NETWORKING FOR BIG DATA AND LABORATORY

Data Center, Challenge 2

Group Nickname: Wiener

Aur Marina Iuliana, 1809715
Balestrucci Sophia, 1713638
Musacchio Michele, 2070948

A.Y. 2022-2023

1 Dataset Analysis

First, **to ensure more consistent data**, we decided to remove rows from our dataset where the CPU usage is equal to 0. Additionally, we converted the values in the Arrival Time column from microseconds to seconds.

By exploring the dataset, we noticed that:

- the tasks take approximately **31 days to complete**;
- **94.8% of jobs consist of only one task** (as illustrated in Figure 1);
- there is a **significant variability in the service time**: the variance of the service time is 1879441.34, with a mean of 40.43 and a standard deviation of 1370.9271822502308;
- **74.5% of jobs have a service time lower than 1** (as illustrated in Figure 2).

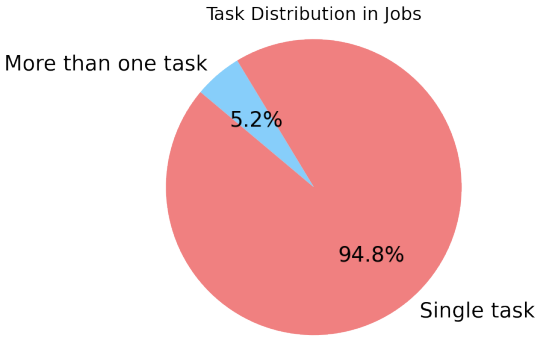

Figure 1: Plot of task distribution in jobs

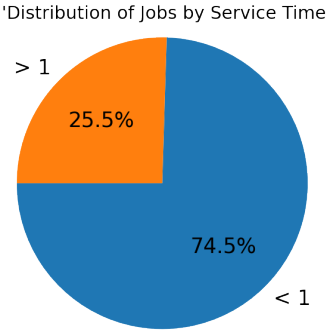

Figure 2: Plot of distribution of jobs by service time

2 Formal description and motivations of our algorithms

2.1 Dispatching algorithm: LWL (Least Work Left)

After examining the dataset, it became evident to us that there is a **significant variability in service times**. Consequently, dispatching policies such as RR (Round Robin), RANDOM, and JSQ (Join Shortest Queue) proved to be unsuitable choices (as also confirmed by several fast tests).

Therefore, we decided to use the **LWL (Least Work Left) dispatching policy**, which has been proven in literature to effectively maintain good stability even in the presence of significant variability in task service times.

The purpose of our work was to **minimize the Mean Job Response Time**. Since the majority of jobs in our system consists of only one task, we found that the LWL dispatching policy remained the most suitable choice for our dataset. Indeed, the LWL policy effectively allocates resources by **prioritizing processes or jobs with less work remaining**. As a consequence, this approach reduces the waiting time for each job and it also leads to a decrease in the Mean Job Response Time.

2.2 Scheduling algorithm: SJN (Shortest Job Next)

In conjunction with this dispatching policy, we decided to use the **SJN (Shortest Job Next) scheduling policy**. The SJN scheduling policy is recognized as being more efficient than FCFS (First Come First Serve) - the baseline scheduling policy - in minimizing the Mean Job Response Time. It achieves this by **prioritizing tasks that require less service time**.

The SJN scheduling policy carries the potential risk of causing jobs with longer service times to experience **starvation**. However, since the majority of jobs in this system comprise only one task with relatively short service time, the impact of these outliers can be minimized.

2.3 Implementation

From the code point of view, the process involves the following steps:

1. First, we **initialize several variables** (including empty lists for each server, a list of unfinished work for each server, a dictionary to store task completion times, a dictionary to store service times for each job, and variables to track message loads, utilization and observation time) because we must keep track of quantities needed to evaluate the required metrics;

2. For each task, we check if there are any available servers by evaluating if the *"available servers"* list is not empty;

   (a) If there are available servers, the algorithm selects the first available server by assigning the index of the first element in the *"available servers"* list to the variable *"server id"*;

   (b) **LWL dispatching algorithm**: otherwise, the algorithm finds the server with the least unfinished work. In order to achieve this, we have created a list, unfinished work, which keeps track of the amount of remaining work for each server. By locating the server ID associated with the minimum value in the unfinished work list, we can identify the server with the least amount of unfinished work;

3. **SJN scheduling algorithm**: after the task is assigned to a server, we use a **priority queue (heap queue)** to order the tasks waiting to use the server based on the shortest service time. The queue is constantly sorted and we utilize the *pop() method* to directly extract the task with the shortest service time;

4. We compute **completion time** for each task using the following formula:

$$U_n = \max(0, U_{n-1} - T_n) + X_n, \quad n \geq 1$$

   where T_n is the inter-arrival time between arrival n - 1 and n;

5. At each step, we update the unfinished work for each server and we save the service time and task completion time in the appropriate dictionaries.

