

A log-linear $(2 + 5/6)$ -approximation algorithm for parallel machine scheduling with a single orthogonal resource^{*}

Adrian Naruszko, Bartłomiej Przybylski, and Krzysztof Rządca

Institute of Informatics, University of Warsaw
Warsaw, Poland

an371233@students.mimuw.edu.pl, {bap,krzadca}@mimuw.edu.pl

Abstract. As the gap between compute and I/O performance tends to grow, modern High-Performance Computing (HPC) architectures include a new resource type: an intermediate persistent fast memory layer, called burst buffers. This is just one of many kinds of renewable resources which are orthogonal to the processors themselves, such as network bandwidth or software licenses. Ignoring orthogonal resources while making scheduling decisions just for processors may lead to unplanned delays of jobs of which resource requirements cannot be immediately satisfied. We focus on a classic problem of makespan minimization for parallel-machine scheduling of independent sequential jobs with additional requirements on the amount of a single renewable orthogonal resource. We present an easily-implementable log-linear algorithm that we prove is $2\frac{5}{6}$ -approximation. In simulation experiments, we compare our algorithm to standard greedy list-scheduling heuristics and show that, compared to LPT, resource-based algorithms generate significantly shorter schedules.

Keywords: Parallel machines · Orthogonal resource · Burst buffers · Rigid jobs · Approximation algorithm

1 Introduction

In a simplified model of parallel-machine scheduling, independent jobs can be freely assigned to processors if only at most one job is executed by each processor at any moment. Thus, the only resource to be allocated is the set of processors. However, this simple model does not fully reflect the real-life challenges. In practice, jobs may require additional resources to be successfully executed. These resources may include—among others—fast off-node memory, power, software licenses, or network bandwidth. All of these example resources are renewable, i.e. once a job completes it returns the claimed resources to the common pool (in contrast to consumable resources such as time or fuel).

^{*} Preprint of the paper accepted at the 27th International European Conference on Parallel and Distributed Computing (Euro-Par 2021), Lisbon, Portugal, 2021, DOI: 10.1007/978-3-030-85665-6_10

Some of the resources may be orthogonal which means that they are allocated independently of other resources. For example, in standard High-Performance Computing (HPC) scheduling, node memory is not orthogonal as a job claims all the memory of the node on which it runs. On the other hand, in cloud computing, node memory is an orthogonal resource, as it is partitioned among containers or virtual machines concurrently running at the node. Thus, node memory (and also network bandwidth) is managed in a similar manner to the processors.

The results presented in this paper are directly inspired by the practical problem of parallel-machine scheduling of jobs in HPC centers with an orthogonal resource in a form of a burst buffer. A burst buffer is a fast persistent NVRAM memory layer logically placed between the operational memory of a node and the slow external file system. Burst buffers are shared by all the jobs running in a cluster and thus they are orthogonal to processors. They can be used as a cache for I/O operations or as a storage for intermediate results in scientific workflows.

The main contribution of this paper is a log-linear $2\frac{5}{6}$ -approximation algorithm for a parallel-machine scheduling problem with independent, sequential and rigid jobs, a single orthogonal resource and makespan as the objective. We thus directly improve the classic $(3 - \frac{3}{m})$ -approximation algorithm by Garey and Graham [6]. Although a $(2 + \varepsilon)$ -approximation algorithm [21] and an asymptotic FPTAS [14] are known, their time and implementation complexities are considerable. Our algorithm can be easily implemented and it runs in log-linear time. Additionally, it can be combined with known fast heuristics, thus providing good average-case results with a guarantee on the worst case.

The paper is organized as follows. In Sec. 2, we define the analysed problem and review the related work. Then, in Sec. 3 we present a $2\frac{5}{6}$ -approximation algorithm and prove its correctness. In Sec. 4 we simulate our algorithm, compare its performance to known heuristics, and discuss its possible extensions. Finally, in Sec. 5, we make general conclusions.

2 Problem definition and related work

We are given a set of m parallel identical machines, a set of n non-preemptable jobs $\mathcal{J} = \{1, 2, \dots, n\}$, and a single resource of integer capacity R . Each job $i \in \mathcal{J}$ is described by its processing time p_i and a required amount of the resource $R_i \leq R$. We use the classic rigid job model [5] in which the processing time does not depend on the amount of the resource assigned. Our aim is to minimize the time needed to process all the jobs (or, in other words, the maximum completion time). Based on the three-field notation introduced in [10] and then extended in [3], we can denote the considered problem as $P|res1..|C_{\max}$. Here, the three values after the ‘res’ keyword determine: (1) the number of resources (one in this case); (2) the total amount of each resource (arbitrary in this case); (3) the maximum resource requirement of a single job (arbitrary in this case). This problem is \mathcal{NP} -Hard as a generalization of $P||C_{\max}$ [7]. However, both the problems have been deeply analyzed in the literature. Moreover, different machine, job, and resource characteristics have been considered in the context of various objective functions.

We refer the reader to [4] for the most recent review on resource-constrained scheduling.

In case of the variant without additional resources – $P||C_{\max}$ – it was proved that the LPT strategy leads to a $(\frac{4}{3} - \frac{1}{3m})$ -approximation algorithm [9] while any arbitrary list strategy provides a $(2 - \frac{1}{m})$ -approximation [8]. A classic result on polynomial-time approximation scheme (PTAS) for $P||C_{\max}$ was presented in [11]. In general, an efficient polynomial-time approximation scheme (EPTAS) for the $Q||C_{\max}$ problem (with uniform machines) is known [12,13]. As the $P||C_{\max}$ problem is strongly \mathcal{NP} -Hard, there exists no fully polynomial-time approximation scheme unless $\mathcal{P} = \mathcal{NP}$.

When orthogonal resources are introduced, the upper-bounds increase. Given s additional resources and more than two machines, any arbitrary list strategy leads to a $(s + 2 - \frac{2s+1}{m})$ -approximation algorithm [6] and this general bound is tight for $m > 2s + 1$. For $s = 1$, i.e. in the case of a single resource, this ratio becomes $3 - \frac{3}{m}$. We also get $\lim_{m \rightarrow \infty} (3 - \frac{3}{m}) = 3$.

