Improved Scheduling with a Shared Resource via Structural Insights

Christoph Damerius[1(✉)], Peter Kling[1], Minming Li[2], Florian Schneider[1], and Ruilong Zhang[2]

[1] Universität Hamburg, Hamburg, Germany
christoph.damerius@uni-hamburg.de
[2] City University Hong Kong, Kowloon, Hong Kong SAR, China

Abstract. We consider a scheduling problem with resource-dependent processing speeds in which n jobs have to be scheduled on m machines that share a common resource. The resource may be distributed arbitrarily among the machines. This distribution is under the control of the scheduler and can be changed over time. Each job j has a processing volume $p_j \in \mathbb{N}$ and a resource requirement $r_j \in (0, 1]$. The latter indicates what fraction of the resource a job requires to run at full speed. Providing it with a larger share is not beneficial, but lowering its share results in a proportionally lowered processing speed. The goal is to schedule all jobs non-preemptively while minimizing the latest completion time.

This problem was introduced by Kling et al. [SPAA'17], who proved NP-hardness and gave an efficient algorithm with approximation ratio $2 + 1/(m - 2)$. The (asymptotic) tightness of that bound was left as an open question. We focus on the case of two machines and derive a strong, structural lower bound. This lower bound is based on a relaxed version and allows us to design an asymptotic 3/2-approximation that runs in time $\mathrm{O}(n \cdot \log n)$. As an immediate consequence we also get an improved 9/4-approximation for the case of three machines.

Keywords: Approximation algorithm · Multiprocessor scheduling · Relaxation · Resource constraints · Shared resource · Makespan

1 Introduction

Resource allocation is probably among the oldest and most well-studied optimization problems. In the context of computing systems, the resource typically corresponds to computational power, often in the form of a number of machines that must process a set of incoming jobs while optimizing a suitable quality of service measure. Even for this restricted scenario, there is a huge variety of models, differing in both machine and job properties as well as in the considered quality of service measures (see [11] for a detailed overview).

However, computational power is not the only contended resource in computing systems. In fact, in modern HPC environments computational power is

© Springer Nature Switzerland AG 2020
W. Wu and Z. Zhang (Eds.): COCOA 2020, LNCS 12577, pp. 168–182, 2020.
https://doi.org/10.1007/978-3-030-64843-5_12

rarely the performance bottleneck. Instead, often other shared resources, like the bandwidth or an I/O bus, constitute the performance bottleneck of such systems. Thus, the distribution of these additional shared resources can severely impact the system performance [14]. In this work, we study systems with such an additional shared resource.

Both standard *resource constrained scheduling* [4,8,12] (in which jobs require a certain amount of the resource to be able to run) and scheduling models with *resource dependent processing times* [6–9] (where a job's processing time depends on the amount of resource it receives) found considerable interest in the research community. The model we study falls into the category of resource dependent processing times. However, a particular feature is that we not only allow the scheduler to assign a resource share to a job *once* (when it is started). Instead, the scheduler may readjust the resource distribution adaptively at integral time points (processor cycles).

As an example, consider a multiprocessor system with a shared communication bus of limited bandwidth. The processed jobs may have different communication requirements, depending on their data-processing (generation or consumption) rate. Assigning a job a bandwidth that saturates its data-processing rate yields optimal performance, while throttling its bandwidth typically results in an immediate, proportional efficiency drop. On the other hand, increasing a job's bandwidth above its data-processing rate has no beneficial effect. As jobs enter and leave the system, the scheduler should adjust the resource distribution to the new situation. Note that while the linear efficiency drop is natural in the described setting, the resource dependency may be more complex (e.g., concave), as in the case of a shared power supply or cooling system [13].

We obviously adopt an idealized perspective by disregarding aspects like how CPU-intensive a given job is and by assuming that the shared resource is the performance bottleneck. Nevertheless, we aim at understanding exactly this aspect of resource allocation in modern data driven computing centers, in which computational power is often available in abundance.

