Fault tolerant scheduling of non-uniform tasks under resource augmentation

Dariusz R. Kowalski *and* Prudence W.H. Wong (Speaker) [*] Elli Zavou [†]

1 Introduction

Dealing with computationally intensive jobs is becoming a necessity rather than an additional advantage of new computational systems. Some of the multiple challenges that appear with the complexity of such systems include the dynamicity of job (or task) arrivals, the diversity of their computational demands (e.g. different processing times), the unpredictable machine failures, as well as the preservation of power consumption.

In this work we focus on the simple model of one single machine prone to unpredictable crashes and restarts, and tasks of sizes $c \in [c_{min}, c_{max}]$ arriving dynamically in the system. Values c_{min} and c_{max} represent the smallest and largest processing times a task may need respectively, when executed by the machine running without additional *resource augmentation*. We consider a parameter s representing *speedup*; the amount of resource augmentation added to the machine, such that the processing time of a task of size c becomes c/s. We apply resource augmentation to overcome the machine failures, as an alternative to using more processing entities (e.g. multiprocessor systems).

Due to the unpredictable nature of the machine and the dynamicity of task arrivals, we consider crash, restart and injection patterns to be controlled by an adversarial entity $\mathcal{A}$, and perform worst-case competitive analysis for the performance of online scheduling algorithms. We focus on two efficiency measures: the *completed time*, which is the aggregate size of all tasks that have been completely executed, and *latency*, which is the longest time a task spends in the system. In some sense, the former corresponds to the *utilization* of the machine, while the latter on the *fairness* of the scheduling algorithm.

In a previous work, Fernández Anta et al. [2] looked at the *pending time* competitiveness of a similar system of multiple machines and showed that in order to achieve competitiveness, it is necessary to use speedup. They proved the NP-hardness of the offline version of the problem and gave lower bounds on speedup, under which no competitiveness can be achieved. These were given by conditions **C1**: $s < \rho$ and **C2**: $s < 1 + \gamma/\rho$, where $\rho = c_{max}/c_{min}$, the ratio of maximum over minimum task sizes, and $\gamma > 0$ a parameter that represents the number of c_{min} tasks that a machine with speedup s can complete in addition to a c_{max} task, in an interval of length $(\gamma + 1)c_{min}$. In a different line of work and environment, Fernández Anta et al. [1] have shown that even with no speedup, an algorithm that gives priority to the shortest tasks can achieve

[*]D.Kowalski@liverpool.ac.uk and pwong@liverpool.ac.uk. Department of Computer Science, University of Liverpool, L69 3BX Liverpool, UK.

[†]elli.zavou@imdea.org. Universidad Carlos III de Madrid and IMDEA Networks Institute, 28911 Madrid, Spain. PhD Candidate partially supported by FPU Grant from MECD, Spain.

completed time competitiveness at most $1/(\rho+1)$. Following their line of work, Jurdzinski et al. [3] proposed an algorithm that generalized the results of [1] for a fixed number of different task sizes (more than two), and improved the competitiveness to 1-completed-time-competitiveness, when working with speedup $s = 2$. Another requirement for this algorithm to work, is the divisibility property of the task sizes. We therefore hope to be able to give an algorithm that needs less resource augmentation to achieve 1-completed-time-competitiveness, even if some restrictions apply on the task sizes.

2 Results

Our first result in this work involves the speedup threshold for non-competitiveness. It is summarized in the following theorem, whose proof is based on defining and analyzing two different adversarial strategies (one for each efficiency measure), under which no algorithm can be competitive, either regarding latency or completed time. Roughly speaking, the adversary attempts to force the online algorithm unable to complete the c_{max}-task and hence incurring infinite latency.

Theorem 1 *For any given c_{min}, c_{max} and s, if both conditions* **C1** *and* **C2** *are satisfied, NO deterministic online algorithm is latency competitive, or 1-completed-time-competitive when run with speedup s against an adversary that injects tasks of sizes $c \in [c_{min}, c_{max}]$, even in a system with one single machine.*

However, considering the result in [1], we introduce a deterministic scheduling algorithm γ-Burst, for the case of only two task sizes, which achieves both 1-latency-competitiveness and 1-completed-time-competitiveness as soon as condition **C2** does not hold (even if condition **C1** still holds, i.e. $s \in [1 + \gamma/\rho, \rho)$). Observe that the speedup required is less than $s = 2$ needed for the algorithm in [3].

Algorithm γ-Burst. It separates the pending tasks in two lists according to their size and sorts them according to their arrival time. This way, the next task to be scheduled from each list, if one of that size is to be scheduled, is the first task (being the one that has been waiting the longest in the system). It then takes its scheduling decisions at the end of each *stage*, which also indicates the beginning of a new one. A stage ends either by being interrupted by a machine crash or by the completion of all the tasks that were decided at the beginning of the stage to be scheduled within the stage. The scheduling decisions are taken based on the following rules:
1. If there are no c_{max} tasks pending, then γ-Burst schedules a c_{min} task.
2. If there are no c_{min} tasks pending, then it schedules a c_{max} task.
3. Else, if there are at least γ tasks of size c_{min} pending, it schedules γ c_{min}-tasks consecutively followed by a c_{max} task.
4. Otherwise, it schedules tasks from the two lists alternatively. In this case, the stage ends after a single task is completed.

Theorem 2 *For any given c_{min}, c_{max} and speedup s satisfying condition* **C1** $\wedge \neg$**C2***, i.e. $s \in [1 + \frac{\gamma}{\rho}, \rho)$, algorithm γ-Burst is 1-latency-competitive and 1-completed-time-competitive.*

The proof of the results claimed in the theorem above is based on the analysis of latency for each group of task sizes, as well as the exhaustive analysis of the completed time in different types of stages.

References

[1] Antonio Fernández Anta, Chryssis Georgiou, Dariusz R. Kowalski, Joerg Widmer, and Elli Zavou. Measuring the impact of adversarial errors on packet scheduling strategies. In *Structural Information and Communication Complexity - 20th International Colloquium, SIROCCO 2013, Ischia, Italy, July 1-3, 2013, Revised Selected Papers*, pages 261–273, 2013.

[2] Antonio Fernández Anta, Chryssis Georgiou, Dariusz R. Kowalski, and Elli Zavou. Online parallel scheduling of non-uniform tasks: Trading failures for energy. In *Fundamentals of Computation Theory - 19th International Symposium, FCT 2013, Liverpool, UK, August 19-21, 2013. Proceedings*, pages 145–158, 2013.

[3] Tomasz Jurdzinski, and Dariusz R Kowalski, and Krzysztof Lorys. Online packet scheduling under adversarial jamming. In *Approximation and Online Algorithms: 12th International Workshop, WAOA 2014, Wrocław, Poland, September 11-12, 2014, Revised Selected Papers*, volume 8952, page 193. Springer, 2015.