Department of Data Science

NBD - DC - Challenge 2

Networking for Big Data and lab.

Group Name: Postel

Professors:

Baiocchi Andrea

Cianfrani Antonio

Students:

Alvetreti Federico	1846936
Barba Paolo	1885324
Corrias Ioan	2079420

Academic Year 2022/2023

Formal Description of the algorithm

Explanatory Data Analysis

The following plots highlight the characteristics of the distribution that we identify as key factors influencing the performance metrics of interest.

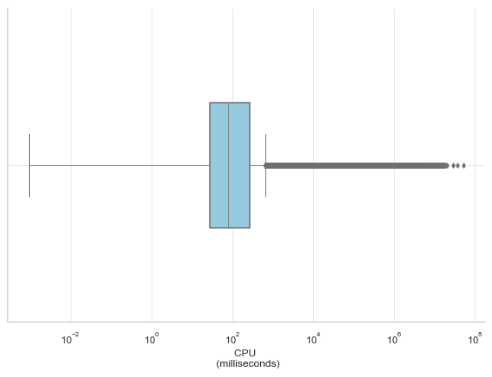

Figure 1:
Box Plot CPU Distribution

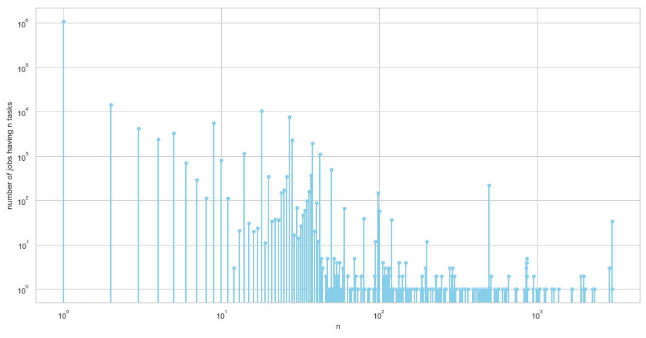

Figure 2:
Number of jobs having n tasks

From the presented plots it is shown that the dataset is composed primarily by small tasks with relatively low service times and jobs tend to have a low number of tasks. This suggests that prioritizing tasks with shorter execution time could represent an effective strategy. Additionally we can conclude that implementing a preemptive priority scheduler for trying to complete smaller jobs first is suitable in our scenario.

Baseline Algorithm

Dispacthing

The dispatching algorithm employed is the Least Work Left (**LWL**). The workload status of each server is estimated by calculating the return time of the last task in the server's queue, which denotes the time at which the server will become available. The iterative process involves assigning tasks to the server with the lowest workload status.

Scheduling

For task scheduling, we have implemented the **FCFS** (First-Come-First-Serve) algorithm. Upon the arrival of a task to a server, we evaluate its delay by subtracting the arrival time of that task from the return time of the previous one. In the case when the queue is empty, the delay is considered as zero. The computed delay enables us to compute the return time of the task.

$$return\ time = arrival\ time + service\ time + delay$$

Custom Algorithm

Dispacthing

The aim of our dispatching algorithm is to reduce the number of messages between the dispatcher and the servers.

To achieve so, at each dispatcher iteration, we reserve the n servers with workload status than the arrival time bound computed as:

$$arrival\ time + \alpha + initial\ window$$

where:

- α initially set as 1 is doubled until we don't find a not empty set of servers.
- the *initial window* hyper-parameter is set to 10

We will then assign each of the following tasks to one of the servers in the set.

Observation Changing the dispaching algorithm will not have an impact on the mean utilization coefficient. In fact this value will always approximately be equal to

$$\frac{\text{sum of all tasks execution time}}{\text{Data Center utilization time}} \approx 0.53$$

The dispatching algorithm will only affect the distribution of these coefficients. The optimal one should follow a uniform.

Scheduling

The aim of our scheduling algorithm is to prevent heavier tasks from slowing down smaller ones, this leads to prioritize the completion of jobs composed by easy tasks and increasing the overall performances.