For the $P|res1 \cdot 1, p_i = 1|C_{\max}$ problem with unit processing times, binary resource requirements and an arbitrary amount of the resource, the optimal C_{\max} can be found in constant time [17]. The more general $P|res1 \cdot 1, r_i, p_i = 1|C_{\max}$ problem with ready times can be solved in linear time [1], while the $P2|res1 \cdot \cdot, r_i, p_i = 1|C_{\max}$ problem with just two machines and no limits on resource requirements of a single job is already \mathcal{NP} -Hard [2].

A number of heuristic and approximation algorithms has been developed for the $P|res1 \cdot \cdot |C_{\max}$ problem and its close variants. A polynomial-time $\frac{4}{3}$ -approximation algorithm for the $P|res1 \cdot \cdot, p_i = 1|C_{\max}$ problem was shown in [18]. On the other hand, a $(3.5 + \varepsilon)$ -approximation algorithm is known for the $P|res1 \cdot \cdot, Int|C_{\max}$ problem [15], i.e. for resource-dependent job processing times. In [21], a $(2 + \varepsilon)$ -approximation algorithm for the $P|res1 \cdot \cdot |C_{\max}$ problem is presented. This algorithm can be transformed into a PTAS if the number of machines or the number of different resource requirements is upper-bounded by a constant. Further, an asymptotic FPTAS for the $P|res1 \cdot \cdot |C_{\max}$ problem was shown [14]. Although the latter results have a great theoretical impact on the problem considered in this paper, the complexity of the obtained algorithms prevents them from being efficiently implemented.

3 Approximation algorithm

In this section, we present a log-linear $2\frac{5}{6}$ -approximation algorithm for the $P|res1 \cdot \cdot |C_{\max}$ problem. In order to make our reasoning easier to follow, we normalize all the resource requirements. In particular, for each job $i \in \mathcal{J}$ we use its relative resource consumption $r_i := R_i/R \in [0, 1]$. Thus, for any set of jobs $J \subseteq \mathcal{J}$ executed at the same moment it must hold that $\sum_{i \in J} r_i \leq 1$. We also denote the length of an optimal schedule with OPT .

Before we present algorithm details, we introduce some definitions and facts. We will say that job $i \in \mathcal{J}$ is *light* if $r_i \leq \frac{1}{3}$, *medium* if $\frac{1}{3} < r_i \leq \frac{1}{2}$, and *heavy* if $r_i > \frac{1}{2}$. Thus, each job falls into exactly one of the three disjoint sets, denoted

by $\mathcal{J}_{\text{light}}$, $\mathcal{J}_{\text{medium}}$, $\mathcal{J}_{\text{heavy}}$, respectively. Note that — whatever the job resource requirements are — not more than one heavy and one medium, or two medium jobs can be executed simultaneously.

Given a subset of jobs $J \subseteq \mathcal{J}$, we define its total resource consumption as $R(J) := \sum_{i \in J} r_i$. We say that a set J is *satisfied* with θ resources, if for each subset $J' \subseteq J$ such that $|J'| \leq m$ we have $R(J') \leq \theta$. In such a case, we write $R_m(J) \leq \theta$.

The set of all the scheduled jobs will be denoted by $\mathcal{J}_{\text{scheduled}}$. For a scheduled job $i \in \mathcal{J}_{\text{scheduled}}$, we denote its start and completion times by S_i and C_i , respectively. We say that job i was *executed* before moment t if $C_i \leq t$, is being executed at moment t if $S_i \leq t < C_i$, and will be executed after moment t if $S_i > t$. The set of jobs executed at moment t will be denoted by $\mathcal{J}(t)$ and the total resource consumption of jobs in set $\mathcal{J}(t)$ by $R(t)$. We present a brief summary of the notation in Tab. 1.

Table 1. Summary of the notation

Symbol	Meaning
\mathcal{J}	The set of all the jobs
$\mathcal{J}(t)$	The set of all scheduled jobs i such that $S_i \leq t < C_i$
$J \subseteq \mathcal{J}$	Subset of all the jobs
$R(J)$	Total resource requirement of the jobs in J
$R_m(J)$	Total resource requirement of m most consumable jobs in J
$R(t)$	Total resource requirement of jobs in $\mathcal{J}(t)$

Let us start with the following lemmas, which are the essence of our further reasoning. In the algorithm, we will make sure that the assumptions of these lemmas are met, so we can use them to prove the general properties of the solution.

Lemma 1. *If $R(t) \geq \frac{2}{3}$ for all $t \in [t_a, t_b)$, then $t_b - t_a \leq \frac{3}{2} \text{OPT}$.*

Proof. It holds that $\text{OPT} \geq \sum_{i \in \mathcal{J}} r_i \cdot p_i$, as the optimal schedule length is lower-bounded by a total amount of resources consumed by all jobs in time. As $R(t) \geq \frac{2}{3}$, we obtain

$$\frac{2}{3} \cdot (t_b - t_a) \leq \int_{t_a}^{t_b} R(t) dt \leq \sum_{i \in \mathcal{J}} r_i \cdot p_i \leq \text{OPT},$$

and thus $t_b - t_a \leq \frac{3}{2} \text{OPT}$.

Lemma 2. *Let $J_1, J_2 \subseteq \mathcal{J}$. If $R_m(J_1) < \theta_1$ and $R_m(J_2) < \theta_2$, then $R_m(J_1 \cup J_2) < \theta_1 + \theta_2$.*

Proof. Let us notice that, given any set of jobs J , the value of $R_m(J)$ is determined by m most resource-consuming jobs in J . Thus, the value of $R_m(J_1 \cup J_2)$ is

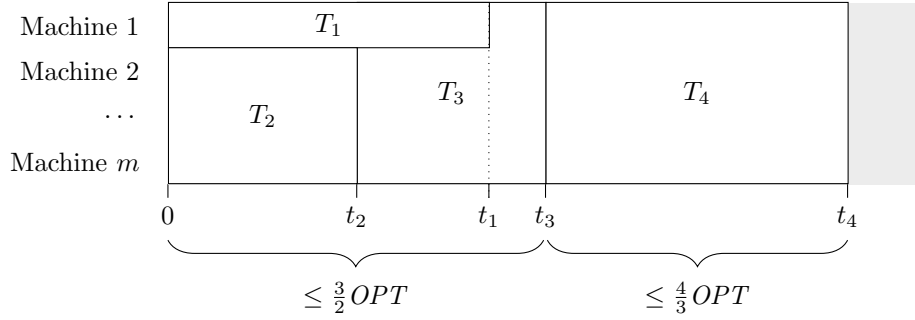


Fig. 1. The structure of a resulting schedule T .

determined by m most resource-consuming jobs from sets J_1 and J_2 . As a consequence, it holds that $R_m(J_1 \cup J_2) \leq R_m(J_1) + R_m(J_2) < \theta_1 + \theta_2$.

