β as a function of $\rho, N, \alpha,$ and $E[T]$

In order to obtain β as a function of ρ, N, α, and $E[T]$ we started from the function of ρ:

$$\rho = \frac{E[X]}{N \cdot E[T]} = \frac{\beta \cdot \Gamma(1 + \frac{1}{\alpha})}{N \cdot [T_0 + (1 - q) \cdot E[Y]]} \tag{1}$$

.

Seen the equation 1, we can easy compute the function of β as:

$$\beta = \frac{\rho \cdot N \cdot [T_0 + (1 - q) \cdot E[Y]]}{\Gamma(1 + \frac{1}{\alpha})} \tag{2}$$

Mean delay and mean message overhead

As first step, the **inter arrival time** (T) and the **processing time** (X) are generated. Then, comparing the expected mean with the empirical mean of the generated samples it is possible to compute the **number of sample needed** to perform our study.
We decide to use 100.000 samples and repeat our simulation 20 times for each ρ, in this way we can compute the **mean delay/message** which are more reliable values for these metrics.
Since there isn't more than one arrival per unit of time and the time for messages exchanged is equal to zero, the *mean delay* which is equal to *dispatching delay* + *response time*, simply becomes the response time.

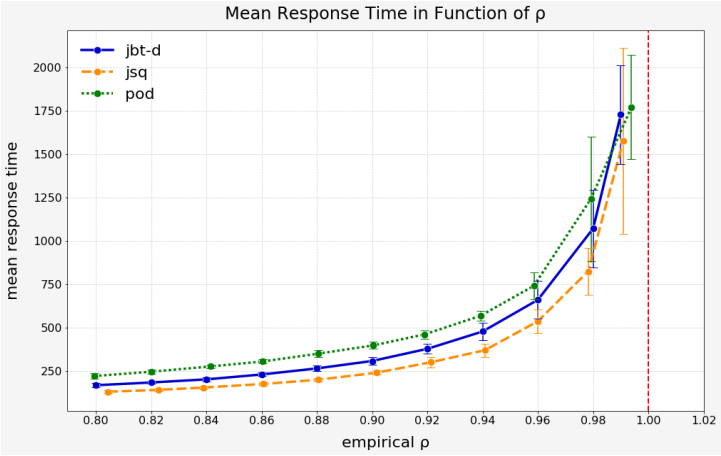


Figure 1: Mean delay (per arrival) for pod, jsq and jbt-d, to varying of ρ

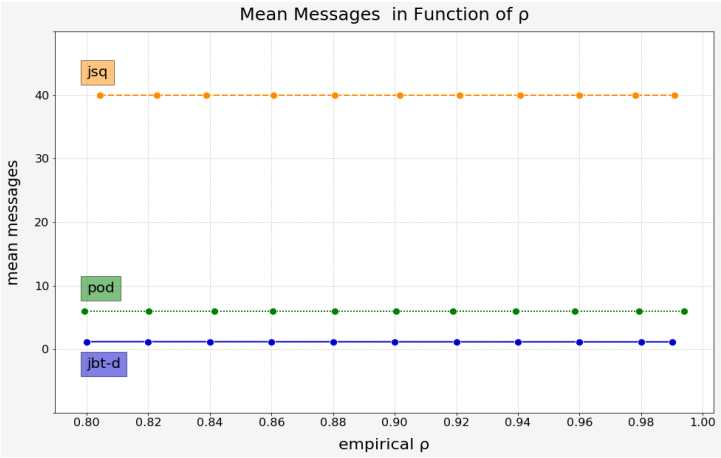
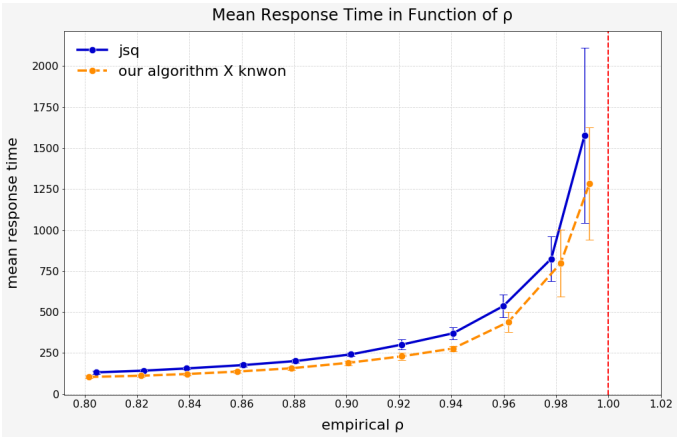