6. Finally, we return the task completion times, service times, utilization coefficients and mean message load, which can be used for performance evaluation.

3 Performance Evaluation

To compare the performance of our algorithm and present summary statistics, we computed the mean values of all the provided metrics (Job Response Time "R", Job Slowdown "S", Utilization coefficient of server "ρ", Messaging Load "L") and the empirical complementary cumulative distribution function (eCCDF) of R and S. More details on how the metrics were calculated can be found in the attached Python file. The obtained results are shown below:

	$E[R]$	$E[S]$	$E[L]$	$E[\rho]$
Baseline	27603.90 seconds	1241675.57 seconds	65 messages	0.538 seconds
Our algorithm	1050.7499 seconds	1050.75 seconds	65 messages	0.008 seconds

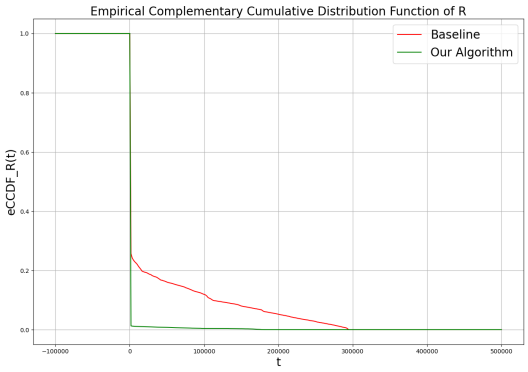

Figure 3: Plot of eCCDF of R

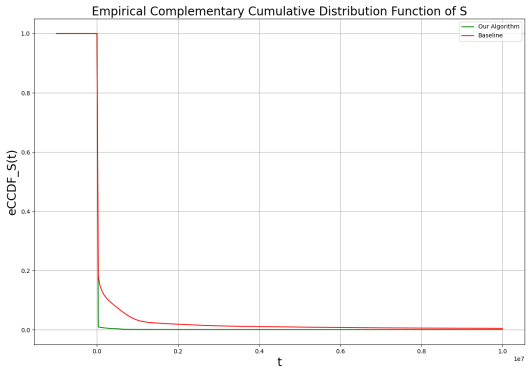

Figure 4: Plot of eCCDF of S

Upon observing the eCCDF of R and S of our algorithm *(green curves)*, we noticed a **steeper decrease**. This suggests that a smaller proportion of jobs experience high response times or slowdowns, highlighting **a better performance of our implementation** with respect the baseline.

3.1 Analysis of results and takeaway

In conclusion, Figure 6 shows impressive improvement in our algorithm *(orange bars)* compared to the baseline *(blue bars)* in almost all the metrics (messaging load remains unchanged as the dispatching algorithm was not modified). Specifically, the improvements we observe in R and S indicate that we are achieving **faster and more efficient job completions** in our algorithm, while the improvement in ρ reflects a **better utilization of server resources** that are distributed more evenly, reducing the load and congestion on individual servers (Figure 5).

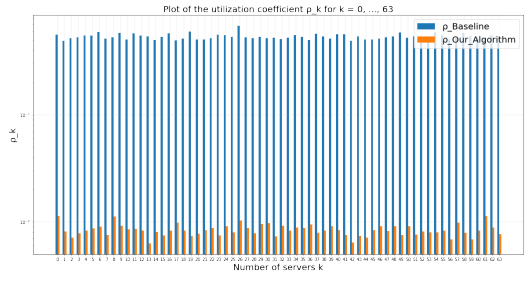

Figure 5: Utilization coefficients for each server

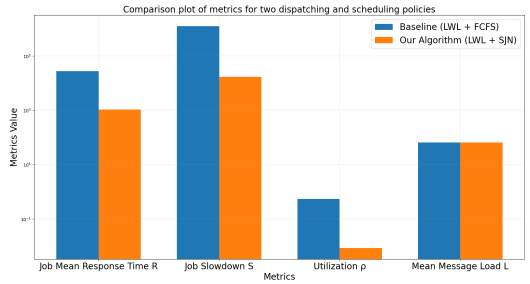

Figure 6: Algorithms comparison