# Ordonnancement en ligne pour les machines parallèles

Elli Zavou

# *Ordonnancement en ligne pour les machines parallèles*

Elli Zavou [†]

*Univ. Lyon, Inria, INSA Lyon, CITI, F-69621 Villeurbanne, France*

L'exécution stable des tâches dans les machines parallèles est très importante. Par leur nature dynamique, ces systèmes doivent faire face à un défi de taille : ils doivent pouvoir répondre aux continuelles requêtes des utilisateurs, qui peuvent requérir des traitements différenciés. De plus, ces requêtes peuvent subir des erreurs imprévisibles, produites soit par un équipement malintentionnée, soit par un taux d'arrivées trop élevé. La consommation d'énergie induite peut également s'avérer importante, ce qui présente un autre défi.

Dans ce papier nous considérons ces deux défis et conduisons une analyse compétitive au pire cas des performances d'algorithmes déterministes en ligne. Nous supposons également une sorte d'augmentation des ressources, d'accélération de la machine, qui caractérise la consommation d'énergie du système. Pour les mesures de performances, nous utilisons la charge complète, la charge en attente, ainsi que le ratio de la latence par rapport aux tâches réalisées. Nous montrons qu'il existe un seuil d'accélération en dessous duquel aucune compétitivité ne peut être atteinte par les algorithmes déterministes, même dans le cas d'une seule machine, et au delà duquel nous analysons les performances des algorithmes les plus utilisés et proposons de nouveaux algorithmes que nous démontrerons comme étant optimaux.

**Mots-clefs :** ordonnancement des tâches, algorithmes en ligne, tolérance aux pannes, augmentation des ressources

## 1 Introduction

**Motivation.** The fast development of computing systems as well as the increase of computationally-intensive demands from the users, are just two of the triggers that have led to the development of multicore-based parallel machines, internet-based computing platforms and co-operational distributed systems. Apart from the dynamic requests from the users (*tasks arrivals*) that may have different computational requirements (*task sizes*), these systems often suffer from unpredictable processor failures (*machine crashes and restarts*), either malicious or due to overload. In this work we choose to consider *speed augmentation* in order to overcome these challenges, increasing the computational power of the machines to a speedup $s \geq 1$. Under this speedup, the machines execute a task $s$ times faster than its baseline execution time (size). It affects however, the power consumption of the system. We are hence interested in the trade-off between the speedup and the guaranteed performance of the scheduling algorithms. Our main goal is to achieve reliable and stable computations while keeping the energy consumption of the system to a minimum.

**Model.** We consider a system of $m$ homogeneous, fault-prone machines, with unique ids from the set $[m] = \{1, 2, \ldots, m\}$. We assume that they have access to a shared object, called *Repository*, which represents the interface of the system with the users. The users submit tasks to the repository, the machines decide in a parallel way which tasks they schedule and after each successful execution they notify the repository of their completion. The users are then notified accordingly. A task $\tau$ has an arrival time $\alpha(\tau)$ and a size $c(\tau) \in [c_{min}, c_{max}]$, where $c_{min}$ and $c_{max}$ the smallest and largest size possible, and $\rho = c_{max}/c_{min}$ the size ratio. Note that the size of a task represents the time it needs to be executed by a machine running with no speedup, i.e., $s = 1$, and that task executions are *atomic*, meaning that preemption and migration are not allowed. These task arrivals are represented by an infinite *arrival pattern A*, and the number of task sizes considered is denoted by parameter $k$. Furthermore, the machines may suffer from crashes and restarts,

[†]This research formed part of my PhD studies in IMDEA Networks and the University of Carlos III of Madrid [Zav16]. I was partially supported by the FPU12/00505 Grant from the Spanish Ministry of Education, Culture and Sports (MECD).

which are represented by an infinite *failure pattern* $E$. The task that is being executed by the machine that is crashed at a certain instant is not completed and has to be re-scheduled eventually. An adversarial entity is assumed to have control of both $A$ and $E$ in such a way that it gives worst-case scenarios, however assuming that at every point in time at least one machine will be available (not crashed). Due to the three main parameters of the model : *number of machines*, $m$, *amount of speedup*, $s$, and *number of different task sizes*, $k$, we denote it by $M\langle m,s,k\rangle$.

