HPC WORKLOAD BALANCING ALGORITHM FOR CO-SCHEDULING ENVIRONMENTS

R.I. Kuchumov, V.V. Korkhov[a]

Saint Petersburg State University, 7/9 Universitetskaya nab., St. Petersburg, 199034, Russia

E-mail: [a] v.korkhov@spbu.ru

The goal of this research work is to reduce wait time of HPC (high performance computing) applications in schedulers queue by applying a co-scheduling strategy. This strategy allows the execution of more than one task with different non-overlapping requirements for computational resources simultaneously. Co-scheduling strategy reduces task queue wait time and improves utilization of cluster resources when compared to the scheduling strategies that do not allow for parallel task execution on the same machine. We have proposed a method for measuring application processing speed in its run-time, which can be used as a feedback for scheduling strategies. In this work, we have formalized the co-scheduling problem and proposed strategies for solving it. For some strategies we have shown analytically the upper bounds values of their competitive ratios. Besides that for the proposed scheduling strategies we ran numerical experiments using imitation models to show how they compare to the optimal strategy.

Keywords: Co-scheduling, HPC, Scheduling theory, Stochastic Optimization.

Ruslan I. Kuchumov, Vladimir V. Korkhov

1. Introduction

There are several case studies (e.g [1-3]) showing that by running more than one HPC application on the same computational node it is possible to increase cluster throughput, decrease schedule makespan and to decrease power consumption. The scheduling strategy that allows to run multiple tasks on the same machine in the literature is usually referred to as co-scheduling or co-location strategies. Commonly used batch schedulers do not allow implement such strategy in a controllable way, as they work either by assigning the whole computational node or by assigning a subset of CPU cores (so called slots) to a single task without considering task performance degradation.

In this work we propose a method of measuring task processing speed in its runtime. This metric can be used as a feedback for dynamic scheduling algorithms to quantitatively evaluate co-scheduling decisions. We have formalized co-scheduling problem in terms of scheduling theory and have proposed several strategies that vary by the amount of initial information about each task they require and by the optimality of the schedules they produce. We have analytically shown how greedy strategy compares to an optimal strategy and based on that proposed several stochastic strategies that approximate a greedy strategy and use task processing speed metric as a feedback.

2. Co-Scheduling problem and its strategies

To provide scheduling strategies that implement a co-scheduling approach, we formalised the problem in terms of scheduling theory. We have considered a static problem definition where the number of tasks and their parameters (processing speed, amount of work) does not change in time. In this work we consider makespan (completion time of the last task) as an objective function.

Denote a set of n tasks as $T=\{T_1,...,T_n\}$. Each T_i requires a b_i amount of work units to be completed. Values of b_i form an n-dimensional vector b. Any combination (subset) for T can be simultaneously on the same machine. There are a total of $m = 2^{n-1}$ possible combinations of n tasks. We will denote S_j as a combination of tasks T (S_j is in the power set of T and non-empty).

Denote $a_{i,j}$ as a speed of T_i in combination S_j. Without loss of generosity we will consider the processing speed of any task T_i running alone (i.e. in combination $S_j = \{T_i\}$) to be equal to 1. Values $a_{i,j}$ form a matrix A with dimensions n by m. As task processing speed can not increase, when a new tasks are added to the combination, we can introduce the following constraint on $a_{i,j}$ values:

$$a_{i,p} \leq a_{i,q},\ \forall\ (p, q) \mid S_p \subseteq S_q,\ i = 1,..., n \qquad (1)$$

At first, we will consider a deterministic offline problem to provide an optimal solution. In this problem definition, exac values of all task parameters (A, b) are known in advance. Denote x_j as a total amount of time the combination S_j were run in a schedule. Then the problem of makespan optimization can be solved by reducing it to a linear programming problem of minimizing a sum of x_j, subject to $Ax=b$ and $x \geq 0$.

This solution can not be used in practice as it requires all task data to be available. To mitigate this, we have considered an online problem definition, where values of the vector b are unknown in advance. To solve this problem, we have proposed to use a scheduling strategy that selects the combination of remaining tasks with the maximum sum of tasks processing speed. We refer to this strategy as FCS (Fastest Combination Speed-first). We have analytically shown [4] that the makespan of FCS strategy can not be larger than the optimal makespan by more than 2 times.