To reach so, we have implement **HRRN** (Highest response ratio next). This algorithm aims to find a balance between jobs that have been waiting for a long time and jobs with shorter expected service times. We computed the response ratio to prioritize jobs defined as the following:

$$response\ ratio = 1 + \frac{wainting\ time\ of\ process}{estimated\ run\ time}$$

As the waiting time of a task increases, its response ratio will be greater, making it more likelihood to be executed.

Each time a task is being scheduled we first check, for each of the current task in the server's queue, if the arrival time of the new one is greater than their return times removing them from the queue. Then we update the waiting time of the remaining tasks as the difference between the return time of the last discarded task and their arrival time. Furthermore, we sort out in descending order the queue by evaluating each task's response ratio. Eventually we update the workload status of the server by summing the new task's arrival time and the service times of each task in the queue.

Performance metrics and plots

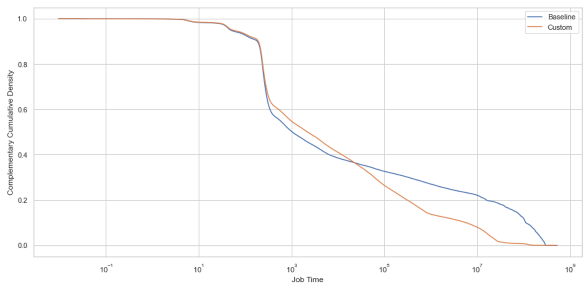

Figure 3:
Job response time R

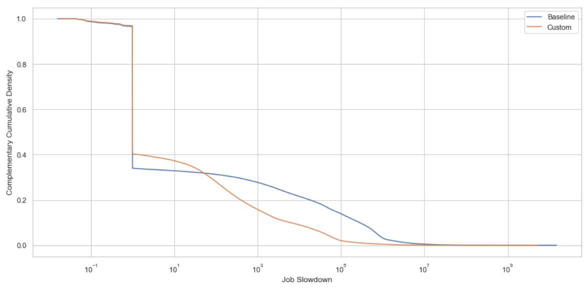

Figure 4:
Job slowdown S

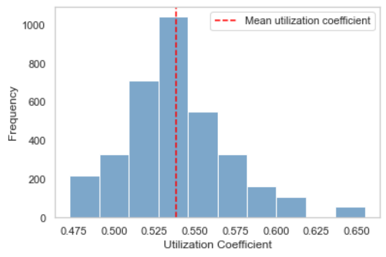

Figure 5:
Baseline utilization coefficient histogram

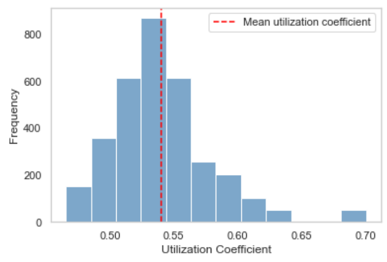

Figure 6:
Custom utilization coefficient histogram

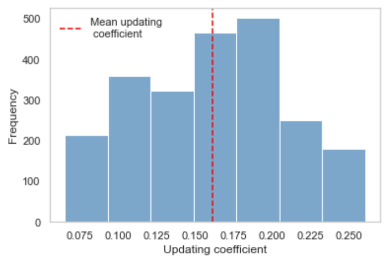

Figure 7:
Custom updating coefficient histogram

Figure 8:
Metrics estimators

	Baseline	Custom
$\overline{R}$	27601.16 s	2744.7 s
$\overline{S}$	1241639.3	166151.1
$\overline{\rho}$	0.5385	0.5399
$\overline{L}$	129	19

As shown in Figure 3 we notice that the custom algorithm, despite an unhappy behaviour in the middle, for higher job response times we always get a lower percentage of jobs with respect to the baseline. This has a direct impact also for the job slowdown eECDF (Figure 4).
As highlighted in the previous observation the average utilization coefficient in both algorithms sticks around 0.53. As shown in Figure 6 the distribution shape of our custom algorithm is far off from our desiderata uniform distribution.

We can conclude with Figure 8 that the custom algorithm performs better with respect to all the metrics of interest (Mean Job response time, Mean Job slowdown, Mean Utilization coefficient, Mean Messaging load). As highlighted in figure 7 our scheduling algorithm updates the order of the tasks on average 16% of the time.