We focus on three evaluation metrics embodying the machine utilization with *completed load*, the buffering with *pending load*, and the user fairness of the system with *latency*. The completed load is defined as $C_t^s(\text{ALG},A,E) = \sum_{w\in N_t^s(\text{ALG},A,E)} c(w)$, the pending load as $P_t^s(\text{ALG},A,E) = \sum_{w\in Q_t^s(\text{ALG},A,E)} c(w)$ and the latency as $L_t^s(\text{ALG},A,E) = max\{f(w)-\alpha(w) : \forall w \in N_t^s(\text{ALG},A,E), t-\alpha(w) : \forall w \in Q_t^s(\text{ALG},A,E)\}$, where $N$ and $Q$ are the sets of completed and pending tasks respectively and $f(w)$ is the completion time of task $w$. Note that, computing the optimal schedule for the three measures offline (knowing a priori patterns $A$ and $E$) is an NP-hard problem [Zav16].

Since the scheduling decisions are to be made in a continuous manner and without knowledge of any future information, we see the problem as an *online* scheduling problem [PST04] and perform *asymptotic competitive analysis* [BEY05, VS02] to evaluate the performance of deterministic and work-conserving online algorithms, under worst-case scenarios. The *asymptotic performance ratio* that corresponds to the analysis, is the long-term competitive ratio over the sets of arrival and error patterns, $\mathcal{A}$ and $\mathcal{E}$, against any algorithm in the set of algorithms $\mathcal{X}$ that solves the scheduling problem. Note here that we only consider the combinations of patterns for which the completed load of any algorithm $X \in \mathcal{X}$ goes to infinity, i.e., $\lim_{t\to\infty} C_t^s(X,A,E) = \infty$. Hence, the three performance measures are defined as :

$$\textit{Completed Load : } \mathcal{C}^s(\text{ALG},\mathcal{A},\mathcal{E}) = \inf_{A\in\mathcal{A},E\in\mathcal{E},X\in\mathcal{X}} \lim_{t\to\infty} \frac{C_t^s(\text{ALG},A,E)}{C_t^1(X,A,E)}.$$

$$\textit{Pending Load : } \mathcal{P}^s(\text{ALG},\mathcal{A},\mathcal{E}) = \sup_{A\in\mathcal{A},E\in\mathcal{E},X\in\mathcal{X}} \lim_{t\to\infty} \frac{P_t^s(\text{ALG},A,E)}{P_t^1(X,A,E)}.$$

$$\textit{Latency : } \mathcal{L}^s(\text{ALG},\mathcal{A},\mathcal{E}) = \sup_{A\in\mathcal{A},E\in\mathcal{E},X\in\mathcal{X}} \lim_{t\to\infty} \frac{L_t^s(\text{ALG},A,E)}{L_t^1(X,A,E)}.$$

**Contributions.** This work presents some of the most important results of my thesis [Zav16], focusing on deterministic, work-conserving algorithms. It shows some general results that hold for all online deterministic scheduling algorithms and then focuses on the performance of four popular scheduling algorithms, analyzing their fault-tolerant properties under the worst-case scenarios created by the adversarial entity. The algorithms are : the *Longest In System* (LIS), which schedules the task that has been waiting the longest in the repository, the *Shortest In System* (SIS), which schedules the task that has arrived latest, the *Largest Processing Time* (LPT), which schedules the pending task of the biggest size, and the *Shortest Processing Time* (SPT), which schedules the pending task of the smallest size. Finally, some alternative algorithms are proposed, that achieve optimal competitiveness in the specified models.

# 2 Results

Table 1 summarizes the results obtained for any deterministic and work-conserving scheduling algorithm, as well as some more specific results for the four widely-used algorithms, in the case of a single machine.It gives an insight to the reader for the challenges of online scheduling algorithms even in the simplest model, thus providing a first idea of the limitations they will have in the case of $m$ parallel machines. Each row represents the performance of an algorithm in the specified model with respect to the three evaluation metrics, and provides the reference of the published paper where their detailed analysis can be found.

