Future Generation Computer Systems 78 (2018) 245–256

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Competitive analysis of fundamental scheduling algorithms on a fault-prone machine and the impact of resource augmentation

Antonio Fernández Anta [a], Chryssis Georgiou [b], Dariusz R. Kowalski [c], Elli Zavou [a,d,*]

[a] IMDEA Networks Institute, Av. del Mar Mediterráneo 22, 28918 Leganés, Madrid, Spain

[b] University of Cyprus, Department of Computer Science, 75 Kallipoleos Str., P.O. Box 20537, 1678 Nicosia, Cyprus

[c] University of Liverpool, Department of Computer Science, Ashton Building, Ashton Street, Liverpool L69 3BX, United Kingdom

[d] University Carlos III of Madrid, Department of Telematics Engineering, Torres Quevedo Building, Av. Universidad 30, 28911 Leganés, Madrid, Spain

HIGHLIGHTS

- Worst-case analysis of fault-tolerant properties of popular scheduling algorithms.
- Competitive analysis regarding completed/pending load and latency of the algorithms.
- Use of resource augmentation to achieve and/or to improve their performance.
- Differences of scheduling policies based either on *arrival time* or *size* of tasks.
- All deterministic and work-conserving algorithms require speedup to be competitive.

ARTICLE INFO

Article history:
Received 11 November 2015
Received in revised form
27 May 2016
Accepted 29 May 2016
Available online 6 June 2016

Keywords:
Scheduling
Online algorithms
Different task processing times
Failures
Competitive analysis
Resource augmentation

ABSTRACT

Reliable task execution in machines that are prone to unpredictable crashes and restarts is both challenging and of high importance, but not much work exists on the analysis of such systems. We consider the online version of the problem, with tasks arriving over time at a single machine under worst-case assumptions. We analyze the fault-tolerant properties of four popular scheduling algorithms: Longest In System (LIS), Shortest In System (SIS), Largest Processing Time (LPT) and Shortest Processing Time (SPT). We use three metrics for the evaluation and comparison of their competitive performance, namely, completed load, pending load and latency. We also investigate the effect of resource augmentation in their performance, by increasing the speed of the machine. Hence, we compare the behavior of the algorithms for different speed intervals and show that there is no clear winner with respect to all the three considered metrics. While SPT is the only algorithm that achieves competitiveness on completed load for small speed, LIS is the only one that achieves competitiveness on latency (for large enough speed).

1. Introduction

Motivation. The demand for processing dynamically introduced jobs that require high computational power has been increasing dramatically during the last decades, and so has the research to face the many challenges it presents. In addition, with the presence of machine failures (and restarts), which in cloud computing is now the norm instead of the exception, things get even worse. In this work, we apply *speed augmentation* [1,2] (i.e., we increase the computational power of the system's machine) in order to overcome such failures, even in the worst possible scenario. This is an alternative to increasing the number of processing entities, as done in multiprocessor systems. Hence, we consider a speedup $s \geq 1$, under which the machine performs a job s times faster than the baseline execution time.

More precisely, we consider a setting with a single *machine* prone to crashes and restarts being controlled by an adversary (modeling worst-case scenarios), and a *scheduler* that assigns injected jobs or *tasks* to be executed by the machine. These tasks arrive continuously and have different computational demands and hence *size* (or processing time). Specifically we assume that each task τ has size $\pi(\tau) \in [\pi_{\min}, \pi_{\max}]$, where $\pi_{\min}$ and $\pi_{\max}$ are the smallest and largest possible values respectively, and

* Corresponding author at: IMDEA Networks Institute, Av. del Mar Mediterráneo 22, 28918 Leganés, Madrid, Spain.
E-mail addresses: antonio.fernandez@imdea.org (A. Fernández Anta), chryssis@cs.ucy.ac.cy (C. Georgiou), D.Kowalski@liverpool.ac.uk (D.R. Kowalski), elli.zavou@imdea.org (E. Zavou).

http://dx.doi.org/10.1016/j.future.2016.05.042
0167-739X/

$\pi(\tau)$ becomes known to the system at the moment of τ's arrival. Since the scheduling decisions must be made continuously and without knowledge of the future (neither of the task injections nor of the machine crashes and restarts), we look at the problem as an *online* scheduling problem [3–7]. The importance of using speedup lies in this online nature of the problem; the future failures, and the instants of arrival of future tasks along with their sizes, are unpredictable. Thus, there is the need to overcome this lack of information. Epstein et al. [8], specifically show the impossibility of competitiveness in a simple non-preemptive scenario (see Example 2 in [8]). We evaluate the performance of the different scheduling policies (*online algorithms*) under *worst-case* scenarios, on a machine with speedup s, which guarantees efficient scheduling even in the worst of cases. For that, we perform competitive analysis [9]. The four scheduling policies we consider are *Longest In System* (LIS), *Shortest In System* (SIS), *Largest Processing Time* (LPT) and *Shortest Processing Time* (SPT). Scheduling policies LIS and SIS are the popular FIFO and LIFO policies respectively. Graham [10] introduced the scheduling policy LPT a long time ago, when analyzing multiprocessor scheduling. Lee et al. [11] studied the offline problem of minimizing the sum of flow times in one machine with a single breakdown, and gave tight worst-case error bounds on the performance of SPT. Achieving reliable and stable computations in such an environment withholds several challenges. One of our main goals is therefore to confront these challenges considering the use of the smallest possible speedup. However, our primary intention is to unfold the relationship between the efficiency measures we consider for each scheduling policy, and the amount of speed augmentation used.