Online definition of this problem still can not be applied in schedulers implementations, as it requires values of matrix A to be known in advance. As FCS strategy makes local optimal decisions (a greedy strategy), provides a small competitive ratio and values of task processing speed can be measured in practice, we have covered several strategies that approximate FCS strategy.

We have considered a non-deterministic definition of co-scheduling problem, where task processing speed $a_{i,j}$ are normal random variables and their parameters can be estimated in tasks run-

time. To do that, we have relaxed constraint (1) by defining variances of $a_{i,j}$ so that constraint (1) holds with a given probability. We have proposed several stochastic search strategies that approximate FCS by searching for the combination with maximum sum of task processing speed. To interpolate values of $a_{i,j}$ and ensure that task variance increases with an increase of uncertainty, we used linear spline interpolation as described in [4].

In this work, we have considered 3 search based strategies, varying by their acquisition functions: PI (probability of improvements), EI (expected improvements) and UCB (upper confidence bounds). We have compared them with FCS and the optimal strategy using the simulation software that we developed. Task processing speed values in different combinations were measured in the experiments and then they were used to generate similar input for numerical simulations. Simulation results are shown in figure 1.

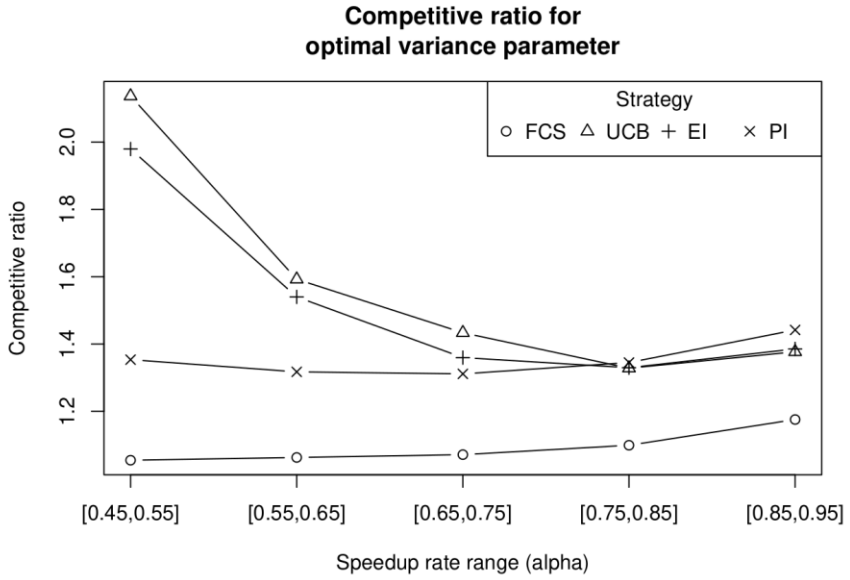

Figure 1. Comparison of the lowest competitive ratio values produced by different scheduling strategies. Speedup rate range is a range of task processing speed decrements when the number of tasks in combinations increases.

3. Processing speed metrics

Tasks running on the same computational node may share bandwidth of such resources as memory bus, network card, last level cache, disk bandwidth, etc. When bandwidth for any of these resources is fully utilized, task processing speed decreases as CPU instructions performing access to the resource take more cycles to complete. That is, when multiple tasks interfere with each other due to co-scheduling IPC (instructions per cycle) values will decrease.

Besided IPC, CPU time may also be affected by co-scheduling as operating system scheduler may assign more than one task thread to a single CPU core. In this case, CPU time would be shared between all of the threads and the more threads there are, the smaller fraction of CPU time would be available to each task.

In this work we proposed to measure task processing speed in runtime as IPC multiplied by CPU time fraction values averaged across all tasks threads. To show that this metric indeed measures task processing speed, we found a set of benchmark applications that have a constant IPC value throughout their runtime and compared how this metric changes in different conditions compared to the change of total task processing time. For that we picked 10 benchmark applications from NPB and

Parsec and a few of our own benchmarks and run them in combinations of a different size. A complete list of benchmarks is presented in [4].

