Goodput optimization with a variable number of processors

Joachim Cendrier, Anne Benoit, Frédéric Vivien

LIP, ENS Lyon, CNRS, Inria

{joachim.cendrier, anne.benoit, frederic.vivien}@ens-lyon.fr

Keywords : *scheduling, variable capacity resources, goodput, dynamic programming.*

Introduction. Because of the growing concern about their energy consumption and impact on the environment, more and more data centers aim at using renewable energies in order to improve their sustainability [2]. From the user point of view, this amounts at seeing the capacity of the HPC platform on which the application is executed evolve over time. For instance, the power supply may be reduced when using coal instead of wind or solar energy. This problem is termed the *variable capacity scheduling* problem [3], and some preliminary solutions have already been proposed [1].

In this work, we go beyond previous approaches by allowing jobs to be checkpointed before a resource change, hence avoiding loosing some work. We assume that the resource provider *warns* the user before a change in the number of processors, hence it is possible to anticipate and take checkpoints before the change happens. However, we do not know the exact number of processors after the change. The goal is then to optimize the goodput within the next period (time between two changes in the number of processors). We first detail the model, then we propose some preliminary solutions and conclude.

1 Model

We consider J infinite jobs j_i, with $1 \leq i \leq J$. Job j_i has to be executed on p_i processors, and it can be checkpointed within a time C_i. The time of recovery for this job is R_i, that corresponds to the time to read the checkpoint to be able to resume the execution of the job.

The jobs are to be executed on a platform with a varying number of processors: the number of available processors regularly changes, at time $T_0, T_1, \ldots, T_n, \ldots$. In a regular pattern, the duration between two changes is constant ($T = T_{n+1} - T_n$ for all value of n). Hence, P_n is the number of processors available from time T_n to time T_{n+1}. At each change in the number of processors, we may gain or loose up to k processors, i.e., $|P_{n+1} - P_n| \leq k$.

The goal is to optimize the use of the processors according to various objective functions, adapting to each change in the number of processors over time. No work will be wasted, since we are always able to checkpoint a job before shutting it down, by anticipating checkpoints because we know that at maximum k processors will disappear at time T_n. Hence, at least k checkpoints are taken just before an expected decrease of the number of processors. Additional checkpoints can be taken right after T_n, once the exact number of available processors P_n is known, to start a new set of jobs that fills the available processors as good as possible, and keep as few processors idle as possible.

We focus in this work on the optimization of the *Goodput* per period $[T_n, T_{n+1}]$, which depends on the time during which useful work has been completed by the running jobs. Within the period, the total available time for work is $P_n(T_{n+1} - T_n)$, but some of this time may be used by job checkpoints or recoveries, or when some processors remain idle. The goodput is the ratio of useful work over this total amount of work:

$$\mathit{Goodput}([T_n, T_{n+1}]) = \frac{\sum_{i=1}^{J} p_i W_n^i}{P_n(T_{n+1} - T_n)},$$

where W_n^i is the duration during which useful work is performed by job j_i within the period. Figure 1 presents an example, where the useful work is represented by the blue zones.

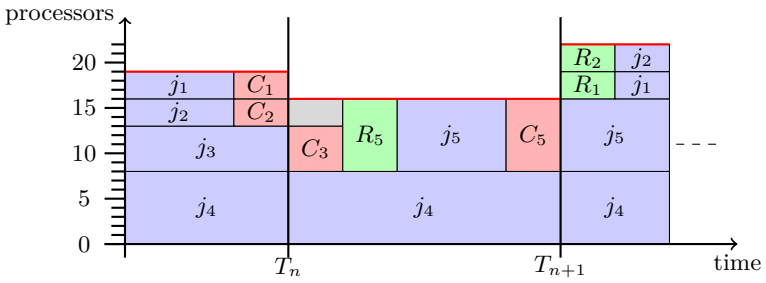

FIG. 1: In this example, we have five jobs taking 3, 3, 5, 8 and 8 processors respectively, and we have a total of 19 processors running at first, then 16, then 22. In blue, we have the useful work phases, in red the checkpoint phases, and in green the recovery phases. Finally, in grey, the processors are switched on but not working, as other processors must be available to work on a larger job.

2 Algorithms

In order to anticipate from loosing processors, the first natural solution consists in checkpointing all jobs before T_n, and then decide which jobs to run at time T_n. The goal is to have as many jobs running as possible, hence we use a subset sum algorithm on the p_i's to fit the total number of available processors P_n. With negligible checkpoint and recovery times ($C_i = R_i = 0$ for $1 \leq i \leq J$), this is optimal.

We also propose a sophisticated dynamic programming algorithm that accounts for checkpointing and recovery times, and computes a solution with optimal goodput on the period. Given a set of checkpointed jobs just before T_n, covering at least k processors, there are three categories of jobs before T_n: the jobs checkpointed, the jobs running but not checkpointed, and the jobs not running. The dynamic programming iteratively decides whether job j_i ($1 \leq i \leq J$) should be executed or not at time T_n, depending of its previous state. It might need to be checkpointed after T_n to be switched off (see j_3 in the example), or may require some recovery, either at time T_n or after some checkpoints have been taken on other processors (see j_5).

With constant checkpointing times ($C_i = C$ for $1 \leq i \leq J$), we formally prove that all additional checkpoints should be taken at time T_n, and the algorithm checks all cases to obtain an optimal solution within a time $O(J \times {P_n}^3)$. The algorithm also works with different C_i's: if JC_n is the set of jobs checkpointed after T_n, all of these checkpoints are aligned so that they end at time $T_n + \max_{i \in JC_n} C_i$, hence some work can be performed before the checkpoint to contribute to the goodput. This leads to a complexity of $O(J^2 \times {P_n}^3)$.

3 Conclusion

This preliminary work provides solutions that optimize the goodput within a period. However, it might not be fair for some jobs that may never be executed. A first natural extension is to account for the yield of jobs, which is the ratio of its actual progress rate over the progress rate that would have been achieved if the job was executing alone on the platform. The goal is hence to maximize the minimum yield so that all jobs are treated fairly. We also plan to combine both objectives in a bi-criteria approach.

References

[1] L. Perotin, C. Zhang, R. Wijayawardana, A. Benoit, Y. Robert, and A. Chien. Risk-aware scheduling algorithms for variable capacity resources. In *Proc. of SC '23 Workshops (PMBS)*, page 1306–1315, 2023.

[2] A. Radovanovic. Our data centers now work harder when the sun shines and wind blows, Apr 2020. `https://blog.google/inside-google/infrastructure/data-centers-work-harder-sun-shines-wind-blows`.

[3] C. Zhang and A. A. Chien. Scheduling challenges for variable capacity resources. In *Job Scheduling Strategies for Parallel Processing*, pages 190–209. Springer, 2021.