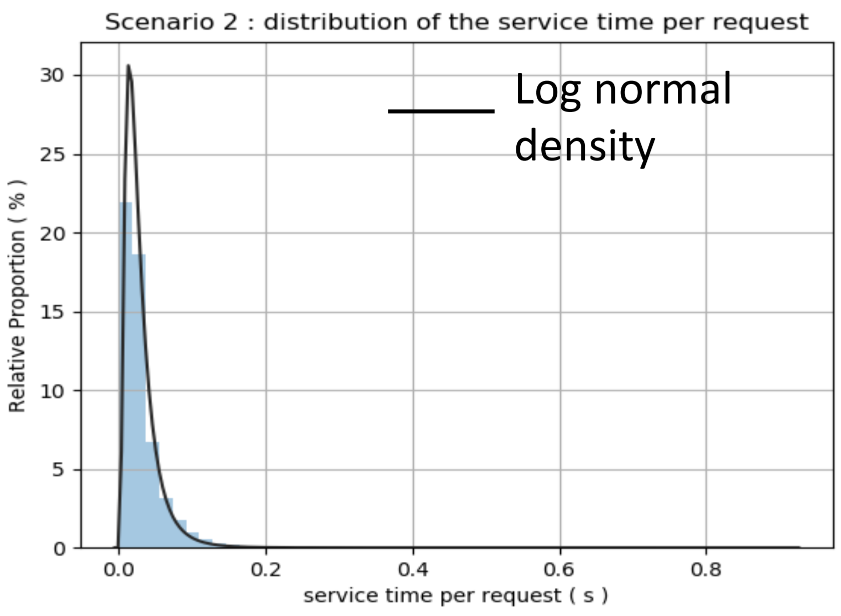
The seasonal component confirms the daily period of the response time. The daily pattern is composed of a spike of long response time and a saller wave of due to high variability (see the standard deviation plot).

II. Probability distribution of the Service Time

Lets plot an histogram representing the service time per request :



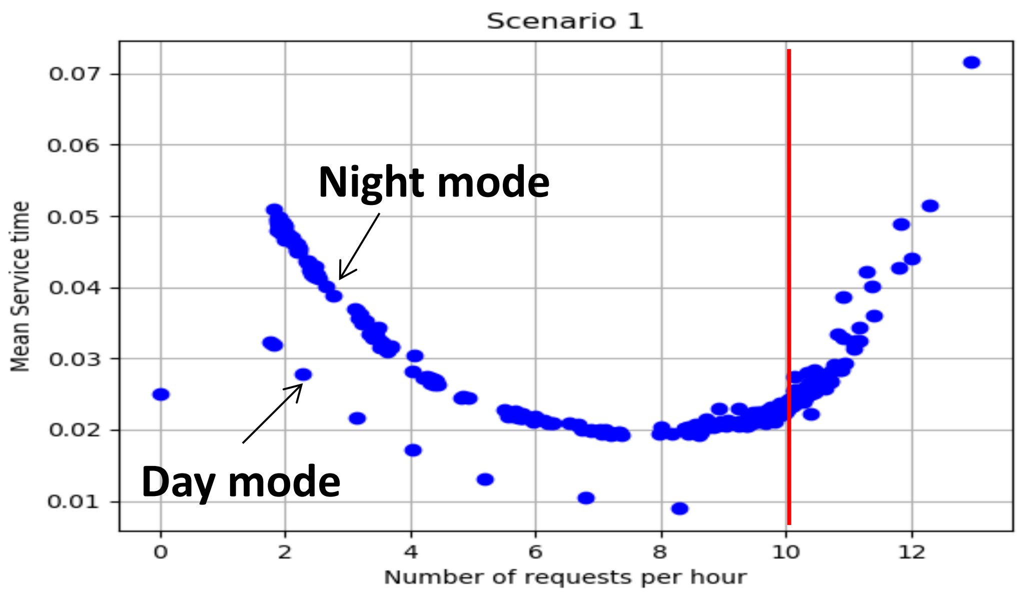
For both scenarios, the best fit is the log-normal distribution (see plots), as the data is assembled around the mean, and the tail component is due to long service time in the data.