1.1 Basic System Model

The following scheduling problem, originally proposed by Kling et al. [10], models the scenario described above. There are $m \in \mathbb{N}$ *machines* from the set $M := [m] = \{1, 2, \dots, m\}$ and $n \in \mathbb{N}$ *jobs* from the set $J := [n]$. Time is partitioned into integral (time) *slots* $t \in \mathbb{N}_0$, representing the time interval $[t, t+1)$. The machines share a common, finite *resource*. During any slot t, each machine $i \in M$ is assigned a fraction $R_i(t) \in [0, 1]$ of the resource. The resource may not be overused, so we require $\sum_{i \in M} R_i(t) \leq 1$. At any time, each machine can be assigned at most one job (no machine sharing) and each job can be assigned to at most one machine (no parallelism). A job $j \in J$ is defined via two parameters, its *processing volume* $p_j > 0$ and its *resource requirement* $r_j \in (0, 1]$.[1] If j is assigned

[1] In [10], $r_j > 1$ is allowed. Our restriction is without loss of generality, as we can assign such jobs a resource requirement of 1 and increase their processing volume by a factor r_j to get an equivalent instance.

to machine i during slot t, it is processed at speed $\min\{1, R_i(t)/r_j\}$, which is also the amount of the job's processing volume that finishes during this slot. Job j *finishes* in the first slot t after which all p_j units of its processing volume are finished. Preemption of jobs is not allowed, so once a job j is assigned to machine i, no other job can be assigned to i until j is finished. The objective is to find a *schedule* (a resource and job assignment adhering to the above constraints) that has minimum *makespan* (the first time when all jobs are finished).

This problem is known as Shared Resource Job-Scheduling *(SRJS)* [10]. In the bandwidth example from before, the resource requirement r_j models how communication intensive a job is. For example, $r_j = 0.5$ means that the process can utilize up to half of the available bandwidth. Assigning it more than this does not help (it cannot use the excess bandwidth), while decreasing its bandwidth results in a linear drop of the job's processing speed.

Simplifying Assumptions. To simplify the exposition, throughout this paper we assume $r_j \neq 1$ for all $j \in J$. This simplifies a few notions and arguments (e.g., we avoid slots in which formally two jobs are scheduled but only one job is processed at non-zero speed) but is not essential for our results. We also assume $p_j \in \mathbb{N}$, which is not required in [10] but does not change the hardness or difficult instances and is a natural restriction given the integral time slots.

1.2 Related Work

In the following we give an overview of the most relevant related work. In particular, we survey known results on SRJS and other related resource constrained scheduling problems. We also briefly explain an interesting connection to bin packing.

Known Results for SRJS. The SRJS problem was introduced by Kling et al. [10], who proved strong NP-hardness even for two machines and unit size jobs. Their main result was a polynomial time algorithm with an approximation ratio of $2 + 1/(m-2)$. If jobs have unit size, a simple modification of their algorithm yields an asymptotic $\big(1 + 1/(m-1)\big)$-approximation.

Althaus et al. [1] considered the SRJS problem for *unit size* jobs in a slightly different setting, in which the jobs' assignment to machines and their orders on each machine are fixed. They prove that this variant is NP-hard if the number of machines is part of the input and show how to efficiently compute a $(2 - 1/m)$-approximation. Furthermore, they provide a dynamic program that computes an optimal schedule in (a rather high) polynomial time, when m is not part of the input. For the special case $m = 2$, a more efficient algorithm with running time $\mathrm{O}\left(n^2\right)$ is given.

Resource Constrained Scheduling. One of the earliest results for scheduling with resource constraints is due to Garey and Graham [4]. They considered n jobs that must be processed by m machines sharing s different resources. While processed, each job requires a certain amount of each of these resources. Garey and

Graham [4] considered list-scheduling algorithms for this problem and proved an upper bound on the approximation ratio of $s+2-(2s+1)/m$. Garey and Johnson [5] proved that this problem is already NP-complete for a single resource, for which the above approximation ratio becomes $3-3/m$. The best known absolute approximation ratio for this case is due to Niemeier and Wiese [12], who gave a $(2+\epsilon)$-approximation. Using a simple reduction from the PARTITION problem, one can see that no efficient algorithm can give an approximation ratio better than 3/2, unless $P = NP$. While this implies that there cannot be a PTAS (polynomial-time approximation scheme), Jansen et al. [8] recently gave an APTAS (asymptotic PTAS).

Resource Dependent Processing Times. A common generalization of resource constrained scheduling assumes that the processing time of a job depends directly on the amount of resource it receives. Among the first to consider this model was Grigoriev et al. [6], who achieved a 3.75-approximation for unrelated machines. For identical machines, Kellerer [9] improved the ratio to $3.5+\epsilon$. In a variant in which jobs are already preassigned to machines and for succinctly encoded processing time functions, Grigoriev and Uetz [7] achieved a $(3+\epsilon)$-approximation. Recently, Jansen et al. [8] gave an asymptotic PTAS for scheduling with resource dependent processing times.

