TWS Exercise 3: analysing a conclusion

Danilo Carastan-Santos and Enikő Kevi

13^{th} of October 2023

1 Exercise Instructions

From the conclusion example below[1], identify the conclusion narrative elements (scope and importance of the work, work summary and short analysis if the results, future perspectives) and revise the text according to the guidelines seen so far during the course (balancing precision with clarity, avoiding ambiguity, sustaining energy, connecting your ideas, and beginning with the familiar).

Conclusion Example

As the scale and power of high-performance computing (HPC) platforms increase, it becomes more crucial to deploy efficient resource management approaches (notably scheduling algorithms) in order to prevent the dampening of such increase in computing power. In an adversarial manner, it is also important that such scheduling algorithms stay simple and easily understandable by the users. Furthermore, changing the scheduling algorithm is often seen as a risky move, mainly due to the possibility of having unseen and unpredictable changes in the performance of the platform, which could be detected only after a long period of time.

In this paper, we move towards providing more knowledge and experience on what are the expectations if one decides to change the First-Come-First-Served (FCFS) scheduling policy with aggressive backfilling – the popular EASY Backfilling – scheduling algorithm. We selected a class of simple scheduling algorithms that differs from EASY Backfilling by changing the scheduling policy (other than FCFS) and adding a thresholding mechanism (to provide the same no starvation guarantees as FCFS). We used a flexible and reliable simulation software and exploited the rich information presented in HPC platform workload traces to find what could be observed and gained by using these other simple scheduling algorithms rather than EASY Backfilling.

Our results indicate that one can only gain by replacing EASY Backfilling with simple policies that consider the estimated processing time and the required resources, notably the Shortest Processing time First (SPF) and Shortest Area First (SAF). By adding a simple thresholding mechanism, it is possible to obtain significant performance improvements for the long run, using three relevant performance objectives, while also guaranteeing that every job in the waiting queue will eventually be executed. We show that these simple policies not only present better performance in average values, but they also significantly increase the number of jobs executed instantly (without waiting) and lower the number of jobs that wait for a long time. The performance gains over EASY Backfilling is distributed among all waiting jobs.

These simple policies also show that they can perform well with less interference from backfilling: the scheduler is more likely to follow the original order as set by the chosen scheduling policy, and not by the rules of backfilling, thus providing more predictability and transparency, two properties that are sought by HPC platform administrators.

We also highlight a less known scheduling policy in the literature, the Shortest Area First (SAF). In our experimental campaign, we found that this policy managed to consistently provide close to the best (if not the best) observed performance in all scenarios and performance objectives we evaluated. For instance, considering the slowdown objective, SAF not only provided an average overall performance increase up to 83.4%, but as well increased the number of jobs that run immediately by up to 9% and lowered the number of jobs who waited for a long time (very long slowdown) by up to 2.8 times, in comparison with FCFS. This result reinforces the relevance of the jobs' area property, which was seen in our previous work, and raises the question about possible analytical properties of SAF. Nevertheless, we reinforce that SAF must be considered as a baseline of comparison in future parallel batch scheduling research.

Last but not least, we present some cautions that must be considered if one wants to provide a scheduling algorithm that minimizes the maximum of an objective function. Taking the slowdown objective function as an example, we observed a class of jobs whose presence in the workload is not negligible and can mistakenly lead to inflated maximum slowdown values. If one only looks at the maximum of an objective function to evaluate the scheduling performance, some good scheduling policies (as the aforementioned ones) can be overlooked.

There is still work to be done in this subject: the first point is to address how users play a role in the global scheduling. As mentioned in Section III-C, our approach on evaluating the scheduling performance is centred at the platform and the jobs, where users are not taken into account. An ideal scenario would be to simulate the users reacting to the scheduler's performance. In this regard, accurate and reliable user models are required to properly simulate user behavior. Another point is how we can exploit SAF to provide SAF-like scheduling policies that are adapted to certain situations and/or time periods.

[1] Conclusion extracted from D. Carastan-Santos, R. Y. De Camargo, D. Trystram and S. Zrigui, "One Can Only Gain by Replacing EASY Backfilling: A Simple Scheduling Policies Case Study," 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Larnaca, Cyprus, 2019, pp. 1-10, doi: 10.1109/CCGRID.2019.00010.