Asymptotic Performance and Energy Consumption of SLACK

Anne Benoit, Louis-Claude Canon, Redouane Elghazi, Pierre-Cyrille Heam

Asymptotic Performance and Energy Consumption of SLACK ⋆⋆⋆

A. Benoit[1], L.-C. Canon[2], R. Elghazi[2], and P.-C. Héam[2]

1. LIP, ENS Lyon, France; 2. FEMTO-ST, U. Franche-Comté, France

Abstract. Scheduling n independent tasks onto m identical processors in order to minimize the makespan has been widely studied. As an alternative to classic heuristics, the SLACK algorithm groups tasks by packs of m tasks of similar execution times, and schedules first the packs with the largest differences. It turns out to be very performant in practice, but only few studies have been conducted on its theoretical properties. We derive novel analytical results for SLACK, and in particular, we study the performance of this algorithm from an asymptotical point of view, under the assumption that the execution time of the tasks follow a given probability distribution. The study is building on a comparison of the most heavily loaded machine compared to the least loaded one. Furthermore, we extend the results when the objective is to minimize the energy consumption rather than the makespan, since reducing the energy consumption of the computing centers is an ever-growing concern for economical and ecological reasons. Finally, we perform extensive simulations to empirically assess the performance of the algorithms with both synthetic and realistic execution time distributions.

1 Introduction

The problem of minimizing the computation time when scheduling n independent tasks on m identical processors is at the basis of scheduling theory, and a building block for solving many more complicated problems, hence it remains very important even though it has already been widely studied. Using Graham's notation [15], this problem is denoted $P||C_{\max}$.

While the problem is NP-complete (equivalent to 2-partition with two processors, or 3-partition when the number of processors m is part of the input), an easy way to get efficient solutions consist in ordering the n tasks according to some criterion, and then perform a list schedule, i.e., schedule the next task of the list on the least loaded processor, hence never leaving a processor idle. A classic ordering is the one of LPT (Longest Processing Time), which orders tasks from the longest to the smallest [16]. This algorithm has proven to have good theoretical and even better practical performance. In particular, its rate of

⋆ This work has been supported by the EIPHI Graduate School (contract ANR-17-EURE-0002).

⋆⋆ The data that support the findings of this study are openly available in figshare[7].

convergence has been studied, and new results were recently established when the distribution of task costs is generated using uniform integer compositions [5].

More recently, the Slack heuristic was proposed in [9], showing promising empirical performance compared to Lpt. Its principle is based on grouping tasks of similar execution times into packs, sorting the resulting packs by non-decreasing similarity (the similarity of a pack denoting the maximum difference of execution times between its tasks), and then scheduling the tasks in the order determined by the packs, following a list schedule (assign the next task to the least loaded processor). The idea is that a single pack cannot bring the imbalance of the processors too high, and the hope is that the packs balance each other. The objective is that the tasks in the last scheduled packs are very close to each other, hence they will not create a large imbalance at the end of the schedule. While this Slack algorithm benefits from favorable empirical performance, fewer analyses have been conducted on its theoretical properties.

These heuristics were proposed in order to minimize the makespan, i.e., the maximum execution time among the processors. Another core problem consists in minimizing the *energy consumption*, as the energy consumption of current platforms is an ever-growing concern, both for economical and ecological reasons. To optimize the energy consumption, modern processors can run at different speeds, and their power consumption is then the sum of a static part (the cost for a processor to be turned on) and a dynamic part, which is a strictly convex function of the processor speed. More precisely, a processor running at speed s dissipates a power of s^α Watts, where $2 \leq \alpha \leq 3$ [2]. Hence, a higher speed allows executing a task more rapidly, but at the price of a much higher amount of energy consumed. Finding a schedule now consists in deciding on which processor to execute each task and to decide at which speed the task is executed.

Therefore, we revisit this classic problem of scheduling n independent tasks onto m identical processors, with the aim of deriving analytical results for Slack, when the goal is to minimize the makespan or the energy consumption. We study the performance of Slack from an asymptotical point of view, under the assumption that the execution times of the tasks follow a given probability distribution. The study is building on a comparison of the most heavily loaded machine compared to the least loaded one, and hence it provides interesting insights both for the study of the classic makespan objective function, and its translation to the energy consumption. The goal of this paper is therefore to answer two main questions left unresolved in the literature so far: (i) provide a theoretical study to analyze the performance of Slack, and (ii) consider the energy consumption in the theoretical and empirical analysis of the algorithms. Our main contributions are the following:

- A fundamental bound related to the result of Slack (Section 4);
- A convergence rate for the makespan of Slack when using uniform and exponential distributions, by applying the bound of Section 4 (Section 5.1);
- A general result for bounding the energy consumption (agnostic of the algorithm and the task distribution) and its application to Slack, by applying the bound of Section 4 (Section 5.2);

– Simulations for comparison with the theoretical bounds that were computed for SLACK and LPT (Section 6).