Note that in resource dependent processing times, the resource a job gets assigned is fixed and cannot change over time. In contrast, the model we consider gives the scheduler the option to adjust the resource assignment over time, which may be used to prioritize a short but resource intensive job during the processing of a long but less resource hungry job.

Connection to Bin Packing. Resource constrained scheduling problems are often generalizations of bin packing problems. For example, for a single resource, unit processing times and k machines, resource constrained scheduling is equivalent to *bin packing with cardinality constraints* [2,8] (where no bin may contain more than k items). Similarly, the SRJS problem is a generalization of *bin packing with cardinality constraints and splittable items*. Here, items may be larger than the bin capacity but can be split, and no bin may contain more than k item parts. This problem is (up to preemptiveness) equivalent to SRJS for k machines if all resource requirements are 1 and processing volumes correspond to item sizes.[2] In this case, each time slot can be seen as one bin.

Since we consider arbitrary resource requirements, we refer to [3] for the state of the art in bin packing with cardinality constraints and splittable items.

1.3 Our Contribution

We construct an efficient algorithm for SRJS on two machines that improves upon the previously best known approximation factor 2. Specifically, our main

[2] Alternatively, one can allow resource requirements > 1 and use these as item sizes while setting all processing volumes to 1, as described in [10].

result is the following theorem: (Due to space constraint, some proofs are omitted in this conference version.)

Theorem 1. *There is an asymptotic* 1.5*-approximation algorithm for SRJS with* $m = 2$ *machines that has running time* $\mathrm{O}(n \log n)$.

This is the first algorithm reaching an approximation ratio below 2. As a simple consequence, we also get an improved asymptotic 9/4-approximation for $m = 3$ machines (compared to the bound $2 + 1/(m-2) = 3$ from [10] for this case).

Our approach is quite different from Kling et al. [10]. They order jobs by increasing resource requirement and repeatedly seek roughly m jobs that saturate the full resource. By reserving one machine to maintain a suitable invariant, they can guarantee that, during the first phase of the algorithm, either always the full resource is used or always (almost) m jobs are given their full resource requirement. In the second phase, when there are less than m jobs left and these cannot saturate the resource, those jobs are always given their full resource requirement. Both phases are easily bounded by the optimal makespan, which is where the factor 2 comes from. While the bound of $2 + 1/(m-2)$ becomes unbounded for $m = 2$ machines, the algorithm in this case basically ignores one machine and yields a trivial 2-approximation.

The analysis of [10] relies on two simple lower bounds: the optimal makespan OPT is at least $\sum_{j \in J} r_j \cdot p_j$ (job j must be assigned a total of $r_j \cdot p_j$ resource share over time) and at least $\sum_{j \in J} \lceil p_j \rceil / m$ (job j occupies at least p_j/m time slots on some machine). Our improvement uses a more complex, structural lower bound based on a relaxed version of SRJS as a blueprint to find a good, non-relaxed schedule. The relaxed version allows (a) preemption and (b) that the resource and job assignments is changed at arbitrary (non-integral) times. More exactly, we show that there is an *optimal* relaxed *structured schedule* S_R in which, except for a single *disruptive* job j_D, one machine schedules the jobs of large resource requirement in descending order of resource requirement and the other machine schedules the jobs of small resource requirement in ascending order of resource requirement. We further simplify such a structured schedule S_R by assigning jobs j with small r_j their full resource requirement, yielding an *elevated schedule* $\hat{S}_R$. This elevated schedule is no longer necessarily optimal, but we can show that it becomes not much more expensive. This elevated schedule $\hat{S}_R$ yields the aforementioned structural lower bound, which we use to guide our algorithm when constructing a valid, non-relaxed schedule. The following theorem states a slightly simplified version of the guarantees provided by our structural lower bound. See Theorem 3 for the full formal statement.

Theorem 2. *There is an optimal structured relaxed schedule* S_R *and an elevated structured relaxed schedule* $\hat{S}_R$ *with a distinguished disruptive job* j_D *such that*

1. *if* $r_{j_D} \leq 1/2$, *then* $|\hat{S}_R| \leq |S_R|$ *and*
2. *if* $r_{j_D} > 1/2$, *then* $|\hat{S}_R| \leq c_{j_D} \cdot |S_R| + 0.042 \cdot A_{j_D}$.

Here, A_{j_D} *denotes the total time for which* j_D *is scheduled in* S_R *and the value* c_{j_D} *depends on* r_{j_D} *but lies in* $[1, 1.18)$.