The first group of results is about all deterministic and work-conserving algorithms, where one can clearly see the limitations even in the case of one machine. In the first row, by assuming no speedup and an infinite amount of task sizes available, we show that no such algorithm can achieve any competitiveness. In the second row, we bound the amount of task sizes available to two, and we are able to show that algorithms may only achieve up to $1/2$-completed-load competitiveness. In the third row, we allow some speedup instead of bounding the number of task sizes, and show that no algorithm can be 1-completed-load competitive (cannot be optimal) if the speedup is less than the minimum of $\rho$ and $1+\gamma/\rho$, where $\gamma = \max\{\lceil\frac{\rho-s}{s-1}\rceil, 0\}$.

| Alg. | Model | Completed Load, $\mathcal{C}$ | Pending Load, $\mathcal{P}$ | Latency, $\mathcal{L}$ | Reference |
|---|---|---|---|---|---|
| **Determ.** | $M\langle 1,1,\infty\rangle$ | 0 | $\infty$ | $\infty$ | [FAGKZ16, KWZ15] |
| | $M\langle 1,1,2\rangle$ | $\leq \frac{\lfloor\rho\rfloor}{\rho+\lfloor\rho\rfloor} \approx \frac{1}{2}$ | $\infty$ | $\infty$ | [FAGK$^+$15, FAGKZ15] |
| | $M\langle 1,s<\min\{\rho,1+\gamma/\rho\},\infty\rangle$ | $<1$ | $\infty$ | $\infty$ | [FAGKZ16, KWZ15] |
| **LIS** | $M\langle 1,s<\rho,2\rangle$ | 0 | $\infty$ | $\infty$ | [FAGKZ16] |
| | $M\langle 1,s\in[\rho,1+1/\rho),\infty\rangle$ | $[1/\rho,\frac{1}{2}+\frac{1}{2\rho}]$ | $[\frac{1+\rho}{2},\rho]$ | $(0,1]$ | [FAGKZ16] |
| | $M\langle 1,s\in[\max\{\rho,1+\frac{1}{\rho}\},2),\infty\rangle$ | $[1/\rho,s/2]$ | $[\frac{s}{2(s-1)},\rho]$ | $(0,1]$ | [FAGKZ16] |
| | $M\langle 1,s\geq\max\{\rho,2\},\infty\rangle$ | 1 | 1 | $(0,1]$ | [FAGKZ16] |
| **SIS** | $M\langle 1,s<\rho,2\rangle$ | 0 | $\infty$ | $\infty$ | [FAGKZ16] |
| | $M\langle 1,s\in[\rho,1+1/\rho),\infty\rangle$ | $\frac{1}{\rho}$ | $\rho$ | $\infty$ | [FAGKZ16] |
| | $M\langle 1,s\in[1+1/\rho,1+\rho),\infty\rangle$ | $[1/\rho,s/(1+\rho)]$ | $[\frac{1}{s}+\frac{\rho}{1+\rho},\rho]$ | $\infty$ | [FAGKZ16] |
| | $M\langle 1,s\geq 1+\rho,\infty\rangle$ | 1 | 1 | $\infty$ | [FAGKZ16] |
| **LPT** | $M\langle 1,s<\rho,2\rangle$ | 0 | $\infty$ | $\infty$ | [FAGKZ16] |
| | $M\langle 1,s\geq\rho,\infty\rangle$ | 1 | 1 | $\infty$ | [FAGKZ16] |
| **SPT** | $M\langle 1,s<\rho,2\rangle$ | $[\frac{1}{2+\rho},\frac{\lfloor(s-1)\rho\rfloor+1}{\lfloor(s-1)\rho\rfloor+1+\rho}]$ | $\infty$ | $\infty$ | [FAGKZ16] |
| | $M\langle 1,s\geq\rho,\infty\rangle$ | 1 | 1 | $\infty$ | [FAGKZ16] |
| **$\gamma$-Burst** | $M\langle 1,[1+\gamma/\rho,\rho),2\rangle$ | 1 | 1 | 1 | [Zav16, KWZ15] |

**TABLE 1:** Detailed metric comparison of online scheduling algorithms for the case of a single machine. The last column provides the references where the results of the corresponding row can be found. Note that by definition, 0-completed-load competitiveness ratio equals to non-competitiveness, as opposed to the other two metrics, where non-competitiveness corresponds to an $\infty$ competitiveness ratio.