Figure 2: Mean message (per arrival) for pod, jsq and jbt-d, to varying of ρ

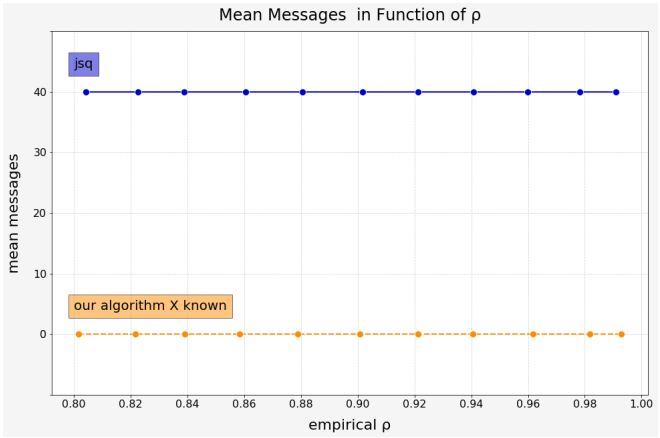
Part 2

The main goal of the new scheduling algorithm is to **minimize the mean delay of the tasks**, since in this framework the time delay into the dispatcher is equal to zero, it means to minimize the mean time of the tasks inside the servers.
Since when a task arrives at the dispatcher, we exactly know the execution time, the best idea is to send this task to the server which has the **minimum usage time**. In this way, this is true for each incoming task, when on arrives at time t it will be sent to the server that processes it as first one.
At this point, it is necessary to develop a method that allows the dispatcher to know which is the right server to send the outgoing task, namely the server that can process it right away. To **avoid messages** among servers and dispatcher, thanks to the fact that we know exactly the processing time of the tasks, it is possible to store a list of N ordered integers in memory, where each integer corresponds to the **usage time** of a server. For each incoming task, the dispatcher checks the list, sends the task to the server with the lowest usage time and add the processing time of that task to corresponding integer. If more than one server has the same usage time the dispatcher chooses randomly. Every unit of time, the dispatcher decreases all non zero values by one, since it knows that all the servers worked for a unit of time.

Below it is possible to appreciate the comparison between *ouralgorithm* and the *jsq*.



(a) Mean delay



(b) mean number of messages per task

It is interesting to see what happens if the dispatcher knows the processing time X but only with relative precision; so for each incoming task we can know its execution time plus a Normal error with $\mu = 0$ and $\sigma = \alpha X$ with $\alpha \in (0, 1)$. As the plot below shows,when α grows, the performances of our algorithm drops.

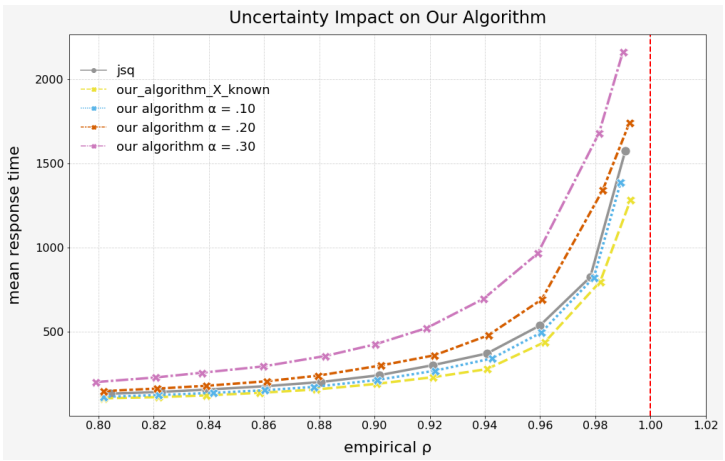


Figure 3: algorithm performance related to the uncertainty $\alpha = (0.10, 0.20, 0.30)$

Code and other matirial related to this challenge can be found at GitHub.