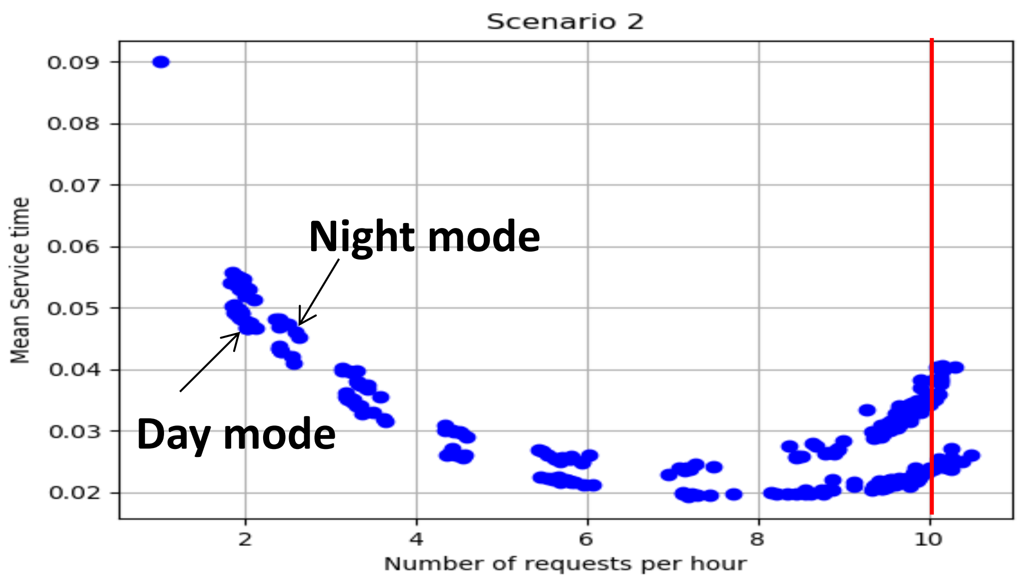


The explanation given to the client is that the goal is to minimise the service time. A single processing server is not enough to process the resquests optimally. Indeed, due to high traffic, above a certain threshold, the number of resquests causes server latency. Tcp error recovery time could explain the long time needed for processing. Lets prove it

III. The causes of the Service Time : Response to a client

The idea is to plot the mean service time against the number of requests :





In both Scenarios :

- We could distinguish two parallel curves, which prove the servers have two modes, that we will call « day mode » and « night mode » :
• On day mode, the server runs at full regime and it process requests optimally.
• On night mode, the server doesn't receive much requests and does not need to be at full regime, so for the same number of requests, it needs more time for processing

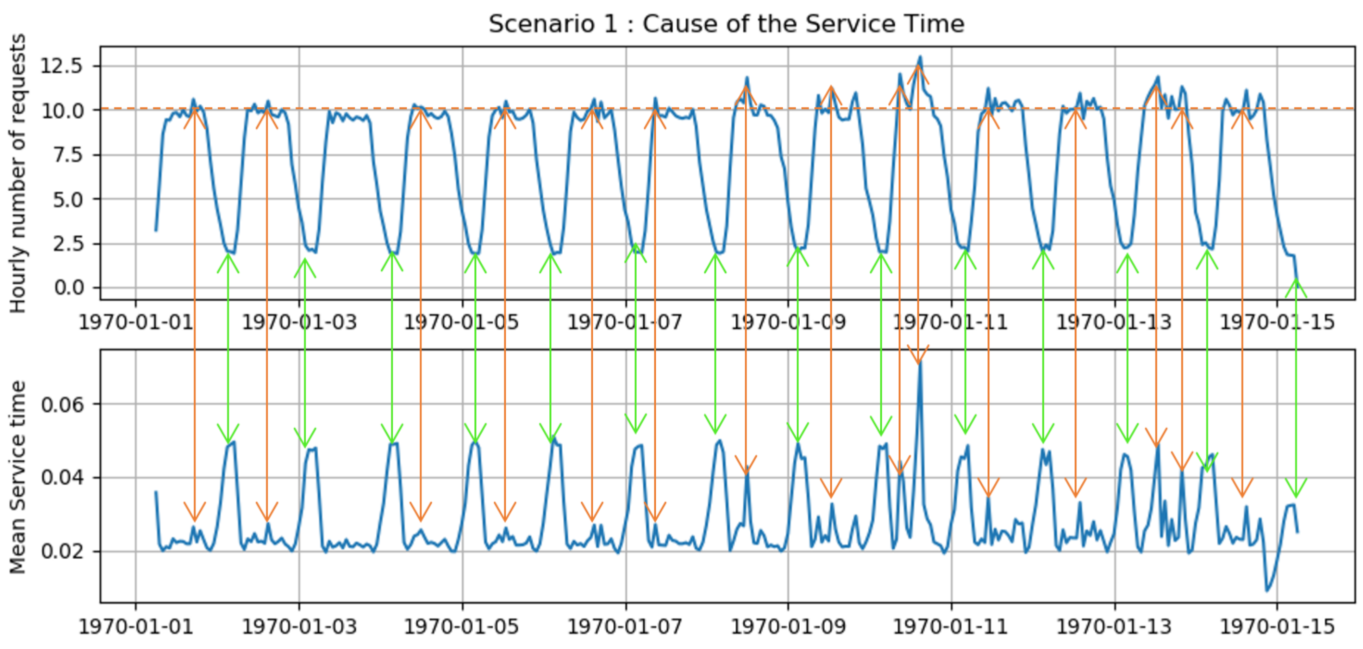
- We clearly see a change in the behaviour of the servers above 10 requests : they have difficulties to process each requests and needs more time. For the second scenario, we do not cross the threshold so often, it confirms that we don't have so many anormal long response times compared to scenario 1