Note that, parameter $\gamma$ represents the smallest number of $c_{min}$-tasks that an algorithm running with speedup $s$ can complete, in addition to a $c_{max}$-task, in an interval of length $(\gamma+1)c_{min}$. What is more interesting, is that in all cases no algorithm can achieve competitiveness neither with respect to pending load nor latency.

Then, studying the four popular algorithms mentioned, we prove some positive results, guaranteeing *some* competitiveness for the different ranges of speedup $s$ and the number of task sizes $k$. Observe that, with the exception of SPT, none of the algorithms is competitive in any of the three metrics when $s<\rho$; algorithm SPT is competitive only in terms of completed load and only when two task sizes are considered. What is more, in terms of latency, only algorithm LIS is competitive when $s\geq\rho$. This specific result may not be surprising, since algorithm LIS gives priority to the tasks that have been waiting the longest in the repository. A rather interesting observation though, is that algorithms LPT and SPT become 1-competitive in terms of completed and pending load for $s\geq\rho$, whereas LIS and SIS require larger speedup to achieve this. We can say that these results demonstrate some differences between two classes of scheduling policies : the ones giving priority based on the task arrival time (LIS and SIS) and the ones giving priority based on the task size (LPT and SPT). Observe also, that different algorithms scale differently with respect to the speedup; with the increase of the machine speed the asymptotic competitive performance of each algorithm changes in a different way for each policy. Nonetheless, it is not easy to denote one of the four algorithms as generally better than the rest. The answer would depend on the exact model and objective.

After the limiting results of the widely-used algorithms we propose algorithm $\gamma$-Burst, for which we show optimal competitiveness in all three measures for the case of speedup $s\in[1+\gamma/\rho,\rho)$, however only for two task sizes. The idea of this algorithm is briefly described as follows : If the tasks pending are of the same size, it schedules one of them. Else, if there are at least $\gamma$ small tasks available, it schedules $\gamma$ of them and then a large task. Otherwise, it schedules one task of each size interchangeably.

Table 2 is the corresponding table for the results in the case of multiple machines. One can observe right away that we did not consider the latency competitiveness and this is because of its high complexity even in the case of one machine. Nevertheless, we have defined a type of algorithms, called GroupLIS($\beta$), for which we have proven some fundamental properties showing that they avoid redundant task executions. An algorithm is of type GroupLIS if the following three conditions hold : (1) it separates the pending tasks into

| Alg. | Model $M\langle m,s,k\rangle$ | Completed Load, $\mathcal{C}$ | Pending Load, $\mathcal{P}$ | Reference |
|---|---|---|---|---|
| **GroupLIS**($\beta$) | $M\langle m,1,1\rangle$ | 1 | 1 | [Zav16] |
| | $M\langle m,s\geq\rho,\infty\rangle$ | $[1/\rho,1]$ | $[1,\rho]$ | [Zav16] |
| | $M\langle m,s\geq 1+\rho,\infty\rangle$ | 1 | 1 | [Zav16] |
| $(m,\beta)$**-LIS** | $M\langle m,s\geq\rho,\infty\rangle$ | $[1/\rho,1]$ | $[1,\rho]$ | [FAGKZ15] |
| $\gamma$**m-Burst** | $M\langle m,s\in[1+\frac{\gamma}{\rho},\rho),2\rangle$ | 1 | 1 | [FAGKZ15] |
| $(m,\beta)$**-LAF** | $M\langle m,7/2,k\rangle$ | 1 | 1 | [FAGKZ15] |

