SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF INFORMATION ENGINEERING, ELECTRONICS AND TELECOMMUNICATIONS (DIET)

Challenge #2 - Job dispatching and scheduling

NBD - NETWORKING FOR BIG DATA AND LABORATORY

Professor:
Andrea Baiocchi

Group Viterbi:

Giovanni Giunta	1177327
Mehrzad Jafari Ranjbar	1937944
Borbála Sára Tóth Gy	2066306

Academic Year 2022/2023

1 Algorithms

(i) Our dispatching algorithm is the Least Laxity First (LLF). The actual LLF algorithm schedules the task with the smallest laxity first, and if two tasks have the same laxity, the one with the earliest deadline is chosen. In our case there are no explicit deadlines, so we aim at implementing an approach similar to LLF by simply assigning each incoming task to the server that is currently the least loaded (the one with the minimum unfinished work at the time of assignment), always prioritizing the shortest tasks.

1. When a task arrives, the dispatcher sends a message to each server to understand which one has enough resources (CPU and memory) to accommodate the task.
2. Among the eligible servers, the one with the least current load (i.e., the one expected to become idle the soonest) is chosen by the job dispatcher to take on the new task. The load on a server is given by the amount of CPU required by all the tasks currently assigned to it.
3. In case multiple servers have the same amount of least unfinished work, one of them is chosen randomly to execute the new task.
4. If no server can accommodate the task, the dispatcher adds the task to a queue. The tasks in the queue are checked and sorted after every iteration, waiting to be assigned to servers as resources become available, following the same server selection and task assignment steps.

The execution continues until all the tasks have been delivered to the servers. In the meantime, the servers process their assigned tasks based on the Shortest Remaining Time First (SRTF) algorithm. Given the provided dataset, this LLF implementation comes with a number of advantages:

- LLF aims to distribute the load evenly among all servers, hence improving resource utilization. It acts as a load balancer by assigning each new task to the server with the least unfinished work, and by preventing any one server from becoming a bottleneck. This could be particularly beneficial given that we have set a relatively high CPU capacity of $GNCU = 52788$ in our servers, compared to the low CPU demand required by most tasks (median: 0.072418). Such value of $GNCU$ is needed to process the most computationally expensive task in the dataset.
- LLF is well suited to scenarios where tasks arrive in batches (in our case, all tasks belonging to the same job arrive together), and their service times are known in advance. This could be advantageous as LLF may help in spreading tasks from the same job across multiple servers, potentially reducing the job's completion time.
- LLF can perform quite well when the resources required to run the tasks are mostly homogeneous (apart from some outliers), like in our case.
- Despite being simple, LLF can be an effective algorithm that can scale well as the number of tasks and servers increase. Given that the dataset contains over 2 million tasks, LLF could be a good fit to scale to higher number of tasks.

(ii) Our scheduling algorithm is the Shortest Remaining Time First (SRTF). It is a pre-emptive version of the Shortest Job Next (SJN) algorithm, also known as Shortest Job First (SJF). We use it to schedule tasks on each selected server. Below we give a formal description of the algorithm we implemented:

1. The incoming task is added to the list of current tasks for the chosen server.
2. At each iteration, the current tasks are sorted by their service time, and the one with the shortest remaining processing time is selected
3. The selected task is processed, then the server status is updated.
4. Response times and service times are recorded during the process, as well as messages exchanged with the dispatcher.

Given the provided dataset, this SRTF implementation comes with a number of advantages:

- SRTF gives priority to tasks with the shortest remaining processing time. This means that shorter tasks (those requiring less CPU time) will be executed before longer tasks whenever possible. Given that the median CPU demand of the tasks in the dataset is quite low (0.072418), most tasks should benefit from this prioritization, reducing the average waiting time and potentially help to ensure that the servers' CPU and memory resources are used efficiently.
- By giving preference to tasks with shorter processing times, SRTF can also help achieve a degree of fairness, preventing tasks with short processing times (they are the vast majority) from being delayed behind tasks with long processing times.
- SRTF can be especially effective in scenarios where we know in advance the service times of each task.
- SRTF is a dynamic algorithm, constantly reassessing the remaining time of the servers as tasks come in. This makes it a good choice for workloads where tasks arrive continuously, as in our case.
- Just like LLF, SRTF is a relatively simple algorithm that can scale well with an increase in the number of servers or tasks, making it a good fit for huge workloads that can potentially scale over time.

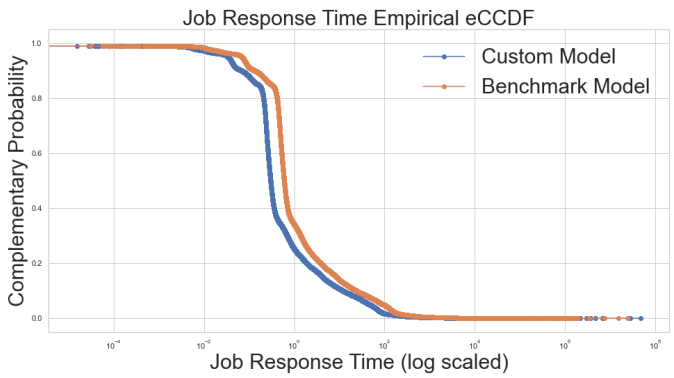
It is important to note that this algorithm operates under the assumption that the exact service time of a task is known upon its arrival (as in our case). Also, we attempted to develop a form of pre-emption: by continually sorting the tasks based on their remaining time, our algorithm effectively pre-empts tasks that have a longer remaining time whenever a task with a shorter remaining time arrives, even though it does not really stop their execution once their processing has started.