3.1 The idea

Our approximation algorithm consists of four separate steps that lead to a resulting schedule T . In each of the steps we generate a part of a schedule denoted by T_1 , T_2 , T_3 and T_4 , respectively. A general structure of schedule T is presented in Fig. 1. Note that some time moments are marked by t_1 , t_2 , t_3 and t_4 . These values are found while the algorithm is being executed. However, our algorithm guarantees that $0 \leq t_2 \leq t_1 \leq t_3 \leq t_4$. We also state that $t_3 \leq \frac{3}{2}OPT$ and that $t_4 - t_3 \leq \frac{4}{3}OPT$. As it is so, we get $C_{\max}(T) \leq \frac{3}{2}OPT + \frac{4}{3}OPT = 2\frac{5}{6}OPT$. The algorithm is structured as follows.

- Step 1.** Schedule all heavy jobs on the first machine in the weakly decreasing order of their resource requirements. As a consequence, it holds that $t_1 = \sum_{i \in \mathcal{J}_{\text{heavy}}} p_i$ and thus $t_1 \leq OPT$.
- Step 2.** Schedule selected light jobs starting from moment 0 in such a way that $t_2 \leq t_1$ and $R(t) \geq \frac{2}{3}$ for all $t < t_2$.
- Step 3.** Schedule all medium jobs and selected not-yet scheduled light jobs starting from moment t_2 in such a way that $t_3 \leq \frac{3}{2}OPT$.
- Step 4.** Schedule all the remaining jobs using an LPT list strategy, starting from moment t_3 . In Steps (2) and (3), we selected jobs to be scheduled in such a way that now all non-scheduled jobs form a set J such that $R_m(J) \leq 1$. Thus, $t_4 - t_3 \leq \frac{4}{3}OPT$.

As Step (1) is self-explanatory, we will not discuss it in details. However, we present its pseudocode in Alg. 1A. Steps (2) and (3) share a subroutine SCHEDULE-2/3 presented in Alg. 2. Given a set J of jobs to be scheduled and a starting time t_s , the procedure schedules some (perhaps none) of the jobs from J and returns t_c such that $R(t) \geq \frac{2}{3}$ for all $t \in [t_s, t_c)$. If $t_s = t_c$, then $[t_s, t_c) = \emptyset$ and the statement remains true. Note that the time complexity of the SCHEDULE-2/3 subroutine is $O(|J| \log |J|)$.

Algorithm 1A Approximation algorithm — Step 1 out of 4

SCHEDULE all the jobs from the $\mathcal{J}_{\text{heavy}}$ set in a weakly decreasing order of r_i , on the first machine, starting from moment 0

$$t_1 \leftarrow \sum_{i \in \mathcal{J}_{\text{heavy}}} p_i$$

Algorithm 2

procedure SCHEDULE-2/3(J, t_s)

$$t_c \leftarrow t_s$$

Sort J in a weakly decreasing order of r_i

for $i \in J$ **do**

$$t_c \leftarrow \min\{t: t \geq t_c \text{ and } R(t) < \frac{2}{3}\}$$

if job i can be feasibly scheduled in the $[t_c, t_c + p_i)$ interval **then**

SCHEDULE job i in the $[t_c, t_c + p_i)$ interval on any free machine

else break

return $\min\{t: t \geq t_c \text{ and } R(t) < \frac{2}{3}\}$

3.2 The analysis of the algorithm

In Step (2) of the algorithm, we partially fill the T_2 block of the schedule in such a way that at least $\frac{2}{3}$ of the resource is consumed at any point in the $[0, t_2)$ interval. As we expect t_2 to be less or equal to t_1 , this step does not apply if $t_1 = 0$ or, equivalently, if $\mathcal{J}_{\text{heavy}} = \emptyset$. The procedure itself is presented in Alg. 1B and its time complexity is $O(|\mathcal{J}_{\text{light}}| \log |\mathcal{J}_{\text{light}}|)$. Note that when it is finished, there might exist one or more scheduled jobs $i \in \mathcal{J}_{\text{light}}$ such that $C_i > t_2$, i.e. $\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}$ might be a non-empty set.

Proposition 1. *Let $t_1 > 0$. After Step (2) is finished, the following statements hold:*

- (a) *If $t_1 = t_2$, then $R(\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}) < \frac{1}{3}$.*
- (b) *If $t_1 > t_2$, then $R(\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}) < \frac{1}{6}$ and $R_m(\mathcal{J}_{\text{light}} \setminus \mathcal{J}_{\text{scheduled}}) < \frac{1}{3}$.*

Proof (1a). Assume that $t_1 = t_2$ and let $h \in \mathcal{J}_{\text{heavy}}$ be the last heavy job scheduled in Step (1). Note that, in particular, $C_h = t_1$. If $\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}} = \emptyset$, then obviously $R(\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}) = 0 < \frac{1}{3}$. Otherwise, each job in the $\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}$ set must have been started before t_2 . Consider a moment $t = t_2 - \varepsilon$, for an arbitrarily

Algorithm 1B Approximation algorithm — Step 2 out of 4

$$t_2 \leftarrow 0$$

if $\mathcal{J}_{\text{heavy}} \neq \emptyset$ **then**

$$t_c \leftarrow \text{SCHEDULE-2/3}(\mathcal{J}_{\text{light}}, 0)$$

$$t_2 \leftarrow \min\{t_1, t_c\}$$

if $t_1 = t_2$ **then**

UNSCHEDULE jobs $i \in \mathcal{J}_{\text{light}}$ for which $S_i \geq t_2$

small ε . The set $\mathcal{J}(t)$ consists of job h , zero or more light jobs $i \in \mathcal{J}_{\text{light}}$ for which $C_i = t_2$, and all the jobs from the $\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}$ set.