While our lower bound does not immediately extend to the case of more machines, we believe that this is an important first step towards designing an improved approximation algorithm for arbitrary number of machines.

Note that we also show that our bound from Theorem 1 is tight in the sense that there are instances for which an optimal relaxed schedule is by a factor of 3/2 shorter than an optimal non-relaxed schedule. Thus, improving upon the asymptotic 3/2-approximation would require new, stronger lower bounds.

2 Preliminaries

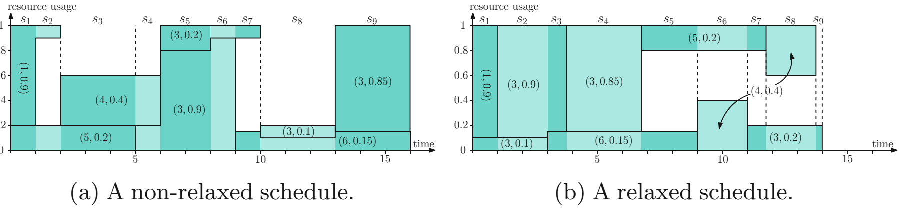
Before we derive the structured lower bound in Sect. 3 and use it to derive and analyze our algorithm in Sect. 4, we introduce some notions and notation which are used in the remainder of this paper.

Schedules. We model a schedule as a finite sequence S of *stripes*. A stripe $s \in S$ represents the schedule in a maximal time interval $I(s)$ with integral endpoints in which the job and resource assignments remain the same. The order of these stripes in the sequence S corresponds to the temporal order of these intervals. We let $J(s) \subseteq J$ denote the jobs scheduled during $I(s)$. For $j \in J(s)$ we let $R_j(s)$ denote the resource share that job j receives during $I(s)$ (i.e., the resource assigned to the machine that runs j). To ease the exposition, we sometimes identify a stripe s with the time interval $I(s)$. This allows us, e.g., to speak of a job $j \in J(s)$ scheduled during stripe s, to use $|s| := |I(s)|$ to refer to the *length* of a stripe s, or to write $s \subseteq T$ if a stripe's interval $I(s)$ is contained in another time interval T.

Any finite sequence S of stripes can be seen as a – possibly invalid – schedule for SRJS. To ensure a valid schedule, we need additional constraints: The resource may not be overused and we may never schedule more than m jobs, so we must have $\sum_{j \in J(s)} R_j(s) \leq 1$ and $|J(s)| \leq m$ for all $s \in S$. Since j is processed at speed $\min\{1, R_j(s)/r_j\}$ during s, we can assume (w.l.o.g.) that $R_j(s) \leq r_j$ for all $s \in S$ and $j \in J(s)$. With this assumption, the requirement that a schedule finishes all jobs can be expressed as $\sum_{s \in S\colon j \in J(s)} |s| \cdot R_j(s)/r_j \geq p_j$ for all $j \in J$. Not allowing preemption implies that the *active time* $A_j := \bigcup_{s \in S\colon j \in J(s)} I(s)$ of each job j must form itself an interval. While the sequence S does not give a specific assignment of jobs to machines, we can derive such an assignment easily via a greedy round robin approach.

W.l.o.g., we assume that for all $s \in S$ we have $J(s) \neq \emptyset$, since otherwise we can delete s and move subsequent stripes by $|s|$ to the left. Thus, we can define the *makespan* of S as $|S| := \sum_{s \in S} |s|$. When dealing with multiple schedules, we sometimes use superscript notation (e.g., A_j^S) to emphasize the schedule to which a given quantity refers.

Relaxed Schedules. We also consider a relaxed version of the SRJS problem (r-SRJS), in which the resource and job assignments may change at non-integral times and in which jobs can be preempted (and migrated) finitely often. This



(a) A non-relaxed schedule. (b) A relaxed schedule.

Fig. 1. A non-relaxed and a relaxed schedule for $m = 2$ machines and $n = 8$ jobs, with their parameters indicated in forms of tuples (p_j, r_j). Stripes are indicated by color grading. Note that in the non-relaxed schedule, all stripes start/end at integral time points and no job is preempted. (Color figure online)

gives rise to *relaxed schedules* S, which are finite sequences of stripes adhering to the same constraints as schedules except that the time intervals $I(s)$ for $s \in S$ may now have *non-integral* endpoints and the jobs' active times A_j are not necessarily intervals. Figure 1 illustrates how relaxed schedules differ from non-relaxed schedules. Relaxed schedules can be considerably shorter.