Contributions. In this paper we explore the behavior of some of the most widely used algorithms in scheduling, analyzing their fault-tolerant properties under worst-case combination of task injection and crash/restart patterns, as described above. The four algorithms we consider are:

(1) *Longest In System* (LIS): the task that has been waiting the longest is scheduled; i.e., it follows the FIFO (*First In First Out*) policy,
(2) *Shortest In System* (SIS): the task that has been injected the latest is scheduled; i.e., it follows the LIFO (*Last In First Out*) policy,
(3) *Largest Processing Time* (LPT): the task with the biggest size is scheduled, and
(4) *Shortest Processing Time* (SPT): the task with the smallest size is scheduled.

We focus on three *evaluation metrics*, which we regard to embody the most important quality-of-service parameters: the *completed load*, which is the aggregate size of all the tasks that have completed their execution successfully, the *pending load*, which is the aggregate size of all the tasks that are in the queue waiting to be completed, and the *latency*, which is the largest time a task spends in the system, from the time of its arrival until it is fully executed. Latency, is also referred to as *flowtime* in scheduling (e.g., [12,13]). These metrics represent the machine's throughput, queue size and delay respectively, all of which we consider essential. They show how efficient the scheduling algorithms are in a fault-prone setting from different angles: machine utilization (completed load), buffering (pending load) and fairness (latency). The performance of an algorithm ALG is evaluated under these three metrics by means of competitive analysis, in which the value of the metric achieved by ALG when the machine uses speedup $s \geq 1$ is compared with the best value achieved by any algorithm X running without speedup ($s = 1$) under the same pattern of task arrivals and machine failures, at all time instants of an execution.

Table 1 summarizes the results we have obtained for the four algorithms. The first results we show apply to *all deterministic algorithms* and *all work-conserving algorithms*—algorithms that do not idle while there are pending tasks and they do not break the execution of a task unless the machine crashes. We show that, if task sizes are arbitrary, these algorithms cannot be competitive when processors have no resource augmentation ($s = 1$), thus justifying the need of the speedup. Then, for work-conserving algorithms we show the following results: (a) When $s \geq \rho = \frac{\pi_{\max}}{\pi_{\min}}$, the completed load competitive ratio is lower bounded by $1/\rho$ and the pending load competitive ratio is upper bounded by ρ. (b) When $s \geq 1 + \rho$, the completed load competitive ratio is lower bounded by 1 and the pending load competitive ratio is upper bounded by 1 (i.e., they are 1-competitive). Then, for specific cases of speedup less than $1+\rho$ we obtain better lower and upper bounds for the different algorithms.

However, it is clear that none of the algorithms is better than the rest. With the exception of SPT, no algorithm is competitive in any of the three metrics considered when $s < \rho$. In particular, algorithm SPT is competitive in terms of completed load when tasks have only two possible sizes. In terms of latency, only algorithm LIS is competitive, when $s \geq \rho$, which might not be very surprising since algorithm LIS gives priority to the tasks that have been waiting the longest in the system. Another interesting observation is that algorithms LPT and SPT become 1-competitive as soon as $s \geq \rho$, both in terms of completed and pending load, whereas LIS and SIS require greater speedup to achieve this.

This is the first thorough and rigorous online analysis of these popular scheduling algorithms in a fault-prone setting. In some sense, our results demonstrate in a clear way the differences between two classes of policies: the ones that give priority based on the *arrival time* of the tasks in the system (LIS and SIS) and the ones that give priority based on the required *processing time* of the tasks (LPT and SPT). Observe that different algorithms scale differently with respect to the speedup, in the sense that with the increase of the machine speed the competitive performance of each algorithm changes in a different way.

Related work. We relate our work to the online version of the *bin packing* problem [15], where the objects to be packed are the tasks and the bins are the time periods between two consecutive failures of the machine (i.e., *alive* intervals). Over the years, extensive research on this problem has been done, some of which we consider related to ours. For example, Johnson et al. [16] analyze the worst-case performance of two simple algorithms (Best Fit and Next Fit) for the bin packing problem, giving upper bounds on the number of bins needed (corresponding to the completed time in our work). Epstein et al. [17] (see also [15]) considered online bin packing with resource augmentation in the size of the bins (corresponding to the length of alive intervals in our work). Observe that the essential difference of the online bin packing problem with the one that we are looking at in this work, is that in our system the bins and their sizes (corresponding to the machine's alive intervals) are unknown. Boyar and Ellen [18] have looked into a problem similar to both the online bin packing problem and ours, considering job scheduling in the grid. The main difference with our setting is that they consider several machines (or processors), but mainly the fact that the arriving items are processors with limited memory capacities and there is a fixed amount of jobs in the system that must be completed. They also use fixed job sizes and achieve lower and upper bounds that only depend on the fraction of such jobs in the system.

Another related problem is packet scheduling in a link. Andrews and Zhang [19] consider online packet scheduling over a wireless channel whose rate varies dynamically, and perform worst-case analysis regarding both the channel conditions and the packet arrivals. We can also directly relate our work to research done on machine scheduling with availability constraints (e.g., [20,21]). One of the most important results in that area