Select job $j \in \mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}$ that was scheduled as the last one. Be reminded that in Step (2), light jobs are scheduled in the decreasing order of their resource requirements. From the construction of the algorithm we conclude that $R(t) - r_j < \frac{2}{3}$. If $r_j \geq \frac{1}{6}$, then it must be the only job in the $\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}$ set, as $r_h + r_j > \frac{2}{3}$. Thus, $\frac{1}{6} \leq R(\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}) < \frac{1}{3}$. If $r_j < \frac{1}{6}$, then $R(t) - r_j - r_h < \frac{2}{3} - \frac{1}{2} = \frac{1}{6}$, so $R(t) - r_h < \frac{1}{6} + r_j < \frac{1}{3}$. Thus, $R(\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}) < \frac{1}{3}$. \square

Proof (1b). Assume that $t_1 > t_2$ and let $h \in \mathcal{J}_{\text{heavy}}$ be the heavy job executed at the moment t_2 . The value returned by the SCHEDULE-2/3 subroutine guarantees that $R(t) \geq \frac{2}{3}$ for $0 \leq t < t_2$. At the same time, it must hold that $\frac{1}{2} < R(t_2) < \frac{2}{3}$. Thus, at moment t_2 we have $R(\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}) + r_h < \frac{2}{3}$ and, as a consequence, $R(\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}) < \frac{1}{6}$.

Now, we will show that $R_m(\mathcal{J}_{\text{light}} \setminus \mathcal{J}_{\text{scheduled}}) < \frac{1}{3}$. There are two cases to be considered: either all the machines are busy at the moment t_2 or not. If not all the machines are busy, then all the light jobs were scheduled. Otherwise, any remaining light job would be scheduled on a free machine before the SCHEDULE-2/3 subroutine would end. Thus, $\mathcal{J}_{\text{light}} \setminus \mathcal{J}_{\text{scheduled}} = \emptyset$ and $R_m(\mathcal{J}_{\text{light}} \setminus \mathcal{J}_{\text{scheduled}}) = 0$.

Now, assume that all the machines are busy at the moment t_2 , i.e. $|\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}| = m - 1$. As the light jobs were scheduled in the weakly decreasing order of their resource requirements and $R(\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}) < \frac{1}{6}$, which was proven before, we have

$$\max\{r_i : i \in \mathcal{J}_{\text{light}} \setminus \mathcal{J}_{\text{scheduled}}\} \leq \min\{r_i : i \in \mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}\} < \frac{1}{6}.$$

As $R(\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}) < \frac{1}{6}$ and $|\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}| < m$, we conclude that $R_{m-1}(\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}) = R(\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}) < \frac{1}{6}$ and $R_{m-1}(\mathcal{J}_{\text{light}} \setminus \mathcal{J}_{\text{scheduled}}) \leq R_{m-1}(\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}) < \frac{1}{6}$. Finally,

$$\begin{aligned} R_m(\mathcal{J}_{\text{light}} \setminus \mathcal{J}_{\text{scheduled}}) &\leq R_{m-1}(\mathcal{J}_{\text{light}} \setminus \mathcal{J}_{\text{scheduled}}) \\ &\quad + \max\{r_i : i \in \mathcal{J}_{\text{light}} \setminus \mathcal{J}_{\text{scheduled}}\} \\ &< \frac{1}{6} + \frac{1}{6} = \frac{1}{3}. \quad \square \end{aligned}$$

Before Step (3) is started, all heavy jobs and some (perhaps none) light jobs are scheduled. At this point, we ignore all the jobs from the $\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}$ set, i.e. the scheduled light jobs for which $S_i \leq t_2 < C_i$. Being ignored, they are treated as scheduled jobs that do not occupy machines and do not use any resources. Thus, we will not schedule these jobs in Step (3). These jobs will be rescheduled in Step (4). As a consequence, at any point $t \geq t_2$ not more than a single heavy job is actually executed. In Step (3), we first schedule all the medium jobs using a standard list scheduling approach, and then, if $t_1 = t_2$, we try to schedule not-yet scheduled light jobs using the SCHEDULE-2/3 routine. This intuition is formalized in Alg. 1C. Note that the time complexity of Alg. 1C is $O(|\mathcal{J}| \log |\mathcal{J}|)$.

Proposition 2. *Let t_c and t_g be defined as in Alg. 1C. After Step (3) is finished, the following statements hold:*

Algorithm 1C Approximation algorithm — Step 3 out of 4

IGNORE all the jobs from the $\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}$ set
 Use a standard list scheduling approach to SCHEDULE all the jobs from $\mathcal{J}_{\text{medium}}$ in a weakly increasing order of r_i , starting from moment t_2
 $t_g \leftarrow \sup\{t: R(t) \geq \frac{2}{3}\}$
if $t_1 = t_2$ **then**
 $t_g \leftarrow \text{SCHEDULE-2/3}(\mathcal{J}_{\text{light}} \setminus \mathcal{J}_{\text{scheduled}}, t_g)$
 $t_c \leftarrow \max\{C_i: i \in \mathcal{J}_{\text{scheduled}}\}$
 $t_3 \leftarrow \max\{t_g, t_1\}$

- (a) If $t_c = t_1$, then $R_m(\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}}) < \frac{1}{3}$.
 (b) If $t_c > t_1 = t_2$, then $R_m(\mathcal{J}(t_g) \cup (\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}})) < \frac{2}{3}$.
 (c) If $t_c > t_1 > t_2$, then $R_m(\mathcal{J}(t_g) \cup (\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}})) < \frac{5}{6}$.