**TABLE 2:** Detailed metric comparison of the algorithms proposed for the case of multiple machines for different ranges of speedup and number of task sizes. Again, the last column provides the reference where the results of the corresponding row can be found. Note that GroupLIS is a type of parallel algorithms and parameter $\beta$ is a constant that characterizes the corresponding algorithms.

classes according to their size, (2) it sorts the tasks in each class by their arrival time, and (3) when a class contains at least $\beta\cdot m^2$ tasks and machine $p$ decides to schedule a task from that class, then it schedules the $(p\cdot\beta m)th$ task in the row. We showed that these parallel algorithms actually become optimal for speedup $s\geq 1+\rho$.

Then, trying to find algorithms that would achieve good competitiveness with a lower speedup, we propose $(m,\beta)$-LIS, $\gamma$m-Burst and $(m,\beta)$-LAF, each with their limitations. The first two are generalizations of algorithms LIS and $\gamma$-Burst presented for the case of one machine. Algorithm $(m,\beta)$-LIS actually belongs to the GroupLIS category, so it gives the same performance guarantees. On the other hand, algorithm $\gamma$m-Burst becomes optimal with a smaller speedup but it only considers two different task sizes. Finally, algorithm $(m,\beta)$-LAF considers $k$ finite task sizes and uses an amortization approach to schedule the tasks, but it needs a relatively high speedup, though it could still be preferred depending on the value of $\rho$. In short, each machine keeps a local variable, named *total*, where it stores the total load of the completed tasks since their last restart. Each machine then schedules a task from the queue with the largest size such that it is not bigger than parameter *total* and there are at least $\beta\cdot m^2$ tasks pending in that queue.

# 3 Discussion

This paper presents only some of the results of my thesis [Zav16] focusing on the most important ones. Some interesting conclusions have been derived with respect to the efficiency of online scheduling algorithms in fault-prone parallel systems in general, as well as for some of the most popular algorithms already used in real life. There are also several questions that remain to be answered, such as the latency study for the multiple machine case. I hope that this work will give the fundamental framework from which to build much further. An interesting extension of this work could be to obtain efficiency bounds as functions of the speedup $s$ used, or try to formalize a single evaluation metric that would encompass the essence of the three we used here. Another, more practical extension of this work, could be to implement real life experiments, for example in data centers, in order to see how the average case results actually scale and whether there is any important aspect in the problem that we currently ignore.

# Références


[BEY05] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 2005.

[FAGK+15] Antonio Fernández Anta, Chryssis Georgiou, Dariusz R. Kowalski, Joerg Widmer, and Elli Zavou. Measuring the impact of adversarial errors on packet scheduling strategies. *Journal of Scheduling*, pages 1–18, 2015.

[FAGKZ15] Antonio Fernández Anta, Chryssis Georgiou, Dariusz R. Kowalski, and Elli Zavou. Online parallel scheduling of non-uniform tasks. *Theor. Comput. Sci.*, 590(C) :129–146, July 2015.

[FAGKZ16] Antonio Fernández Anta, Chryssis Georgiou, Dariusz R Kowalski, and Elli Zavou. Competitive analysis of fundamental scheduling algorithms on a fault-prone machine and the impact of resource augmentation. *Future Generation Computer Systems*, 2016.

[KWZ15] Dariusz R Kowalski, Prudence WH Wong, and Elli Zavou. Fault tolerant scheduling of non-uniform tasks under resource augmentation. In *Proceedings of the 12th Workshop on Models and Algorithms for Planning and Scheduling Problems*, pages 244–246, 2015.

[PST04] Kirk Pruhs, Jirı Sgall, and Eric Torng. Online scheduling. handbook of scheduling : Algorithms, models, and performance analysis, editor joseph yt. leung, 2004.

[VS02] Rob Van Stee. *On-line scheduling and bin packing*. PhD thesis, Universiteit Leiden, 2002.

[Zav16] Elli Zavou. *Online Scheduling in Fault-prone Systems : Performance Optimization and Energy Efficiency*. PhD thesis, Universidad Carlos III de Madrid, Spain, 2016.