SAPIENZA
UNIVERSITÀ DI ROMA

Networking for Big Data and Laboratory Challenge 2

Group name: OHM

Candi	Protani	Tarantino
Matteo	Andrea	Ramona
1884760	1860126	2082006

Dispatcher and Scheduling algorithms

To develop the code we decide to use two types of objects: tasks and servers.
The class *Task* has as attributes: job id, task id, arrival time (*ta*), service time (*X*), ending time, the number of total tasks of the job to which it belongs (*n_tasks*).
The class *Server* has as attributes: server id, current work assigned to it (*workload*), timer that represents the last time the dispatcher interacted with it, utilization.
We reserve a number of servers to run the tasks that belong to jobs with a total number of tasks lower than 10 (type A servers) and the others that run tasks that belong to jobs with a value greater than this (type B servers).

Algorithm 1 Dispatcher

Given a task that belongs to a job splitted into k different tasks, the algorithm decides to which type of servers to assign the task (type A or type B) and, among these, selects the server with the minimum workload.

Inputs
task ← object of Task's class
servers ← list of objects of Server's class
max_tasks ← int that represents the maximum number of tasks of a job to decide if assigns the new task to a server of type A or type B
threshold ← int that represents the number on servers of type A

```
1:  function Dispatcher(task, servers, max_tasks, threshold)
2:      if task.n_tasks < max_tasks + 1 then
3:          server_group = servers[ : threshold]
4:      else
5:          server_group = servers[threshold : ]
6:      workload = list()
7:      for server in server_group do
8:          workload.append(servers.workload)
9:      idx = argmin(workload)
10:     selected_server = server_group[idx]
11:     return selected_server
```

We implemented this approach because our primary objective in the homework was to identify the best possible Mean Job Response Time. Based on our exploratory analysis of the data, we noticed that a significant portion of the jobs consists of only a few tasks. To solve this problem, we propose to allocate a set of

dedicated servers (type A) specifically for these jobs. We searched for different combinations of *threshold* and *max_tasks* defining an algorithm that, given a *max_tasks*, arrives at the optimal value of *threshold* in the least number of steps possible (it tries 11 instead of 63 combinations) and we reported the results in Figure 1: the "optimal" value we found are *max_tasks*=10 and *threshold*=58. By reserving those servers, we ensure that the tasks of the jobs with few of them are not delayed or impacted by other larger jobs in the system. This allocation strategy helps to prioritize and expedite the execution of these jobs, leading to shorter response times.

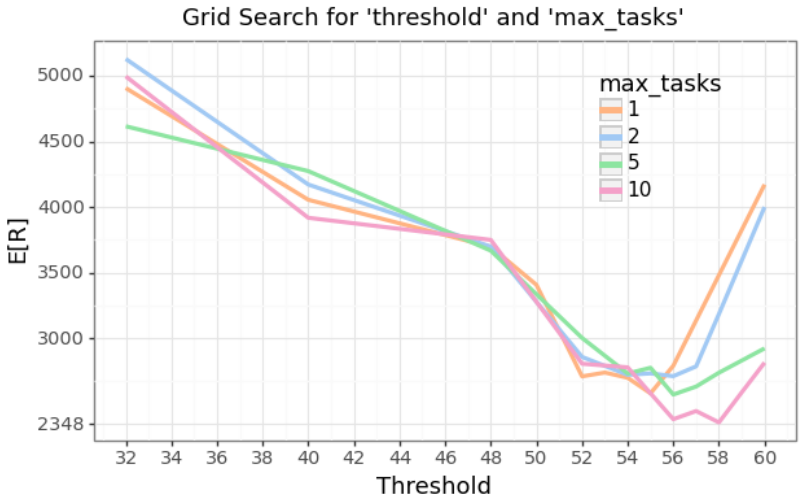

Figure 1: $\mathbb{E}[R]$ based on the threshold and the maximum number of tasks of a job

Algorithm 2 Scheduler

For each server, we implemented the **SJF** (Shortest Job First) algorithm. Every time the server takes, as the next task to run, the one in its queue with the minimum value of the service time (X).

Input

queue ← list of objects of Task's class

```
1: function Scheduler(queue)
2:     service_times = list()
3:     for task in queue do
4:         service_times.append(task.X)
5:     idx = argmin(service_times)
6:     next_task = queue[idx]
7:     return next_task
```

The use of this algorithm for the scheduler decreases even more the average value of the response time. In fact, taking from the queue the task with the minimum service time (X), we reduce the overall response time (R) of the jobs with few and quick tasks which are the majority in our dataset.