Proof (2a). If $t_c = t_1$ and $t_1 = t_2$, then no jobs were scheduled in Step (3) and thus $|\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}}| = 0$. On the other hand, if $t_c = t_1$ and $t_1 > t_2$, then all medium jobs are finished before or at t_1 . The only jobs that were not scheduled yet are light jobs. According to Prop. 1(b), we had $R_m(\mathcal{J}_{\text{light}} \setminus \mathcal{J}_{\text{scheduled}}) < \frac{1}{3}$ after Step (2), so now it must hold that $R_m(\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}}) < \frac{1}{3}$. \square

Proof (2b). Notice that $R(t_g) < \frac{2}{3}$ and thus at most one medium job is being executed at t_g . If no medium jobs are being executed at t_g , then either there were no medium jobs to be scheduled or light jobs made the value of t_g increase. In both cases, as $t_c > t_1 = t_2$, there are only light jobs being executed at t_g and only light jobs are left to be scheduled. Thus, $R_m(\mathcal{J}(t_g) \cup (\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}})) < \frac{2}{3}$. Otherwise, the value of t_g would be even larger. Notice that if exactly one medium job is being executed at t_g , then this medium job is the last one executed. Consider two cases. If not all the machines are busy at t_g , then there are no light jobs left to be scheduled and thus $\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}} = \emptyset$ and $R_m(\mathcal{J}(t_g) \cup (\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}})) = R(t_g) < \frac{2}{3}$. On the other hand, if all the machines are busy at t_g , then a medium job and $m - 1$ light jobs are executed at this moment. As the medium job has larger resource requirement than any light job, and light jobs were scheduled in a weakly decreasing order of their resource requirements, it must hold again that $R_m(\mathcal{J}(t_g) \cup (\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}})) = R(t_g) < \frac{2}{3}$. \square

Proof (2c). From the assumption that $t_c > t_1 > t_2$, we conclude that at least one medium job was scheduled in Step (3) and $t_3 \geq t_g > \max_{i \in \mathcal{J}_{\text{scheduled}}} S_i \geq t_2$. As all the light jobs for which $S_i \geq t_2$ were unscheduled in Step (2), all the light jobs for which $C_i > t_2$ were ignored, and $t_1 \neq t_2$, no light jobs are executed at moment t_g . If two non-light jobs, i and j , were executed at t_g , then it would hold that $r_i + r_j > \frac{2}{3}$ which contradicts the fact that $t_g \geq \sup\{t: R(t) \geq \frac{2}{3}\}$. Finally, if a heavy job was executed at t_g , then it would hold that $t_c = t_1$ which contradicts the assumption that $t_c > t_1$. Thus, at most one medium job can be executed at the moment t_g , and $R(t_g) \leq \frac{1}{2}$. At the same moment, according to Prop. 1(b), we had $R_m(\mathcal{J}_{\text{light}} \setminus \mathcal{J}_{\text{scheduled}}) < \frac{1}{3}$ after Step (2). Based on Lem. 2, we obtain $R_m(\mathcal{J}(t_g) \cup (\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}})) < \frac{1}{2} + \frac{1}{3} = \frac{5}{6}$. \square

Proposition 3. *It holds that $t_3 \leq \frac{3}{2}OPT$.*

Proof. It holds that $t_3 \geq t_1$. If $t_1 = t_3$, then $t_3 \leq OPT \leq \frac{3}{2}OPT$, so assume that $t_3 > t_1$. First, consider a case when $t_3 > t_1 = t_2$. It means that all medium jobs (if they exist) were scheduled starting from moment t_1 . As for any medium job $i \in \mathcal{J}_{\text{medium}}$ we have $\frac{1}{3} < r_i \leq \frac{1}{2}$, any two medium jobs can be executed in parallel, and if it is so, more than $\frac{2}{3}$ of the resource is used. In such a case, just after medium jobs are scheduled, we have $t_g = \sup\{t: R(t) \geq \frac{2}{3}\}$, and after Step (3) is finished, one has $R(t) \geq \frac{2}{3}$ for all $t \in [t_2, t_3)$. Now, be reminded about the ignored jobs from the $\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}$ set. If we reconsider them, even at the cost of exceeding the available amount of resources or the number of machines, we have $R(t) \geq \frac{2}{3}$ for all $t \in [0, t_3)$. This is enough to state that, based on Lem. 1, $t_3 \leq \frac{3}{2}OPT$, although some of the jobs scheduled before t_3 are ignored and will be rescheduled later.

Now, consider a case when $t_3 > t_1 > t_2$. As $t_1 > t_2$, no light jobs are scheduled in Step (3). This inequality implicates that at least one medium job could have been scheduled together with a heavy job in the $[t_2, t_1)$ interval. As heavy jobs are scheduled in a weakly decreasing order of the resource requirements, and medium jobs are scheduled in a weakly increasing order of the resource requirements, there are two possibilities.

If a medium job is being executed at every moment $t \in [t_2, t_1)$, then $t_g = \sup\{t: R(t) \geq \frac{2}{3}\}$ and — based on the same reasoning as in the previous case — we have $R(t) \geq \frac{2}{3}$ for all $t \in [0, t_g)$. As $t_3 = t_g$, based on Lem. 1 we obtain $t_3 \leq \frac{3}{2}OPT$.

In the second possibility, there exists a point t in the $[t_2, t_1)$ interval, for which $R(t) < \frac{2}{3}$. Consider the latest such point, i.e. $t := \sup\{t \in [t_2, t_1): R(t) < \frac{2}{3}\}$. The t value is either equal to t_1 , or to a starting time of a medium job. In both cases, no medium job i for which $S_i \geq t$ could have been scheduled earlier. Thus, in the $[t, t_g)$ interval (if non-empty) all jobs are either heavy or medium, and are scheduled on exactly 2 machines at the same time. Moreover, it would be not possible to execute three such jobs in parallel due to their resource requirements, so in the optimal schedule not more than two machines would be busy starting from point t due to the resource requirements of the medium jobs. In our case, both machines are busy in the $[t, t_g)$ interval, so it must hold that $t_g \leq OPT$. If so, then $t_3 = \max\{t_g, t_1\} \leq OPT \leq \frac{3}{2}OPT$. \square

After Step (3) is finished, we unschedule ignored jobs from the $\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}$ set and all the jobs from the $\mathcal{J}(t_g)$ set, and then we schedule all jobs that are in the updated $\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}}$ set. As we now know that $t_3 \leq \frac{3}{2}OPT$ and that $t_3 \geq t_g$, all the machines are free starting from the t_3 moment. This intuition is shown in Alg. 1D. It can be now shown that all the jobs to be scheduled are satisfied with a single unit of the resource. As it is so, all the machines can execute m jobs in parallel, whichever m jobs we choose. Thus, a schedule provided by the LPT list strategy is a $\frac{4}{3}$ -approximation solution for the $\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}}$ set.