First, Section 2 summarizes the existing contributions related to either the energy minimization problem or LPT and SLACK. Section 3 presents the problems and algorithms (LPT and SLACK). Then, Section 4 presents a useful bound on the result given by SLACK. Section 5 proposes applications of this bound: theoretical asymptotic results related to the minimization of the makespan and the energy with SLACK. In the case of the energy, Section 5.2 also gives a method to derive energy related guarantees for any algorithm bounded similarly to SLACK in Section 4. Section 6 presents the experimental results of the empirical study of LPT and SLACK. Finally, Section 7 concludes.

2 Related work

Lowering the energy consumption of computational tasks has been widely studied in the last decades, be it in the context of High Performance Computing or in other contexts, such as Cloud Computing. Many models have been proposed for the energy consumption of CPUs. For instance, the energy consumption is scaling quadratically with the speed of the CPU in [22], and there is a focus on the online evaluation of the expected idle time. In [23], the only assumption is that the energy consumption is a convex function of the speed of the CPU, and clairvoyant online and offline solutions are proposed to the problem. The heuristics presented in these two articles are then evaluated, either empirically in [22], or with approximation ratios in [23]. In our work, we explore another way of evaluating algorithms, following the remark that with large systems, stochastic asymptotic results should be relevant.

Recent surveys such as [10] and [21] compile various techniques used for energy-efficient computing, including scheduling techniques. These techniques may use either Dynamic Voltage and Frequency Scaling (DVFS), as in [17], where the frequency (and hence the speed) of processors may be chosen, or Dynamic Power Management (DPM) as in [4]. These studies propose algorithms, but they mainly focus on an empirical evaluation of these algorithms, without theoretical study.

As for scheduling algorithms that have low complexities (and therefore low energy consumption), LPT has been a well known algorithm for decades and is known to provide good theoretical and practical performance while keeping a low time complexity in $O(n \log n)$ [16]. A more recent algorithm, SLACK, also remains with an $O(n \log n)$ time complexity, while providing results that are sometimes better than LPT [9,5].

There are multiple results about the asymptotic behavior of LPT under different assumptions. Frenk and Rhinnooy Kan [14] and Coffman et al. [8] study the difference between LPT and the optimal solution in the case where the execution times of the tasks follow a probability distribution of cumulative distribution function of the form $F(x) = x^\alpha$, where $0 < \alpha < +\infty$. Loulou [18] and Piersma and Romeijn [19] do not look at specific distributions, but instead they

study LPT under the assumption that the execution times are independent and identically distributed random variables. More recently, Benoit et al. [5] studied the asymptotic optimality of SLACK and LPT under the assumption that the execution times are generated using a distribution called the uniform integer composition.

3 Framework

The $P||C_{\max}$ problem is a classic scheduling problem, where n tasks have to be scheduled on m identical machines, with the objective function of makespan minimization, i.e., minimize the execution time of the machine that completes last ($C_{\max}$). There are no constraints on tasks, which can be assigned to any machine in any order. Each task has a number of operations to perform, that we call its work and denote by w_i, and the time to execute the task is usually $t_i = w_i$, assuming that the machine executes one operation per time unit (speed $s = 1$). The problem complexity is well known, and in particular the associated decision problem is NP-complete as soon as $m \geq 2$.

List scheduling and LPT. In order to solve this $P||C_{\max}$ problem, a simple but effective heuristic algorithm consists in never letting a machine idle, i.e., as soon as a task completes on a machine, a new task is assigned to this machine. This is called *list scheduling*, and it can be implemented as in Algorithm 1, by keeping the load of each machine in a vector $\overrightarrow{W}$ of length m initialized to $(0, 0, \ldots, 0)$. For each task, we assign it to the currently least loaded machine, and the makespan is the maximum value of the vector $\overrightarrow{W}$ at the end of the execution. Any list schedule (whatever the order of tasks) is know to be a $(2-\frac{1}{m})$-approximation algorithm [16]. A variant of the List Scheduling heuristic consists in first sorting the list L by non-increasing task works, and it is called *Longest-Processing-Time-first* (LPT for short). This can be used if all tasks are known beforehand (offline scheduling), and it improves the approximation ratio of the algorithm to $(\frac{4}{3} - \frac{1}{3m})$ [16].