Performance & metric evaluation

The average value of ρ significantly decreases because the servers dedicated for tasks belonging to jobs with many tasks are utilized almost the entire time, making all others available for computing the other tasks but most of the time they remain unused as illustrated in Figure 2. In this way we reach the aim to decrease the *Mean Job Response Time* ($\mathbb{E}[R]$) and the *Mean Job Slowdown* ($\mathbb{E}[S]$) as reported in Table 1.

Metric	Baseline	Our
Job response time (R)	27601.16	2348.83
Job slowdown (S)	1241639.33	207371.67
Utilization coefficient (ρ)	0.538	0.183
Messaging load (L)	129	38.9

Table 1: Comparison of Average Metrics between Baseline and Our Approach

In the baseline, the *Average Message Load* ($\mathbb{E}[L]$) is 129 due to two messages exchanged for workload and one for task assignment per server. However, our algorithm minimizes the message load by only requesting workload only from a selected group of servers. This reduces the number of messages exchanged.
In addition, as we can see from Figure 3, with our algorithm we get to the point that the probability of having a higher value of R and S for a job decreases more quickly compared with the baseline.

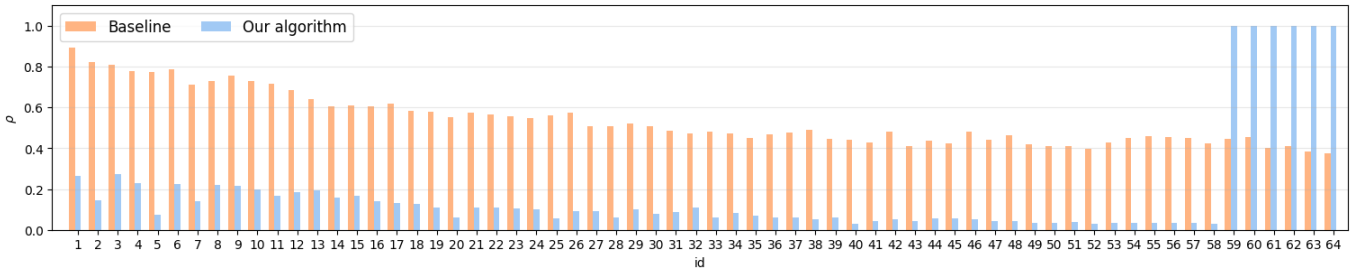

Figure 2: Servers Utilization

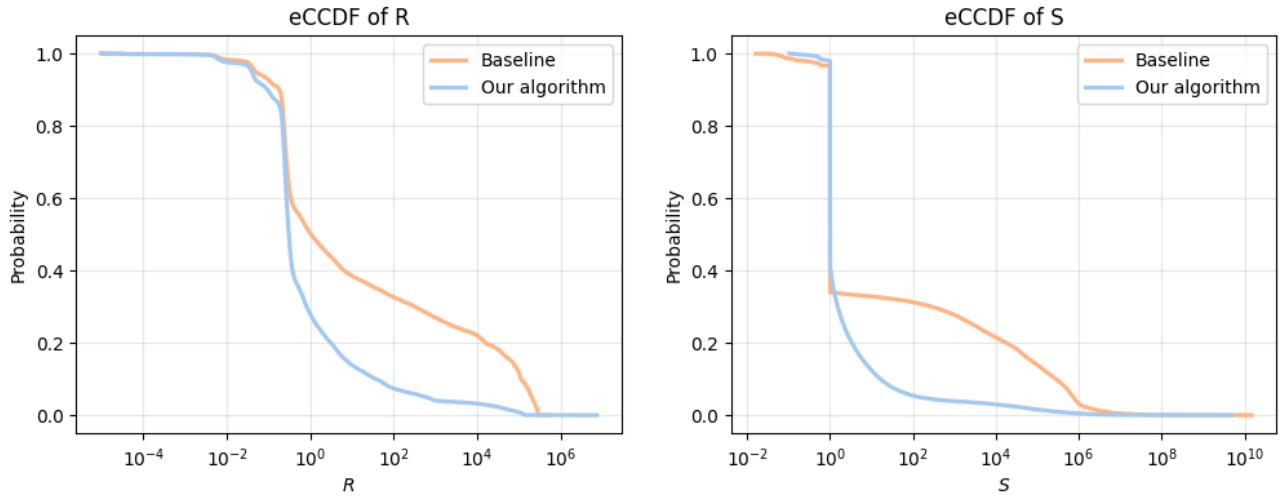

Figure 3: Empirical Complementary Cumulative Distribution Function for R and S