Proposition 4. *After Step (3) is finished, it holds that*

$$R_m((\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}) \cup \mathcal{J}(t_g) \cup (\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}})) < 1.$$

Algorithm 1D Approximation algorithm — Step 4 out of 4

UNSCHEDULE all the ignored jobs from the $\mathcal{J}(t_2) \cap \mathcal{J}_{\text{light}}$ set
 UNSCHEDULE all the jobs from the $\mathcal{J}(t_g)$ set
 Use an LPT list scheduling approach to SCHEDULE all the jobs from the $\mathcal{J} \setminus \mathcal{J}_{\text{scheduled}}$ set (including just unscheduled ones), starting from moment t_3
 $t_4 \leftarrow \max\{t_3, C_{\max}\}$

Proof. The proof follows directly from Prop. 1–2 and Lem. 2.

Theorem 1. *Algorithm 1A–1D is a log-linear $2\frac{5}{6}$ -approximation algorithm for the $P|\text{res1} \cdot |C_{\max}$ problem.*

Notice that the $2\frac{5}{6}$ -approximation ratio leaves us room for immediate improvement. In fact, based on the result by Graham [9], as we can apply the LPT strategy in Step (4) without any concerns about the orthogonal resource, it must hold that $t_4 - t_3 \leq \left(\frac{4}{3} - \frac{1}{3m}\right) \cdot OPT$. Thus, Alg. 1A–1D is $\left(2\frac{5}{6} - \frac{1}{3m}\right)$ -approximation.

4 Simulations and extensions

The log-linear algorithm presented in Sec. 3 provides a schedule that is not more than $2\frac{5}{6}$ times longer than the optimal one. This is so for arbitrary independent jobs and arbitrary resource requirements. While our principal contribution is in theory, our log-linear algorithm is also easily-implementable, so in this section we evaluate our algorithm and compare its average-case performance to standard heuristics.

4.1 Compared algorithms

We will compare three variants of our algorithm against a number of greedy heuristics based on list scheduling algorithms. All the algorithms were implemented in PYTHON; we performed our experiments on an Intel Core i7-4500U CPU @ 3.00GHz with 8GB RAM.

The theoretical algorithm from Sec. 3 will be denoted by $ApAlg$. Its first extension, denoted by $ApAlg-S$, introduces an additional step of backfilling. Namely, after the $ApAlg$ algorithm is finished, we iterate over all the jobs in order of their starting times, and reschedule them so they start at the earliest moment possible. Note that this additional step never increases the starting time of any job, and thus $ApAlg-S$ is also a $2\frac{5}{6}$ -approximation algorithm. The second extension, denoted by $ApAlg-H$ is a heuristic algorithm based on $ApAlg$. In this case, the SCHEDULE-2/3 subroutine (see Alg. 2) is replaced. In $ApAlg-H$, it schedules jobs in a strict weakly decreasing order of r_i (so no job j such that $r_j < r_i$ is started before job i), regardless of whether the total resource consumption at t has exceeded $\frac{2}{3}$.

We compare the $ApAlg$, $ApAlg-S$ and $ApAlg-H$ algorithms against four list scheduling algorithms: LPT (Longest Processing Time), HRR (Highest Resource

Requirement), *LRR* (Lowest Resource Requirement), and *RAND* (Random Order). Any list scheduler starts by sorting jobs according to the chosen criterion. Then, when making a scheduling decision, it seeks for the first job on the list that can be successfully scheduled, i.e. has its resource requirements not greater than what is left given jobs being currently executed (if no such job exists, or all processors are busy, the algorithm moves to the next time moment when any job completes). Thus, the worst-case running time of the *ApAlg-S*, *LPT* and *RAND* algorithms is $O(n^2)$. The worst-case running time of the *LRR* and *HRR* algorithms is $O(n \log n)$ – these algorithms can use binary search to find the first job from the list having resource requirement not exceeding the currently available resources.

4.2 Instances

Our simulations are based on the dataset provided by the MetaCentrum Czech National Grid [19,16]. In order to avoid normalizing data from different clusters, we arbitrarily chose the cluster with the largest number of nodes (*Zapat*). The *Zapat* cluster consisted of 112 nodes, each equipped with 16 CPU cores and 134GB of RAM. This cluster was monitored between January 2013 and April 2015, which resulted in 299 628 log entries.

Each entry provides information about job processing times (p_i) and their main memory requirements (r_i). We limit ourselves to entries for which both p_i and r_i are less or equal to their respective 99th percentiles, so the data can be safely normalized. As different jobs may be executed in parallel on each node, we consider the main memory to be a single orthogonal resource. We assume that the total memory size (total amount of the resource) is equal to the maximum memory requirement in the set of all the considered job entries. Thus, we normalize all the resource requirements so $r_i \in [0, 1]$ (where 1 is the resource capacity of the simulated system). As we study the problem with sequential jobs, we also assume that each job from the trace requires a single CPU. In Fig. 2, we present how the job processing times and normalized memory requirements were distributed within the log. Most of the jobs have rather low resource requirements. In fact, the 25th, 50th and 75th percentiles of the resource requirements distribution are equal to 0.0165, 0.0169, and 0.068, respectively. We have also analysed how the values of p_i and r_i correlate to each other. As the distributions of p_i and r_i clearly are not Gaussian-like, we calculated the Spearman's correlation coefficient. This requires an additional assumption that the relation between p_i and r_i is monotonic. The calculated value is 0.14207 which suggests positive, yet not very significant correlation. This result was verified visually.

