NETWORKING FOR BIG DATA

Challenge 2

Group Rivest

Author:
Bassani Aurora, 1852791
Luciani Erica, 1868647
Mignella Laura, 1920520

1 Data Analysis and Motivations

In order to choose the best dispatching and scheduling algorithms for our data we did a data analysis.
Analysing the data we got to notice that in the dataset there were some jobs -around 11000- with the sum of the total service time equal to zero. Since this kind of data would have lead us to some problems when computing the job slowdown S, we decided to drop them. Before starting our analysis we also scaled the data by taking the minimum arrival time to zero and subtracting this value from all the others.
Now, if we take a look at the curve in **Fig.1**, representing the ecdf of tasks per job, we can immediately notice that almost the entirety of the jobs - approximately 94% - has only a single task. So this first information made us think that the majority of the response time per job actually depends only on the performance of a single task.

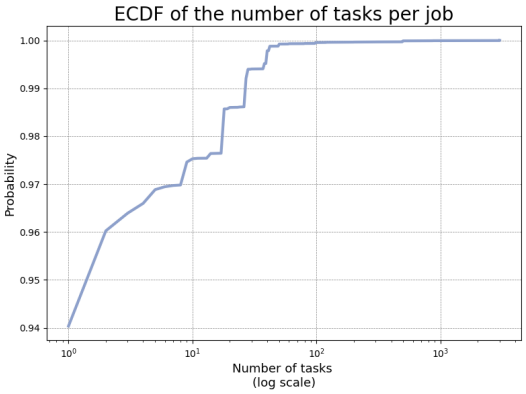
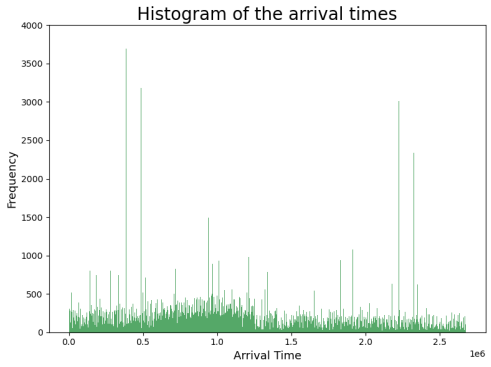
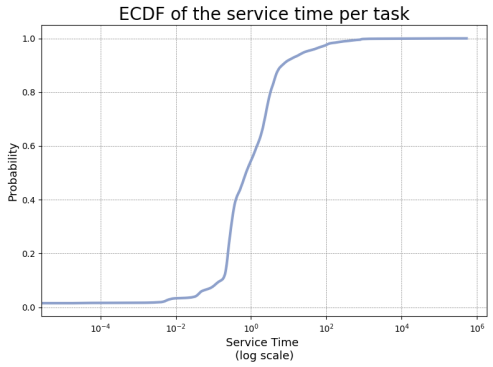


Figure 1

Then we tried to better observe the arrival time of each task, to get an idea of the frequency of incoming tasks in a given time interval. For this purpose we plotted a histogram, shown in **Fig.2(a)**, from which it appears that there are some time intervals when the number of tasks that arrive in the dispatcher is way over the average. In particular, there are few intervals in which this number almost reaches 4000. This plot suggested that we needed to implement a dispatching algorithm that takes into account the queues of the servers and a scheduling algorithm that executes the tasks assigned to a server with a certain priority.



(a)



(b)

Figure 2

Since most of the jobs has a single task, the last thing that we decided to analyse was the service time of each task. Hence we plotted the ecdf of the service time, that can be seen in **Fig.2(b)**. What we can conclude by looking at it is that about the 50% of the tasks has a service time under 1 second and most of the remaining tasks have a service time between 1 and 100. Although most of the tasks seem to be short, given that there might be some queues, the plot points out that a suitable priority criterion could be executing first the shortest tasks in time. The goal of this approach is to not to worsen the response times of shorter tasks due to the delay caused by the execution of a small amount of longer tasks.

2 Formal description of the algorithm

Dispatching Algorithm

As mentioned in the previous section, in the dispatching algorithm we would like to take into consideration the tasks already assigned to each server. Our first choice would have been Least-Work-Left (LWL), because it assigns tasks to the server with less work left and so it seemed to us the most reasonable approach. Nevertheless, since we already had to use it in the baseline algorithm, we opted for a different dispatcher with similar behaviour. Looking for some other algorithms, we came across Join-Below-Threshold (JBT) and tried to pull together some useful ideas from both the algorithms. In particular we were interested in keeping the anticipative property of the LWL and at the same time the behaviour according to which we assign a task to a server that is under a threshold.