Subschedules, Substripes, and Volume. For a schedule S we define a *subschedule* S' of S as an arbitrary, not necessarily consecutive subsequence of S. Similarly, a *relaxed subschedule* S'_R of a relaxed schedule S_R is a subsequence of S_R. A *substripe* s' of a stripe s is a restriction of s to a subinterval $I(s') \subseteq I(s)$ (we denote this by $s' \subseteq s$). In particular, s' has the same job set $J(s') = J(s)$ and the same resource assignments $R_j(s') = R_j(s)$ for all $j \in J(s')$. For a (sub-)stripe s we define the *volume* of job $j \in J(s)$ in s as $V_j(s) := R_j(s) \cdot |s|$. The *volume* of a (sub-)stripe $s \in S$ is defined as $V(s) := \sum_{j \in J(s)} V_j(s)$ and the *volume* of a subschedule S' as $V(S') := \sum_{s \in S'} V(s)$.

Big and Small Jobs. Using the resource requirements, we partition the job set $J = J_B \uplus J_S$ into *big jobs* $J_B := \{\, j \in J \mid r_j > 1/2 \,\}$ and *small jobs* $J_S := \{\, j \in J \mid r_j \leq 1/2 \,\}$. Given a (relaxed or non-relaxed) schedule S, a *region* is a maximal subschedule of consecutive stripes during which the number of big jobs and the number of small jobs that are simultaneously scheduled remain the same. The *type* $\mathcal{T} \in \{\, B, S, (B, B), (S, S), (B, S) \,\}$ of an interval, (sub-)stripe, region indicates whether during the corresponding times exactly one big/small job, exactly two big/small jobs, or exactly one big job and one small job are scheduled. We call a stripe of type $\mathcal{T}$ a $\mathcal{T}$-stripe and use a similar designation for (sub-)stripes, regions, and intervals. If there exists exactly one $\mathcal{T}$-region for schedule S, then we denote that region by $S^{\mathcal{T}}$.

3 A Lower Bound for SRJS

To derive a lower bound, we aim to "normalize" a given optimal relaxed schedule such that it gets a favorable structure exploitable by our algorithm. While this

normalization may increase the makespan, Theorem 3 will bound this increase by a small factor. In the following we first provide the high level idea of our approach and major notions. Afterwards we give the full formal definitions and results.

High Level Idea. We can assume that, at each time point, the optimal relaxed schedule either uses the full resource or both two jobs reach their full resource requirement (see Definition 1). A relaxed schedule that satisfies this property is called *reasonable*. Any unreasonable schedule can be easily transformed into a reasonable schedule.

Next, we balance the jobs by ordering them. We show that it is possible to transform a relaxed optimal schedule such that one machine schedules the jobs of large resource requirement in decreasing order of resource requirement and, similarly, the other machine schedules the jobs of small resource requirement in ascending order of resource requirement. However, the load of machines may not be equal in the end. Therefore, we may need a *disruptive* job j_D to equalize the loads. The aforementioned order then only holds up to some point where j_D starts. Intuitively, if this job would be scheduled later, the load of both machines would be further imbalanced and the makespan would increase. A relaxed schedule that satisfies the above ordering constraint is called *ordered* (see Definition 3 for the formal definition). If an ordered relaxed schedule satisfies an additional property about the scheduling of j_D, we call that schedule *structured* (see Definition 4). We prove that we can always transform an optimal relaxed schedule into an ordered one without increasing the makespan (see Lemma 1).

As a further simplification, we increase the resource of all small jobs to their full resource requirement. This process is called *elevating* (see Definition 5). Intuitively, elevating makes the small jobs be processed with a higher speed, but this may come at the price of processing big jobs with a lower speed and thus increase the makespan. To be more precise, we show that the makespan may only increase when small jobs scheduled together with a disruptive big job are elevated. In Theorem 3, we analyze the makespan increase incurred by elevating in detail.

Formal Definitions and Results

Definition 1. For a job set J, we call a (sub-)stripe s with $J(s) \subseteq J$ *reasonable* if $\sum_{j \in J} R_j(s) = 1$ or $R_j(s) = r_j$ for all jobs $j \in J(s)$. A subschedule is called *reasonable* if all of its (sub-)stripes are reasonable and *unreasonable* otherwise.

Definition 2. Define a strict total order $\prec$ on $j, j' \in J$ where $r_j \neq r_{j'}$ as $j \prec j' :\Leftrightarrow r_j < r_{j'}$. Otherwise order j,j' arbitrarily but consistently by $\prec$.