For each combination of the number of jobs $n \in \{500, 1000, 5000, 10000\}$ and the number of machines $m \in \{10, 20, 50, 100\}$, we generated 30 problem instances. The processing time and resource requirement of each job were set to the processing time and memory requirement of a job randomly chosen from the log. As the lower bound on the optimal schedule length is $L = \max\{\sum_{i \in \mathcal{J}} p_i / m, \sum_{i \in \mathcal{J}} p_i \cdot r_i\}$, we only considered instances in which $\max_{i \in \mathcal{J}} p_i < L$. This way, we omitted

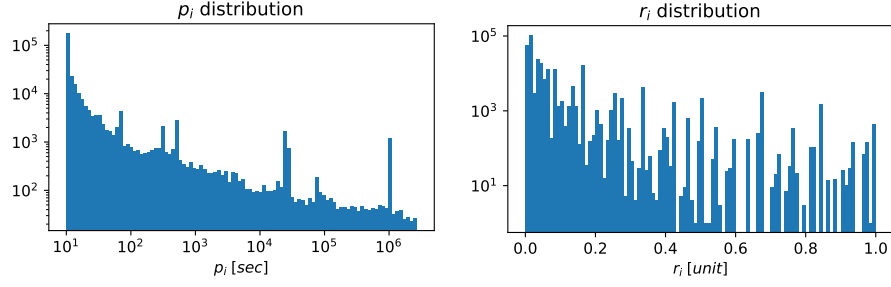


Fig. 2. Job processing time and resource requirement distribution in cluster *Zapat* (both axes limited to the 99th percentile).

(trivial) instances for which the length of the optimal schedule is determined by a single job.

4.3 Simulation results

In Fig. 3, we present the results obtained for all the algorithms and all the (n, m) combinations. We report the returned C_{\max} values as normalized by the lower-bound of the optimal schedule length, i.e. $\max\{\sum_{i \in \mathcal{J}} p_i / m, \sum_{i \in \mathcal{J}} p_i \cdot r_i\}$. For lower numbers of machines (left part of the figure) the results of *ApAlg*, *ApAlg-S*, and *ApAlg-H* algorithms are comparable. However, when the number of machines increases, the original approximation algorithm is significantly outperformed by the *ApAlg-S* and *ApAlg-H* variants. In the considered job log, the 50th percentile on the normalized resource requirement was 0.0169. We would thus expect that usually not more than 50 machines are busy in the initial part of a schedule provided by the *ApAlg* variant (due to the threshold of $\frac{2}{3}$ on a total resource consumption). In such cases, the *ApAlg-S* and *ApAlg-H* variants gain a clear advantage, as they can potentially make use of all the machines, if possible.

When the numbers of jobs and machines increase (right-bottom part of the figure), the normalized C_{\max} values decrease for all the algorithms. In such cases, a greedy heuristic, *HRR*, provides almost optimal schedules, with a maximum normalized C_{\max} value of 1.03. There might be two reasons for that. First, when the number of jobs increases and their processing times come from the same distribution, it is easier for greedy algorithms to provide better results (as the normalized C_{\max} value is relative). Second, if job resource requirements are not too small compared to the number of machines, the *HRR* produces a schedule with the resource being almost fully-utilized most of the time, compared to *LPT* which makes decisions solely based on the job length.

We also compared the runtime of the algorithms on large instances with 10000 jobs and 100 machines. As expected, log-linear *ApAlg*, *ApAlg-H*, *LRR* and *HRR* algorithms were significantly faster: their runtimes (median over 30 instances) were equal to 2.83s, 3.29s, 0.74s and 0.83s, respectively — in contrast to *ApAlg-S*, *LPT* and *RAND* algorithms with runtimes of 88.39s, 88.58s and 92.94s.

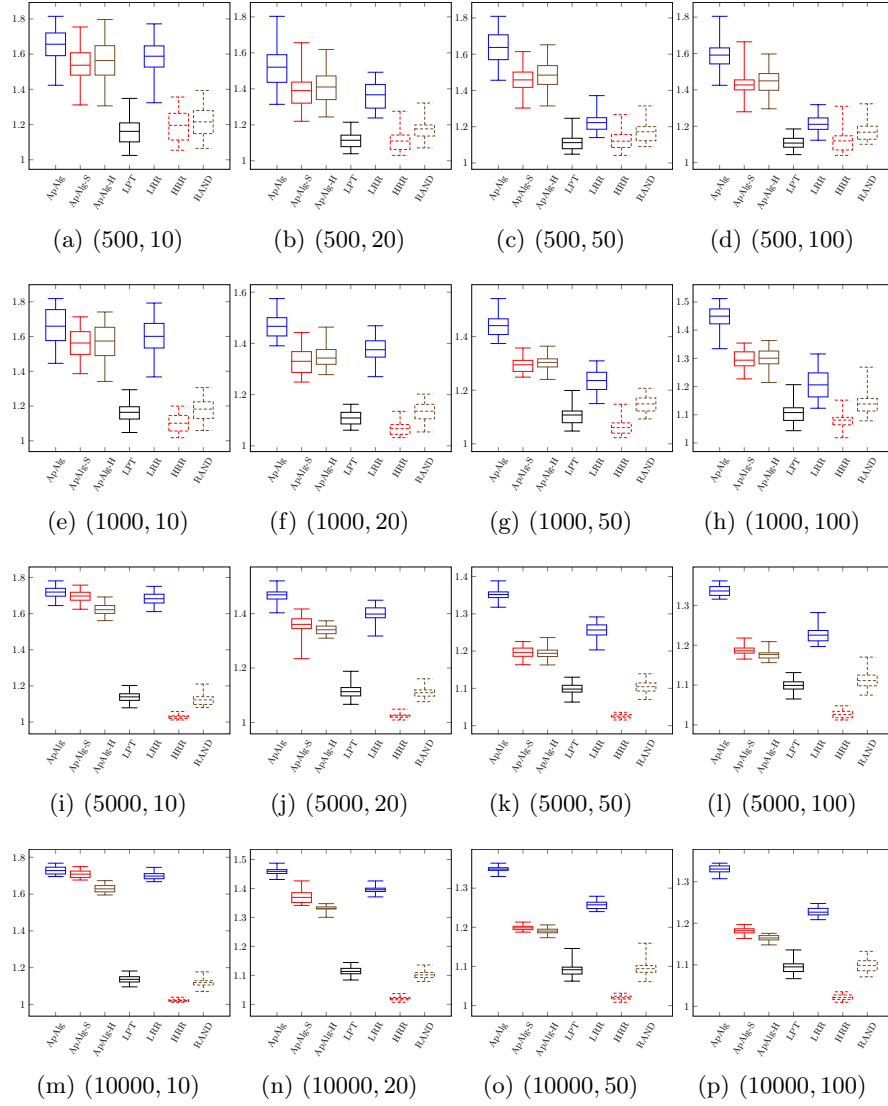


Fig. 3. The C_{\max} values normalized by the lower-bounds on the optimal schedule lengths. Captions (n, m) describe the number of jobs n and the number of machines m .