Slack. In this paper, we mainly focus on the SLACK algorithm, that was introduced in [9] and consists in applying the List Scheduling heuristic with a particular pretreatment on the list of tasks, as detailed in Algorithm 2. We first

Algorithm 1 ListScheduling(L, m)

Require: List L of n positive floats (task works); Number of processors m.
1: Let $\overrightarrow{W}$ be a vector of length m initialized to $\overrightarrow{W} = (0, 0, \ldots, 0)$;
2: **for** $w \in L$ in the order they appear in the list **do**
3: Let j be the index of a minimal element of $\overrightarrow{W}$;
4: $\overrightarrow{W}[j] = \overrightarrow{W}[j] + w$;
5: **end for**
6: **return** $\overrightarrow{W}$;

Algorithm 2 SLACK (L, m)

Require: List L of n positive floats (task works); Number of processors $m \leq n$.
1: Add $(-n \mod m)$ elements of work 0 at the end of L;
2: $r = n + (-n \mod m)$;
3: $L' = [x_1, \ldots, x_r]$ is obtained by sorting L non-increasingly;
4: **for** $0 \leq i \leq \frac{r}{m} - 1$ **do**
5: $\quad K_i = [x_{im+1}, x_{im+2}, \ldots, x_{im+m}]$;
6: $\quad \alpha_i = x_{im+1} - x_{im+m}$;
7: **end for**
8: Let $H = [\alpha_{i_1}, \ldots, \alpha_{i_{\frac{r}{m}}}] = [\beta_1, \ldots, \beta_{\frac{r}{m}}]$ be a non-increasing sequencing of the α_i's; L_{SLACK} is obtained by concatenating the K_i's in the same order as the α's in H.
9: $\overrightarrow{W}$ =ListScheduling(L_{SLACK}, m);

fill the list L to have a number of elements r that is a multiple of m, by adding dummy tasks of work 0. Then, tasks are sorted by non-increasing works and grouped by packs of m tasks, and then the packs are themselves sorted by non-increasing difference between the work of the longest task of the pack and the smallest one (α_i's). These differences are denoted β_k, where $\beta_1 \geq \beta_2 \ldots \geq \beta_{r/m}$. They correspond to the sorted α_i's.

Let us denote by $c_i(j)$ the load of processor j after $i \times m$ tasks (i.e., the i first packs) have been scheduled. Hence, $c_i(j) = \overrightarrow{W}[j]$ after $i \times m$ steps of the loop line 2 of Algorithm 1. One has for instance $c_0(j) = 0$ for all j (initial load), and then at each iteration i, we schedule one more pack with m tasks. We then define $\delta_i = \max_{0 \leq j,j' < m}(|c_i(j) - c_i(j')|)$, which is the maximum difference of load between two processors after iteration i.

Note that these values β_i and δ_i can be extended to any list algorithm, in particular LPT, by simply considering the list of a tasks as a succession of $\frac{n}{m}$ packs.

From makespan to energy consumption. When the goal is to minimize the energy consumption, we further consider that the frequency of the processors can be scaled using DVFS (Dynamic Voltage and Frequency Scaling). Hence, these processors have a static power P_{stat}, and can be operated at any speed (or frequency) $s \in \mathbb{R}_+^*$ [3], while we assumed so far that $s = 1$.

The execution time of task T_i at speed s then becomes $t_{i,s} = \frac{w_i}{s}$. In terms of energy consumption, there is a static part, which corresponds to the power consumed when the m processors are turned on, during a time $C_{\max}$, hence a total of $m \times C_{\max} \times P_{stat}$. For each task T_i, there is also a dynamic energy consumption, directly related to the speed s at which the processor operates the task. Using a general model, the dynamic energy consumption is $t_{i,s} \times s^\alpha$ [2], where $\alpha > 1$ (in general, $2 \leq \alpha \leq 3$). Finally, the total energy consumption of a schedule of length $C_{\max}$, where T_i is operated at speed s_i, is:

$$E = m \times C_{\max} \times P_{stat} + \sum_{i=1}^{n} t_{i,s_i} \times s_i^\alpha.$$

Table 1. Main Notations

Symbol	Definition
m	number of processor
n	number of tasks
$\{T_1, \ldots, T_n\}$	the n tasks
w_i	work of T_i (corresponding to the number of operations required by the task)
$t_i = w_i$	the execution time of T_i at speed 1
$t_{i,s} = \frac{w_i}{s}$	execution time of T_i at speed s
$m \times C_{\max} \times P_{stat}$	static energy consumption for a duration $C_{\max}$
$t_{i,s} \times s^\alpha$	dynamic energy consumption of T_i at speed s
δ_i	largest difference between the total execution times of two processors after having processed $i \times m$ tasks (first i packs)
β_i	largest difference between the execution time of any two tasks in pack i
$c_i(j)$	total execution time of processor j after $i \times m$ tasks have been scheduled (first i packs)
$W_j = \sum_{alloc(i)=j} w_i$	total work (number of operations) on processor j; $alloc(i)$ is the processor on which T_i is allocated
$W_{\max} = \max_{1 \le j \le m} W_j$	maximal number of operations allocated to a processor
$W = \sum_{1 \le i \le n} w_i$	total number of operations to perform
$\overrightarrow{W} = (W_1, \ldots, W_m)$	the $\rightarrow$ notation is used for m-length vectors (not only for W)
$\|\overrightarrow{x}\|_\alpha = \sqrt[\alpha]{\sum x_i^\alpha}$	classic α-norm of a vector

For convenience, the main notations are summarized in Table 1.