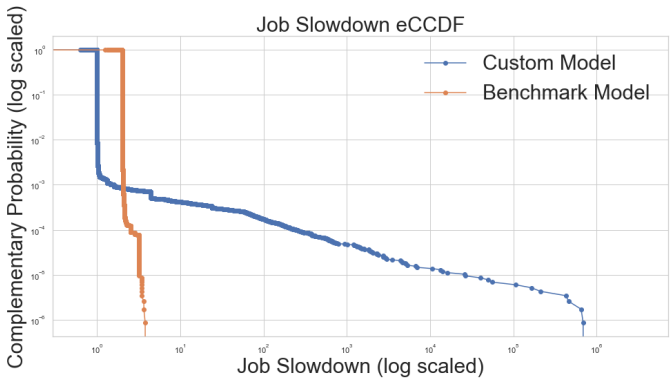
2 Performance evaluation

(i) Here we present $\bar{R}$, $\bar{S}$, ρ, and $\bar{L}$. We also included the max job response time $max(R)$. The discrepancy between $max(R)$ and $\bar{R}$ might be explained by the fact that most jobs have only 1 task (median number of tasks per job: 1; mean number of tasks per job: 2.002), and most tasks have low CPU demand. The table containing the 64 values of ρ_k can be found in the Appendix.

	$max(R)$	$\bar{R}$	$\bar{S}$	ρ	$\bar{L}$
Custom	13:14:03 (hh:mm:ss)	166 ms	50	30:23:53:12 (dd:hh:mm:ss)	130
Baseline	06:34:55 (hh:mm:ss)	158 ms	2	30:23:52:53 (dd:hh:mm:ss)	130

(ii) The bottom left plot represents the eCCDF of the Job Response Times. As we can see, the custom model performs slightly better than the baseline model, as its curve falls more rapidly towards 0. This seems to be in contrast with the mean job response time metric presented above, that appears to be slightly lower for the baseline model. This might be explained by the fact that the eCCDF can be affected by the presence of outliers in our dataset (very few tasks have high demands of CPU, compared to the others). The bottom right plot shows the eCCDF of the Job Slowdowns. At first, the custom model shows a sudden steep drop which suggests a low level of slowdown: that is likely due to the fact that it processes the shorter tasks first. Then, it starts dropping off more gradually, indicating a wider spread of slowdowns. The baseline model, on the other hand, catches up quickly and eventually outperforms the custom model.





(iii) The model implementing the LWL and FCFS algorithms performs marginally better due to its slightly lower mean job response time. The differences look more evident when comparing the two job slowdown values: our custom model does not seem to efficiently handle its arriving tasks, as they are taking 50 times longer than their service time to be completed. The messaging load of the two models is the same. In fact, by design, the job dispatcher and its servers need to exchange $(N \times 2)+2$ messages per task. In the case of the mean overall utilization coefficient of the 64 servers, we have a slightly lower value for the baseline model. However, this metric does not necessarily mean that the baseline model is better in general, as it does not indicate how evenly the workload has been distributed over the servers. In conclusion, the baseline model seems to be the winning model, especially given the better performance achieved with respect to the job slowdown.

3 Appendix

Table 1: Custom Model ρ rounded values (μs)

1	2677951522059	17	2678289783506	33	2678238013221	49	2677661299118
2	2677420930811	18	2677637111082	34	2677944971514	50	2677604772980
3	2678091311351	19	2677908704965	35	2677604772408	51	2678171369049
4	2678298004587	20	2677695196322	36	2677688397129	52	2677596441770
5	2677740070748	21	2678316943788	37	2678015203871	53	2678249738009
6	2678104401317	22	2677983303226	38	2677652505709	54	2677649981973
7	2678060786078	23	2678318003285	39	2677823398343	55	2677760222623
8	2678021138565	24	2678282349729	40	2678307099125	56	2678298004584
9	2678018010780	25	2678141670434	41	2678289645919	57	2678268712169
10	2677677563606	26	2678274373067	42	2678268712200	58	2677601069589
11	2677577159965	27	2678247269089	43	2678106019692	59	2677910251713
12	2677568750112	28	2677674479898	44	2678319872827	60	2678271235841
13	2678260290681	29	2678282513564	45	2678263153951	61	2678219236804
14	2678133675758	30	2677607996610	46	2678057030628	62	2678017727548
15	2678309644172	31	2678307120468	47	2677765105548	63	2678365834101
16	2678066080433	32	2677579684059	48	2677968642678	64	2677653298162

Table 2: Baseline Model ρ rounded values (μs)

1	2678307120492	17	2678298004588	33	2677513549037	49	2677762825267
2	2677577163966	18	2678002882531	34	2677765161253	50	2678271235868
3	2678084492360	19	2678015487559	35	2677910251719	51	2678133676138
4	2678296324586	20	2677571356012	36	2678018995542	52	2678103156748
5	2678271235911	21	2678246916751	37	2678015872151	53	2677672301830
6	2678298004592	22	2678260290683	38	2677359477945	54	2678014765583
7	2677652474995	23	2677849075877	39	2677548927022	55	2677691748005
8	2678058365983	24	2678260630346	40	2678289783533	56	2677951494168
9	2678263154033	25	2677792376494	41	2678271235895	57	2678178362094
10	2678249792813	26	2677692418316	42	2677868582117	58	2677747581512
11	2678018335848	27	2677611498952	43	2678066080470	59	2678189365298
12	2677654073865	28	2677671957827	44	2677687956535	60	2678309644216
13	2678348263133	29	2678296324557	45	2677571355489	61	2677577163313
14	2678289524799	30	2677579685970	46	2677571355627	62	2677944972017
15	2677762582415	31	2678017757934	47	2678289783553	63	2678279747210
16	2678155858834	32	2678303640693	48	2677652505896	64	2678289483314