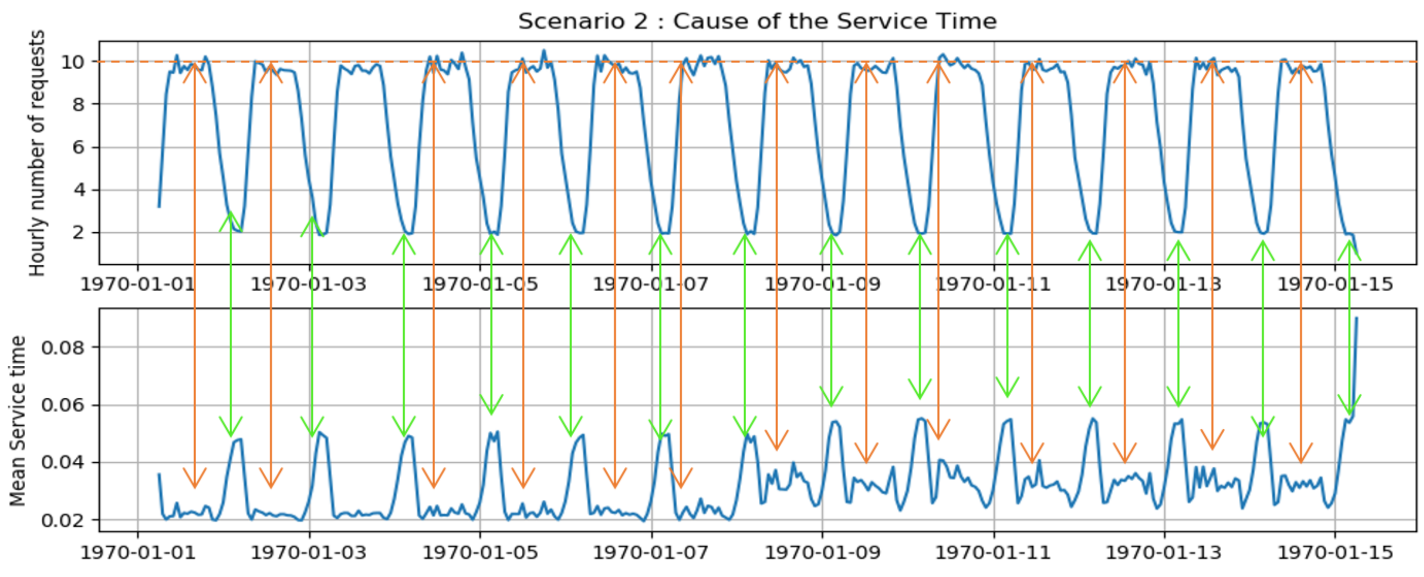
Indeed, recall that :

-maximum response time per request for Scenario 1 : **8,42 seconds**
-maximum response time per request for Scenario 2 : **0,94 seconds**

III. The causes of the Service Time : Response to a statistician

We plot the hourly average of the number of requests per second as well as the mean service time for both scenarios :





These time series plots provide evidence that the variability in service time is due to traffic :

- If the mean number of requests is above the threshold (10), the server processing time per request increases drastically : the performances are worse $\longleftrightarrow$

- The server turns into « night mode » when the number of requests drops (during night). The server does not receive a high number of requests, so it saves resources by diminishing the performance to **50 ms/request**. $\longleftrightarrow$

Conclusion

As a conclusion, the upward shift in mean response time during the second week is explained by the high traffic that increases above a threshold and put the processing in difficulties.

We have proved that, rather than the overall mean of requests, it’s the spikes of high traffic/requests that causes latency.

We can imagine the case of a website which sells clothes and has discounts during the second week (causes high traffic). In the scenario 1, the discounts are on day 10 and 13 (see time series decomposition, the trend component). The discounts were announced at the beginning of the second week. As a consequence, people go to the website to get informed (that explains the overall higher traffic in week 2)

For the Scenario 2, the announcement has been made during the first week because we already have an increasing trend. The discounts are on the whole second week.

We can recommend to the client of the first scenario who faces long processing time issues to use several servers in parallel Or to increase the performances of his server.

Also, based on the available data, we could imagine forecasting methods (ARIMA models) to predict high traffic and allocate more servers for a corresponding period of time.