5 Conclusions

In this paper, we presented a log-linear $2\frac{5}{6}$ -approximation algorithm for the parallel-machine scheduling problem with a single orthogonal resource and makespan as the objective function. Our algorithm improves the $(3 - \frac{3}{m})$ -approximation ratio of Garey and Graham [6]. It is also considerably easier to implement than the approximation algorithms proposed by Niemeier and Wiese [21] and Jansen, Maack and Rau [14].

In the computational experiments, we compared three variants of our algorithm to four list scheduling heuristics. We used the real-life data provided by the MetaCentrum Czech National Grid. The results provided by the *HRR* (Highest Resource Requirement) list heuristic significantly outperformed all other algorithms for the considered dataset. Although the results provided by the *HRR* algorithm are promising, the approximation ratio can be improved in general. Thus, in order to provide the best results, one can combine the *HRR* algorithm with our algorithms and thus obtain good schedules with a better guarantee on their approximation ratio.

Acknowledgements and Data Availability Statement

This research is supported by a Polish National Science Center grant Opus (UMO-2017/25/B/ST6/00116). The authors would like to thank anonymous reviewers for their in-depth comments that helped to significantly improve the quality of the paper.

The datasets and code generated during and/or analyzed during the current study are available in the Figshare repository: <https://doi.org/10.6084/m9.figshare.14748267> [20].

References

1. Blazewicz, J.: Complexity of computer scheduling algorithms under resource constraints. In: Proceedings of the First Meeting AFCET-SMF on Applied Mathematics. pp. 169–178 (1978)
2. Blazewicz, J., Cellary, W., Slowinski, R., Weglarz, J.: Scheduling under resource constraints—deterministic models. JC Baltzer AG (1986)
3. Blazewicz, J., Lenstra, J., Kan, A.R.: Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics **5**(1), 11–24 (1983). [https://doi.org/10.1016/0166-218X\(83\)90012-4](https://doi.org/10.1016/0166-218X(83)90012-4)
4. Edis, E.B., Oguz, C., Ozkarahan, I.: Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. European Journal of Operational Research **230**(3), 449–463 (2013). <https://doi.org/10.1016/j.ejor.2013.02.042>
5. Feitelson, D.G.: Job scheduling in multiprogrammed parallel systems (1997)
6. Garey, M., Graham, R.: Bounds for multiprocessor scheduling with resource constraints. SIAM J. Comput. **4**, 187–200 (1975). <https://doi.org/10.1137/0204015>

7. Garey, M., Johnson, D.: Strong NP-completeness results: motivation, examples, and implications. *J. Assoc. Comput. Mach.* **25**(3), 499–508 (1978). <https://doi.org/10.1145/322077.322090>
8. Graham, R.: Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal* **45**(9), 1563–1581 (1966). <https://doi.org/10.1002/j.1538-7305.1966.tb01709.x>
9. Graham, R.: Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics* **17**(2), 416–429 (1969)
10. Graham, R., Lawler, E., Lenstra, J., Kan, A.R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Hammer, P., Johnson, E., Korte, B. (eds.) *Discrete Optimization II*, *Annals of Discrete Mathematics*, vol. 5, pp. 287–326. Elsevier (1979). [https://doi.org/10.1016/S0167-5060\(08\)70356-X](https://doi.org/10.1016/S0167-5060(08)70356-X)
11. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM* **34**(1), 144–162 (1987). <https://doi.org/10.1145/7531.7535>
12. Jansen, K.: An eptas for scheduling jobs on uniform processors: Using an milp relaxation with a constant number of integral variables. *SIAM Journal on Discrete Mathematics* **24**(2), 457–485 (2010). <https://doi.org/10.1137/090749451>
13. Jansen, K., Klein, K.M., Verschae, J.: Closing the gap for makespan scheduling via sparsification techniques. *Mathematics of Operations Research* **45**(4), 1371–1392 (2020). <https://doi.org/10.1287/moor.2019.1036>
14. Jansen, K., Maack, M., Rau, M.: Approximation schemes for machine scheduling with resource (in-)dependent processing times **15**(3) (2019). <https://doi.org/10.1145/3302250>
15. Kellerer, H.: An approximation algorithm for identical parallel machine scheduling with resource dependent processing times. *Operations Research Letters* **36**(2), 157–159 (2008). <https://doi.org/10.1016/j.orl.2007.08.001>
16. Klusáček, D., Tóth, Š., Podolníková, G.: Real-life experience with major reconfiguration of job scheduling system. In: Desai, N., Cirne, W. (eds.) *Job Scheduling Strategies for Parallel Processing*, pp. 83–101. Springer (2017)
17. Kovalyov, M.Y., Shafransky, Y.M.: Uniform machine scheduling of unit-time jobs subject to resource constraints. *Discrete Applied Mathematics* **84**(1), 253–257 (1998). [https://doi.org/10.1016/S0166-218X\(97\)00138-8](https://doi.org/10.1016/S0166-218X(97)00138-8)
18. Krause, K.L., Shen, V.Y., Schwetman, H.D.: A task-scheduling algorithm for a multiprogramming computer system. *SIGOPS Oper. Syst. Rev.* **7**(4), 112–118 (1973). <https://doi.org/10.1145/957195.808058>
19. MetaCentrum Czech National Grid: MetaCentrum workload logs (2015), https://www.cs.huji.ac.il/labs/parallel/workload/1_metacentrum2/index.html
20. Naruszko, A., Przybylski, B., Rządca, K.: Artifact and instructions to generate experimental results for the Euro-Par 2021 paper: A log-linear $(2 + 5/6)$ -approximation algorithm for parallel machine scheduling with a single orthogonal resource (August 2021). <https://doi.org/10.6084/m9.figshare.14748267>
21. Niemeier, M., Wiese, A.: Scheduling with an orthogonal resource constraint. In: Erlebach, T., Persiano, G. (eds.) *Approximation and Online Algorithms*, pp. 242–256. Springer Berlin Heidelberg (2013)