The following Definition 3 formalizes the *ordered*-property of relaxed schedules S_R, which is central to our lower bound. Intuitively, it requires that S_R can be split into a left part and a right part (separated by a stripe index l). The left part schedules the smallest jobs $J_{\prec}^{S_R}$ in ascending order (according to $\prec$) and the biggest jobs $J_{\succ}^{S_R}$ in descending order. It is separated from the right part

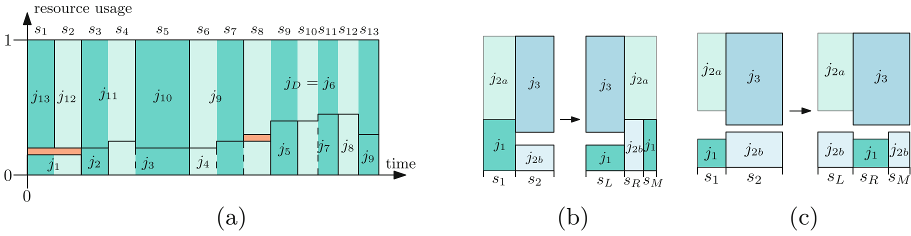


Fig. 2. (a) An ordered relaxed schedule $S_R = (s_i)_{i=1,...,13}$ with the disruptive job $j_D = j_6$, where $j_1 \prec \cdots \prec j_{13}$. In this example, $l = 8$, $J_R^{S_R} = \{ j_4, \ldots, j_9 \}$, $j_D \in J_B$ and as such the other jobs in $s_8, \ldots, s_{13}$ are sorted ascendingly after $\prec$. (b+c) Exchange argument of Observation 1.

throughout which the disruptive job j_D and the remaining jobs are scheduled. j_D could be any of the jobs from $J \setminus (J_{\prec}^{S_R} \cup J_{\succ}^{S_R})$ and adheres to no further ordering constraint, hence its disruptive nature. The ordering of the remaining jobs which are scheduled together with j_D is either ascending (if j_D is big) or otherwise descending. (See Fig. 2a for an example.) The definition also includes a special case where all stripes schedule j_D (then S_R only comprises out of its right part). We start by giving the main definitions for *ordered*, *structured* and *elevated* relaxed schedules.

Definition 3. For a job set J, we call a reasonable relaxed schedule $S_R = (s_1, \ldots, s_k)$ *ordered* for the *disruptive* job $j_D \in J$, if

1. If s is a stripe with $|J(s)| = 1$, then $s = s_k$ and $A_{j_D} = [0, |S_R|)$.
2. If $\forall i = 1, \ldots, k : |J(s_i)| = \{ j_{i,1}, j_{i,2} \}$ with $j_{i,1} \prec j_{i,2}$, then there exists an $l \in [k]$ such that $j_D \in J(s_i)$ iff $i \geq l$ and for $J_R^{S_R} := \{ j \mid R_j(s_i) > 0, i \in \{ l, \ldots, k \} \}$, we have $j_{1,1} \preceq \cdots \preceq j_{l-1,1} \preceq J_R^{S_R} \preceq j_{l-1,2} \cdots \preceq j_{1,2}$. Further, define $J_{\prec}^{S_R} = (j_{i,1})_{i=1,\ldots,l-1}$ and $J_{\succ}^{S_R} = (j_{i,2})_{i=1,\ldots,l-1}$.
3. For all $i \leq k-1$ where $J(s_i) = \{ j_D, j \}, J(s_{i+1}) = \{ j_D, j' \}$ with $j \neq j'$, we have $j \prec j'$ iff $j_D \in J_B$.

Remark 1. In the context of Definition 3, we can assume that j_D is always chosen such that there exists a stripe s with $j_D \in J(s)$ such that $j \preceq j_D$ for all $j \in J(s)$. Otherwise, choose s with $J(s) = \{ j'_D, j_D \}$ with j'_D being minimal according to $\prec$. We assign j'_D to be the new disruptive job, and reverse the order of all stripes that schedule j_D if necessary, to reobtain property 3 of Definition 3.

Remark 2. The orderedness immediately implies the existence of up to three regions in the order $(B,S), (B,B), B$ or $(B,S), (S,S), S$. For example, in Fig. 2a, $j_D \in J_B$ and therefore the jobs $j_6 = j_D, \ldots, j_{13}$ are big jobs, while $j_1, \ldots, j_5$ might all be small jobs. Then the stripes $s_1, \ldots, s_9$ form a (B,S) region, while $s_{10}, \ldots, s_{13}$ form a (B,B) region. No B region exists in this case.