4 A Bound for SLACK

This section is dedicated to proving a fundamental bound related to Slack.

Let $\mathcal{X}$ be a distribution with positive values. We denote by $C(n, m, \mathcal{X})$ the random variable of the makespan returned by the Slack algorithm on m processors on a list of n tasks that are independent random variables of distribution $\mathcal{X}$. Let $X_1, \ldots, X_n$ be n independent random variables distributed according to $\mathcal{X}$. Let $X_{1:n} \le X_{2:n} \le \ldots \le X_{n:n}$ be associated order statistics. Particularly $X_{1:n}$ is the minimum of the X_i's and $X_{n:n}$ the maximum. Let $D_i = (X_{i:n} - X_{i-1:n})$ for every $1 \le i \le n$, with the convention $X_{0:n} = 0$. The D_i's are classically called spacings of adjacent order statistics. Let $\Delta_{\mathcal{X},n}$ be the random variable of the maximal value of D_i's, that is the maximal difference between two consecutive $X_{i:n}$ (and between 0 and $X_{1:n}$).

Theorem 1. *When using Slack (Algorithm 2),* $\max_{1 \le i,j \le m} (W_i - W_j) \le m\Delta_{\mathcal{X},n}$.

We provide here a sketch of the proof (see the companion research report [6] for all the complete proofs).

Proof (Sketch of proof). The proof is decomposed into a sequence of lemmas, aiming at proving that for every j, $\delta_{j+1} \leq \max(\beta_{j+1}, \delta_j)$. This inequality means that the difference between the most loaded processor and the least loaded processor after j rounds of task affectation is bounded by either the difference after the previous round, or the maximum difference between the tasks handled at this round of affectation. It is proved by disjunctive elimination depending on the number of tasks allocated at the current step to the processor maximizing c_{j+1} (i.e., maximizing the total execution time after step j).

This result is then used for an induction on j to prove that $\max_{1 \leq i,j \leq m}(W_i - W_j) \leq \beta_1$. This result means that at the end of the execution of SLACK, the load difference between the most and the least loaded processors is bounded by β_1, the difference between the largest and the smallest tasks of the first round of task affectation. Finally, we use the fact that $\beta_1 \leq (m-1) \times \Delta_{\mathcal{X},n}$ to derive the desired result. □

5 Convergence Speed of SLACK

In this section, we use the fundamental bound found in Section 4 to derive asymptotic results on the optimality of SLACK, first in terms of makespan in Section 5.1, and then in terms of energy consumption in Section 5.2.

5.1 Convergence of the Makespan

This section is dedicated to prove asymptotic results on the optimality of SLACK. The following main result is a direct application of Theorem 1:

Proposition 1. *The makespan of SLACK differs from the optimal one by at most $m\Delta_{\mathcal{X},n}$:*

$$0 \leq C(n, m, \mathcal{X}) - OPT \leq m\Delta_{\mathcal{X},n}.$$

Now, we will use known results on order spacings to obtain convergence results for SLACK. It is proved in [20, corollary 1.4], [1, Section 3] that

$$\mathbb{E}\left(\Delta_{\mathcal{U}[0,1],n}\right) \sim \frac{\ln n}{n+1}, \tag{1}$$

where $\mathcal{U}[0,1]$ is the uniform distribution between 0 and 1.

From Proposition 1 and Equation (1), one has the following result, proving that for a fixed m, the SLACK algorithm provides a scheduling that converges in expectation to the optimal (for the makespan):

Corollary 1. *For any fixed $m \geq 2$,*

$$0 \leq \mathbb{E}\left(C(n, m, \mathcal{U}[0,1])\right) - \mathbb{E}\left(OPT\right) = O\left(m\frac{\ln n}{n+1}\right).$$

As for the exponential distribution, it is shown in [12] that, almost surely,

$$\limsup_{n\to+\infty}\left(\frac{\Delta_{\mathcal{E}_1,n}}{\ln\ln n}\right)=1, \tag{2}$$

where $\mathcal{E}_1$ is the exponential distribution (with rate 1).

Using Proposition 1 and Equation (2), we then have the following result:

Corollary 2. *For any fixed $m \geq 2$, one has almost surely*

$$0 \leq \limsup_{n\to+\infty}\left(\frac{C(n,m,\mathcal{E}_1)-OPT}{m\ln\ln n}\right) \leq 1.$$

Corollary 2 does not show a convergence of the makespan of Slack to the optimal, but that, almost surely, the gap between their difference is under control since $\ln\ln n$ has a very slow growing speed.

5.2 Convergence of the Energy Consumption

Building upon the previous results bounding the δ_i's for Slack and analyzing its impact on the makespan, we now move to the problem of minimizing the total energy consumption E, where the speed of each processor can take any value in $\mathbb{R}_+^*$. The main result, stated in Theorem 2, shows how to adapt a classic scheduling algorithm (without speed and energy consideration) into an energy-oriented one. The quality of the solution is bounded by a factor depending on the maximal difference δ between the execution times of the last finishing processor and the first finishing processor.