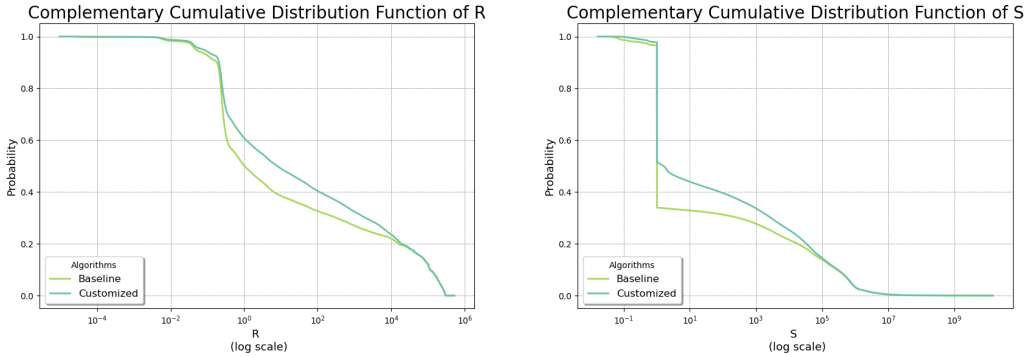
More in details, our customised algorithm works in the following way: first thing first, we initialise all the servers available to be chosen for task execution and set a threshold t equal to the mean of the service times ($t = 39s$). Each time a new task arrives we randomly choose the server between the list of the available ones and, if that server -once assigned the task- has a waiting time above the threshold, we remove it from the list of available servers. We iterate this process until there are no more servers under the threshold. The next step is to compute a new threshold, that we actually compute every time the list empties out, by taking the minimum waiting time between thirty servers -or less, if there aren't enough- randomly extracted between all those currently busy. Once we have the new threshold, we re-fill the list of available servers. If there are no busy servers, we consider all of them available and the threshold is brought back to $39s$. Using this algorithm we do not expect a great improvement in the values of R and S, but for sure there will be in the mean $\bar{L}$ of the number of messages exchanged for each task.
To compute L we use the following method: when a task arrives to the dispatcher, if we have to re-fill the available list, we find L as $64 * 2 + 2$, where $64 * 2$ is the number of back and forth messages of all the servers with the dispatcher to let it know about their waiting time, while the $+2$ part is the number of messages exchanged when the dispatcher assigned the tasks to the server. When a task arrives and the dispatcher still has available servers, L is simply equal to 2.

Scheduling Algorithm

As anticipated in the data analysis, we thought of using a scheduling algorithm with a priority mechanism based on the service time. A natural choice seemed the Shortest-Job-First (SJF), that is an anticipative but non-preemptive algorithm.

The structure of the algorithm is the following: once a task i is assigned to a server j by the dispatcher, we remove from the list of the tasks assigned to j all the ones with a final time smaller than the arrival time of i. Then, if j does not have any further task, we simply append i to the tasks list of j -and so it will be immediately executed- and compute both the final time of i and the time at which the server j finishes its work. When i arrives, if the server is already busy executing a task, but has no other tasks queuing up, we repeat a similar process as before, adding the task i to the queue. Otherwise, if it is still busy executing a task and also has other tasks queuing up, then we insert the task i in a position of the queue such that all the other tasks after will have a greater service time and all the other before will have a service time less or equal.

3 Performance evaluation



(a) (b)

Figure 3

As we can see from the summarising table below, the mean response time, the mean job slowdown and the mean utilization coefficient of the customised algorithm do not show significant improvements with respect to the baseline algorithm, but at least they do not show a significant worsening neither. In particular from the Complementary Cumulative Distribution Functions in **Fig.3** the custom curve is always slightly above the baseline one and so it is always slightly worse. Nevertheless, the number of messages exchanged for each task is far lower in the custom version, since in the LWL the dispatcher exchanges two messages with each server for each task.

	$\bar{R}$	$\bar{S}$	L	$\bar{\rho}$
Baseline	27601.158	1241639.327	130	0.539
Custom	27973.390	1257139.427	6.82	0.539

Another unchanged result is the mean of the utilisation coefficient. We expected this result, because, using a non totally randomized dispatcher, it is reasonable that, on average, the servers work the same amount of time. This is particularly highlighted in **Fig.4**, where we reported the boxplots of the baseline on the left and of the custom on the right. We can notice that the interquartile range is narrower in the custom version and in general it has a lower variety. The utilisation coefficients assume values in a small range and most of them are concentrated around the mean 0.539. We can also see the presence of a few outliers in both cases and this means that some servers -but actually a small percentage of them- have utilisation coefficients with values quite far (in proportion) from the mean $\bar{\rho}$.

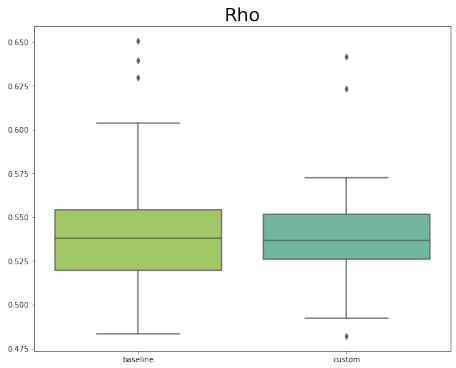


Figure 4