We will first show that any optimal relaxed schedule can be transformed into an ordered schedule without losing its optimality (Lemma 1). The proof mainly relies on an exchange argument (Observation 1) to deal with unreasonable relaxed schedules.

Observation 1. Let s_1, s_2 be stripes with $J(s_1) = \{ j_1, j_{2a} \}$, $J(s_2) = \{ j_{2b}, j_3 \}$, where $j_1 \prec j_{2x} \prec j_3 \ \forall x \in \{ a, b \}$ and $j_{2a} \neq j_{2b}$. We can transform them into stripes s_L, s_R and possibly s_M, such that $J(s_L) = \{ j_1, j_3 \}$, $J(s_R) = \{ j_{2a}, j_{2b} \}$ and $J(s_M)$ is either $J(s_1)$ or $J(s_2)$, such that the volume of jobs scheduled and total length of stripes is unchanged.

Lemma 1. *For any job set J there exists an ordered, optimal relaxed schedule S_R.*

Definition 4. We call an ordered relaxed schedule S_R for J and for $j_D \in J$ *structured*, if no stripes s with $J(s) = \{ j_D, j' \}$ with $j_D \prec j'$ exist or $R_{j_D}(A_{j_D} \setminus S_R^{(B,S)}) = r_{j_D}$.

Definition 5. We call a subschedule $S'_R \subseteq S_R$ *elevated* in a relaxed schedule S_R, if $R_j(A_j^{S'_R}) = r_j$ for all $j \in J_S$.

The following two lemmas show the existence of structured, optimal relaxed schedules. Lemma 2 essentially tells us that an optimal relaxed schedule can either be fully elevated or structured and at least partially elevated. Lemma 3 then gives rise to structured optimal elevated relaxed schedules if the full elevation in Lemma 2 was possible. Unfortunately, the full elevation step can not always be pursued while staying optimal. Theorem 3 gives details about the makespan increase.

Lemma 2. *For every job set J there exists an optimal ordered (for job j_D) relaxed schedule S_R which is either elevated, or is elevated in $S_R \setminus A_{j_D}^{S_R}$, $R_{j_D}(A_{j_D}^{S_R} \setminus S_R^{(B,S)}) = r_{j_D}$ and $j_D \in J_B$.*

Lemma 3. *For any optimal elevated ordered relaxed schedule S_R for a job set J there exists a structured optimal elevated relaxed schedule $\hat{S}_R$ for J.*

Theorem 3. *For every job set J there exists a structured (for $j_D \in J$) optimal relaxed schedule S_R and a structured (for j_D), elevated relaxed schedule $\hat{S}_R$ such that one of the following holds:*

1. *$j_D \in J_S$ and $|\hat{S}_R| \leq |S_R|$.*
2. *$j_D \in J_B$ and $\hat{S}_R^B = \varnothing$. Let $a_X := |X^{(B,S)} \setminus A_{j_D}|$ and $b_X := |X^{(B,S)} \cap A_{j_D}|$ for $X \in \{ S_R, \hat{S}_R \}$. Then $|\hat{S}_R| \leq |S_R| + \lambda b_{S_R}$, $a_{\hat{S}_R} \leq a_{S_R}$ and $b_{\hat{S}_R} \leq b_{S_R}$, where λ is the smallest positive root of $(\lambda + 1)^3 - 27\lambda$.*
3. *$j_D \in J_B$, $\hat{S}_R^B \neq \varnothing$ and $|\hat{S}_R| \leq (4 - 2r_{j_D} - \frac{1}{r_{j_D}})|S_R| \leq (4 - \sqrt{8})|S_R|$.*

4 Approximation Algorithm and Analysis

Our approximation algorithm ALG for SRJS constructs the schedule by using the structure derived in Theorem 3 as a starting point. To accomplish this, ALG is designed to first gain information about the relaxed schedule $\hat{S}_R$ given by Theorem 3 by essentially replicating the order given by the orderedness property. Based on this information, either $\mathrm{ALG}_{\mathrm{big}}$ or $\mathrm{ALG}_{\mathrm{small}}$ is executed. Essentially, ALG determines whether j_D, as given by Theorem 3, is in J_S or in J_B and branches into $\mathrm{ALG}_{\mathrm{small}}$ or $\mathrm{ALG}_{\mathrm{big}}$ accordingly.