We show in the companion research report [6] that a better solution can always be achieved by using a constant speed per processor, and by modifying the speeds so that all processors finish at the same time.

Theorem 2. *If an algorithm without speeds outputs a schedule with* $\max(W_i - W_j) = \delta$*, then we can transform it in polynomial time, with the optimal choice of speeds, into a schedule with* $E \leq (1+\frac{m\delta}{W})\mathrm{OPT}$*, where* OPT *is the minimal energy consumption that could be attained.*

Proof (Sketch of proof). We assume that the tasks are already assigned to the processors, with processor j having a total work of W_j. We write $\vec{W}$ the vector containing every W_j. In this case, we can choose optimally the speed of processor j as $s_j = \frac{W_j \sqrt[\alpha]{m\times P_{stat}}}{\left\|\vec{W}\right\|_\alpha \sqrt[\alpha]{\alpha-1}}$.

We then bound the optimal energy E_{OPT} by induction over m to show that:

$$P_{stat}^{\frac{\alpha-1}{\alpha}} \times \left[(\alpha-1)^{\frac{1}{\alpha}} + (\alpha-1)^{\frac{1-\alpha}{\alpha}}\right] \times W \leq E_{\mathrm{OPT}}.$$

We also bound the worst case for the energy of the algorithm $E_{\mathcal{A}}$:

$$E_{\mathcal{A}} \leq P_{stat}^{\frac{\alpha-1}{\alpha}} \times \left[(\alpha-1)^{\frac{1}{\alpha}} + (\alpha-1)^{\frac{1-\alpha}{\alpha}}\right] \times (W+m\delta).$$

Finally, from these two bounds, we derive the desired result. □

Proposition 2. *The energy consumption of* SLACK *differs from the optimal one by at most* $m^2\Delta_{\mathcal{X},n}\frac{\mathrm{OPT}}{W}$*:*

$$0 \leq \frac{E(n,m,\mathcal{X}) - \mathrm{OPT}}{\mathrm{OPT}} \leq \frac{m^2\Delta_{\mathcal{X},n}}{W}.$$

Analogously to Proposition 1, Proposition 2 provides asymptotic results on SLACK used for optimizing the energy consumption. It is derived directly from Theorems 1 and 2. Further results can be obtained both for the uniform distribution in Corollary 3 and for the exponential distribution in Corollary 4. Intuitively, the result shows that the relative difference between the energy provided by the adapted SLACK algorithm and the optimal energy consumption converges to 0 almost surely, when $n \to +\infty$, with a speed at least $\frac{m^2 \log n}{n^2}$ for the uniform distribution and $\frac{m^2 \log\log n}{n}$ for an exponential distribution.

It is proved in [11] that, almost surely,

$$\limsup_{n\to+\infty}\left(\frac{n\Delta_{\mathcal{U}[0,1],n} - \ln n}{2\log n}\right) = 1.$$

Corollary 3. *When using* SLACK *as a base scheduling algorithm with uniform distribution for the tasks, one has almost surely*

$$\limsup_{n\to+\infty}\left(\frac{E_{\mathrm{SLACK}}(n,m,\mathcal{U}[0,1]) - \mathrm{OPT}}{\mathrm{OPT}} \times \frac{n^2}{2(2+\ln 2)m^2\log n}\right) \leq 1.$$

The proof is available in the companion research report [6].

As proved in [12], with $\mathcal{E}_1$ the exponential distribution of rate 1, almost surely,

$$\limsup_{n\to+\infty}\left(\frac{\Delta_{\mathcal{E}_1,n}}{\ln\ln n}\right) = 1.$$

The rate λ of an exponential distribution is a scaling parameter, so almost surely

$$\limsup_{n\to+\infty}\left(\frac{\lambda\Delta_{\mathcal{E}_\lambda,n}}{\ln\ln n}\right) = 1.$$

Corollary 4. *With* SLACK *as a base scheduling algorithm with exponential distribution of rate* λ *for the tasks, for any fixed* $m \geq 2$*, one has almost surely*

$$\limsup_{n\to+\infty}\left(\frac{E_{\mathrm{SLACK}}(n,m,\mathcal{E}_\lambda) - \mathrm{OPT}}{\mathrm{OPT}} \times \frac{n}{m^2\ln\ln n}\right) \leq 1.$$

6 Simulations

We first present the simulation setting in Section 6.1, before studying the δ_j's and β_j's in Section 6.2, and the energy consumption in Section 6.3.

6.1 Experimental Setting

All the following experiments have been conducted on Python 3.8.10. Two types of instances have been used. Both instances have in common that the platform is composed of $m = 100$ processors.

Theoretical instances have been generated using the *random* package. These instances have been generated following commonly used random distributions: the uniform distribution, $\mathcal{U}[0, 1]$; the exponential distribution of rate 1, $\mathcal{E}_1$; the distribution of cumulative distribution function $F(x) = x^\alpha$ where $0 < \alpha < \infty$ [14]. These simple distributions correspond to the ones for which there exist convergence results in the literature and they cover a wide range of situations.