Results for some benchmarks are shown in figure (2) along with a complete list of benchmarks. The figure shows a scatter plot of processing time ratio version processing speed ratio (as measured by the proposed metric). Each point corresponds to measurements of a single task in combinations with other tasks.

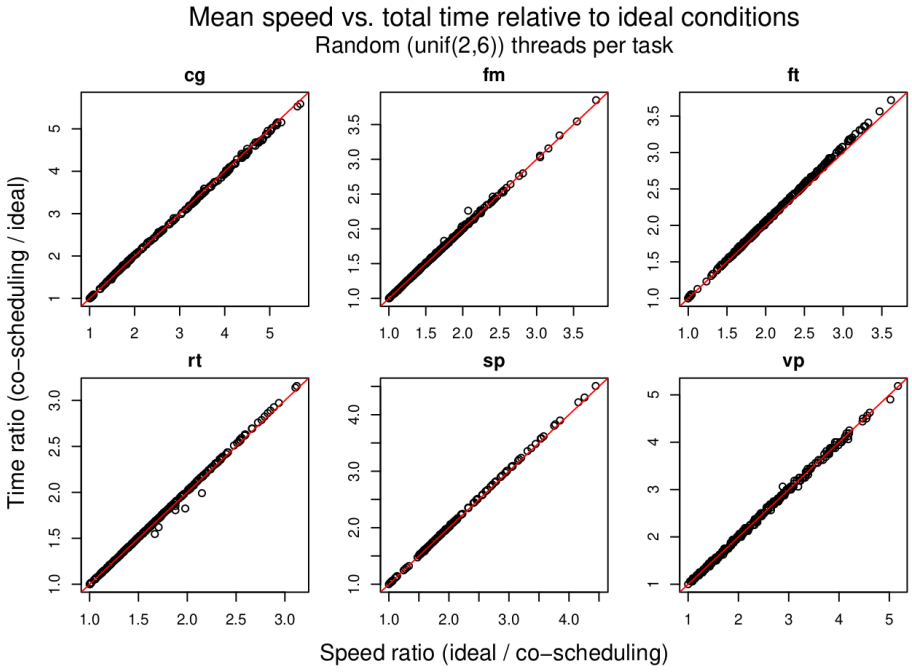

Figure 2. Scatter plots of task processing time ratio version processing speed ratio. Each point corresponds to tasks run in different combinations with other tasks. Red line is a line where ratios are equal.

4. Results and Conclusion

We have formalized the co-scheduling problem in terms of scheduling theory and have proposed multiple strategies that provide a solution for the problem. An optimal strategy was proposed for offline deterministics problem definitions. For online deterministic problem we have proposed a greedy strategy (FCS) and have analytically shown that its competitive ratio can not be larger than two. For non-deterministics counterpart problem we have proposed three search-based strategies (PI, EI, UCB) that use different acquisition functions. Using numerical simulations we have shown (figure 1) how competitive ratio compares between different scheduling strategies. Simulation results show that PI produces a lower value of competitive ratio than EI and UCB strategies for almost all problem input ranges.

We have proposed a method for measuring task processing speed in its runtime and have validated it using test applications that mimic HPC workloads. Results of the evaluation (figure 2) showed that this metric measures task processing speed with a very high accuracy in different conditions. These results allow to apply proposed scheduling strategies in schedulers implementations.

5. Acknowledgement

Research has been supported by the RFBR grant 19-37-90138.

References

[1] Trinitis, Carsten, and Josef Weidendorfer, eds. Co-scheduling of HPC applications. Vol. 28. IOS Press, 2017.

[2] de Blanche, Andreas, and Thomas Lundqvist. "Node Sharing for Increased Throughput and Shorter Runtimes–an Industrial Co-Scheduling Case Study." Proceedings of the 3rd Workshop on Co-Scheduling of HPC Applications (COSH 2018). 2018.

[3] Breslow, Alex D., et al. "The case for colocation of hpc workloads." Concurrency and Computation: Practice and Experience Preprint (2012).

[4] Kuchumov, R.; Korkhov, V. Analytical and Numerical Evaluation of Co-Scheduling Strategies and Their Application. Preprints 2021, 2021090053 (doi: 10.20944/preprints202109.0053.v1).