Realistic instances have been generated using the experimental cumulative distribution functions of actual workloads [13]. These real workloads can be found on the Parallel Workload Archive from the website `https://www.cs.huji.ac.il/labs/parallel/workload/`. We used 3 specific instances: KIT ForHLR II with 114,355 tasks; NASA Ames iPSC/860 with 18066 tasks; and San Diego Supercomputer Center (SDSC) DataStar with 84907 tasks.

6.2 Simulations: Study of δ_j and β_j

In this section, we describe the results of our simulations comparing the values of δ_j and β_j (the largest differences between the execution times of the processors and the tasks, as defined in Section 3) over the execution of Slack and Lpt.

In Figures 1 and 2, we can see the evolution of the quantities studied in Section 4 when bounding the performance of Slack. The quantities are:

- β_j the difference between the largest and the shortest task of pack j (i.e., at step j of the algorithm), it describes the imbalance between consecutive tasks during the execution of the algorithms;
- δ_j the difference between the largest processor and the shortest processor after step j of the algorithm (i.e., after allocating $j \times m$ tasks), it describes the imbalance between processors during the execution of the algorithms.

With these experiments, we can both investigate the relation we stated in Section 4, and investigate the unexplained "wave pattern" presented in [5].

In the case of tasks drawn through a uniform distribution with Figure 1, we observe that δ_j, the imbalance between processors, alternates between high and low values, in a sort of wave pattern. With a new representation of the pattern, we now present more elements explaining it. This pattern can be explained by the fact that the imbalance created by m consecutive tasks is then canceled by the m following tasks, as they have similar relative differences. Once the imbalance on the processors have decreased, the next m tasks will restore a new but smaller imbalance.

For most other distributions, on Figure 2, Slack and Lpt perform similarly in terms of makespan, which is characterized by the last value $\delta_{\frac{n}{m}}$. Out of our six

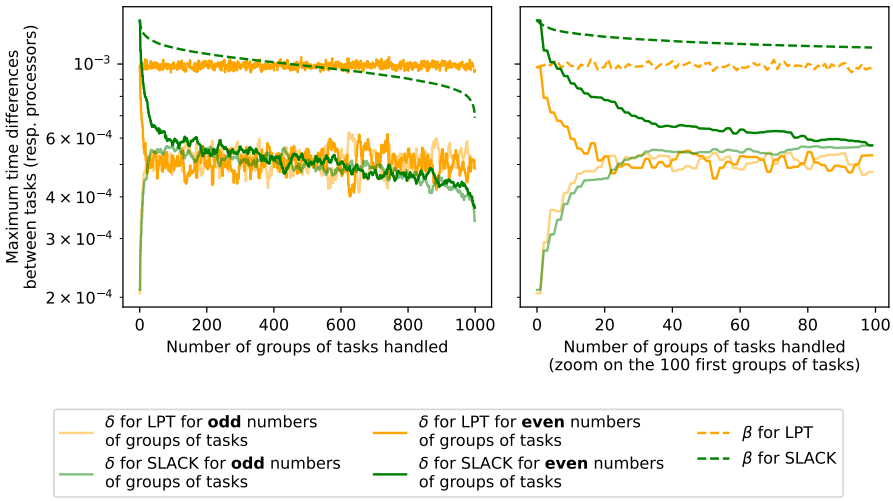

Fig. 1. Evolution of δ_j and β_j (as defined in Section 3) during the execution of SLACK and LPT with the uniform distribution $\mathcal{U}[0,1]$ for the tasks. Each execution is done with $m = 100$ processors and $n = 100\,000$ tasks. The right graph is a zoomed version of the 100 first values of δ_j and β_j. Each point represents the average value of δ_j (resp. β_j) over 30 executions.

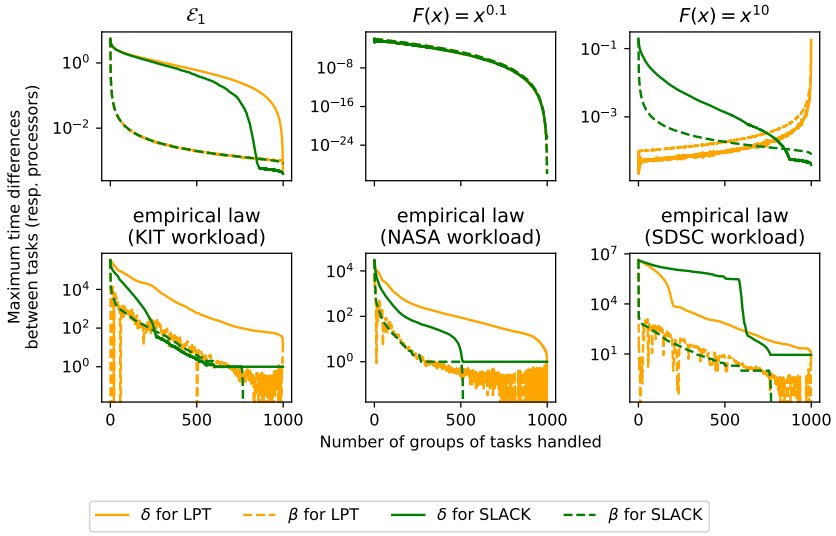

Fig. 2. Evolution of δ_j and β_j (as defined in Section 3) during the execution of SLACK and LPT with various probability distributions for the tasks. Each execution is done with $m = 100$ processors and $n = 100\,000$ tasks. Each point represents the average value of δ_j (resp. β_j) over 30 executions.

examples, the only distribution for which Slack performs significantly better than Lpt is the distribution with cumulative distribution function $F(x) = x^{10}$, namely the one for which there are a few small tasks but many large ones.

A closer look at the evolution of δ_j and β_j gives more insights about the differences of execution between Slack and Lpt, and allows us to understand why Slack performs better than Lpt in some cases. Generally speaking, Slack balances the different processors more quickly than Lpt, and then keeps them balanced. In the specific case of $F(x) = x^{10}$, Lpt performs significantly worse than Slack because there is a high density of big tasks, and a low density of small tasks. It means that the big tasks are easy to balance whereas the small tasks are very different from each other. Lpt finishes its execution with small tasks that have a very high difference β_j, whereas Slack is able to balance the processors using big tasks.

6.3 Simulations: Energy minimization

In this section, we describe the results of the simulations, evaluating the energy consumed by the schedules of the algorithms derived from Lpt and Slack (as defined in Section 3).

We normalize the energy E because the value of W can vary depending on the instance. Instead, we consider the relative difference between the energy found by the algorithm and a lower bound on OPT, i.e., $\frac{E - E_{\frac{W}{m},\dots,\frac{W}{m}}}{E_{\frac{W}{m},\dots,\frac{W}{m}}}$. We have shown in the proof of Theorem 2 that $E_{\frac{W}{m},\dots,\frac{W}{m}}$ was indeed a lower bound on OPT.

The main conclusion that we can get from Figures 3 and 4 is that Lpt and Slack both perform very well on all created instances, both theoretical and realistic. The schedule that the two algorithms output is at most a few percents away from the optimal for very small instances, and the room for improvement rapidly decreases to less than $10^{-8}\%$ for larger instances. It can be noted that Slack performs better than Lpt on average, even if they are both near optimal.

7 Conclusion

This paper proposes a bound for Slack, a recent and efficient heuristic for scheduling independent tasks on homogeneous machines, from which two asymptotic results are derived: on the makespan, with either uniformly or exponentially distributed task costs; and on the energy consumption, thanks to a general mechanism that can adapt algorithms for the makespan to this criterion.

References

1. I. Bairamov, A. Berred, and A. Stepanov. Limit results for ordered uniform spacings. *Statistical Papers*, 51(1):227–240, 2010.
2. Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Trans. Embed. Comput. Syst.*, 15(1), January 2016.

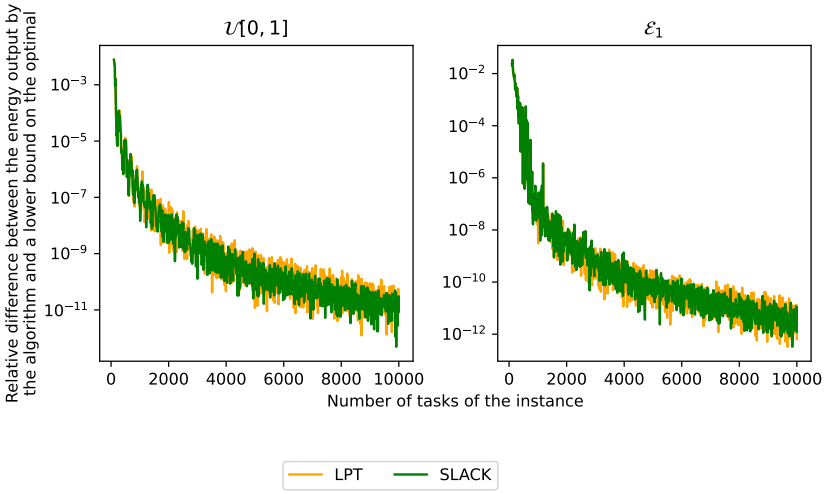

Fig. 3. Relative difference between the energy found by SLACK or LPT with the speed strategy described in Theorem 2 and a lower bound on OPT, with various theoretical probability distributions for the tasks. Each execution is done with $m = 100$ processors. Each point represents the average value of energy over 30 executions.

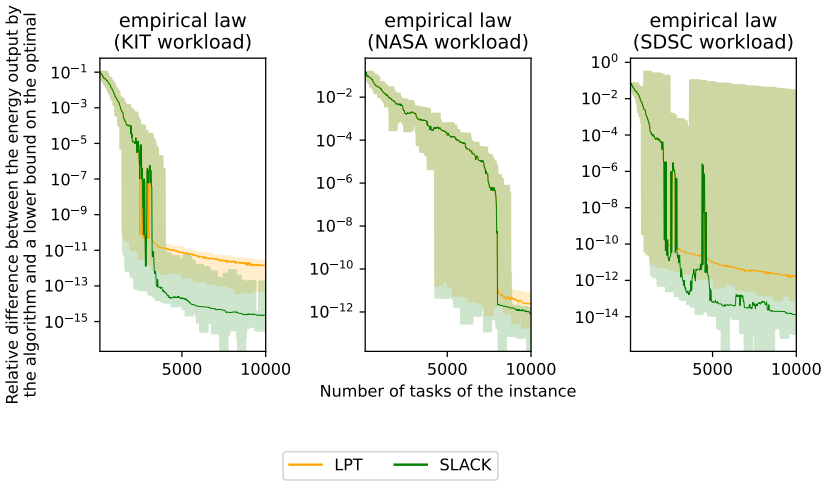

Fig. 4. Relative difference between the energy found by SLACK or LPT with the speed strategy described in Theorem 2 and a lower bound on OPT, with various empirical probability distributions for the tasks. For each number of tasks, the execution is repeated 30 times with $m = 100$ processors. The thick lines represent the moving median, while the ribbons extend to the moving minimum and maximum over 45 values.

3. Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1):1–39, 2007.
4. Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Trans. on very large scale integration (VLSI) systems*, 8(3):299–316, 2000.
5. A. Benoit, L.-C. Canon, R. Elghazi, and P.-C. Héam. Update on the Asymptotic Optimality of LPT. In *Euro-Par 2021: Parallel Processing*, pages 55–69, 2021.
6. A. Benoit, L.-C. Canon, R. Elghazi, and P.-C. Héam. Asymptotic Performance and Energy Consumption of SLACK. Research report 9501, Inria, 2023. Available at https://graal.ens-lyon.fr/~abenoit/papers/RR-9501.pdf.
7. Anne Benoit, Louis-Claude Canon, Redouane Elghazi, and Pierre-Cyrille Héam. Artifact and instructions to generate experimental results, Jun 2023. Available at https://doi.org/10.6084/m9.figshare.23579472.
8. E. G. Coffman Jr, G. S. Lueker, and A. H. G. Rinnooy Kan. Asymptotic methods in the probabilistic analysis of sequencing and packing heuristics. *Management Science*, 34(3):266–290, 1988.
9. F. Della Croce and R. Scatamacchia. The longest processing time rule for identical parallel machines revisited. *Journal of Scheduling*, 23(2):163–176, 2020.
10. Pawel Czarnul, Jerzy Proficz, and Adam Krzywaniak. Energy-aware high-performance computing: survey of state-of-the-art tools, techniques, and environments. *Scientific Programming*, 2019, 2019.
11. L. Devroye. Laws of the Iterated Logarithm for Order Statistics of Uniform Spacings. *The Annals of Probability*, 9(5):860–867, 1981.
12. L. Devroye. The largest exponential spacing. *Utilitas Mathematica*, 25:303–313, 1984.
13. D.G. Feitelson, D. Tsafrir, and D. Krakov. Experience with using the parallel workloads archive. *J. of Parallel and Distributed Comp.*, 74(10):2967–2982, 2014.
14. Johannes Bartholomeus Gerardus Frenk and A. H. G. Rinnooy Kan. The rate of convergence to optimality of the LPT rule. *Discrete Applied Mathematics*, 14(2):187–197, 1986.
15. R.L. Graham, E.L. Lawler, J.K. Lenstra, and AHG R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.
16. Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.
17. X. Lin, Y. Wang, Q. Xie, and M. Pedram. Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment. *IEEE Trans. on Services Computing*, 8(2):175–186, 2014.
18. R. Loulou. Tight bounds and probabilistic analysis of two heuristics for parallel processor scheduling. *Mathematics of Operations Research*, 9(1):142–150, 1984.
19. Nanda Piersma and H Edwin Romeijn. Parallel machine scheduling: A probabilistic analysis. *Naval Research Logistics (NRL)*, 43(6):897–916, 1996.
20. Iosif Pinelis. Order statistics on the spacings between order statistics for the uniform distribution. *arXiv preprint arXiv:1909.06406*, 2019.
21. Ankit Thakkar, Kinjal Chaudhari, and Monika Shah. A comprehensive survey on energy-efficient power management techniques. *Procedia Computer Science*, 167:1189–1199, 2020.
22. Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced CPU energy. In *Mobile Computing*, pages 449–471. Springer, 1994.
23. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. of IEEE 36th annual found. of computer science*, pages 374–382, 1995.