$\mathrm{ALG}_{\mathrm{small}}$ processes the jobs by first scheduling the small jobs in descending order of their processing volumes (using Assign), and scheduling the big jobs in arbitrary order afterwards. For this case it can be easily shown that our bound is satisfied (see the proof of Theorem 4).

For the case of $j_D \in J_B$, a more sophisticated approach is needed. $\mathrm{ALG}_{\mathrm{big}}$ roughly schedules the jobs in the (B, S) region, mimicking the order as in $\hat{S}_R$. Afterwards, $\mathrm{ALG}_{\mathrm{big}}$ schedules all remaining big jobs using a slightly modified longest-first approach. Care has to be taken on the transition between both parts of $\mathrm{ALG}_{\mathrm{big}}$. For that reason, we calculate two schedules S and S', one of which can be shown to match the desired bound. Their job scheduling order mainly differs in whether the longest (S) or most resource intensive remaining job (S') is scheduled first. The remaining jobs are then scheduled in a longest-first order. Lastly, the machine loads are balanced by adjusting the resources given to the big jobs in the second part (BalanceLength).

We will first give the pseudocode of the algorithm, then describe the subroutines involved and then give some analysis.

ALG(job set J)
1 $S'_R :=$ empty relaxed schedule
2 $J_S := \{j \in J \mid r_j \le 0.5\}$
3 $J_B := \{j \in J \mid r_j > 0.5\}$
4 Sort J_S ascendingly after $\prec$
5 Sort J_B descendingly after $\prec$
6 **for** $j \in J_S$ **do** Assign$(S'_R, j, 1, 0)$
7 $V := \sum_{j \in J_B} p_j r_j$
8 $H := J_B$
9 **for** $j \in J_B$ **do**
10     $t = load^{S'_R}(2)$
11   **if** GetEndPoint2$(S'_R, j) \ge load^{S'_R}(1)$:
12   **return** $\mathrm{ALG}_{\mathrm{big}}(J_S, H, J_B \setminus H, t)$
13   **if** $|H| = 1$:
14   **break**
15     $j_l := \mathrm{argmax}_{j' \in H \setminus \{j\}} p_{j'}$
16   Assign$(S'_R, j, 2, 0)$
17     $V := V - p_j r_j$
18     $u :=$ GetEndPoint2(S'_R, j_l)
19   **if** $(u - load^{S'_R}(1))(1 - r_{j_l}) > V - r_{j_l} p_{j_l}$:
20   **return** $\mathrm{ALG}_{\mathrm{big}}(J_S, H, J_B \setminus H, t)$
21     $H := H \setminus \{j\}$
22 **return** $\mathrm{ALG}_{\mathrm{small}}(J_S, J_B)$

$\mathrm{ALG}_{\mathrm{big}}$(job set J_S, job set H, job set J'_B, t)
1 $S := \varnothing$
2 $t' := \lceil \lfloor t \rfloor / 2 \rceil + 2$
3 Sort J'_B descendingly after $\prec$
4 Sort H descendingly after $\prec$
5 **for** $j \in J_S$ **do** Assign$(S, j, 1, t')$
6 **for** $j \in J'_B$ **do** Assign$(S, j, 2, 0)$
7 $j_f :=$ Pop(H)
8 Sort $j \in H$ descendingly after p_j
9 **if** $H = \varnothing$:
10   Assign$(S, j_f, 1, t')$ **and return** S
11 $j_l :=$ Pop(H)
12 $S' := S$
13 Assign$(S, j_l, 2, t')$
14 Assign$(S', j_f, 2, t')$
15 **if** $i_{\min}(S) = 2$: Assign$(S, j_f, 2, t')$
16 **else** : $H = H \cup \{j_f\}$
17 Assign$(S', j_l, i_{\min}(S'), t')$
18 **for** $j \in H$ **do** Assign$(S, j, i_{\min}(S), 1, t')$
19 **for** $j \in H \setminus \{j_f\}$ **do**
20   Assign$(S', j, i_{\min}(S'), 1, t')$
21 **for** $\hat{S} \in \{S, S'\}$ **do** BalanceLength$(\hat{S})$
22 **return** $\mathrm{argmin}_{\hat{S} \in \{S, S'\}} |\hat{S}|$

$\mathrm{ALG}_{\mathrm{small}}$(job set J_S, job set J_B)
1 $S :=$ empty schedule
2 Sort the $j \in J_S$ descendingly after p_j
3 **for** $j \in J_S$ **do** Assign$(S, j, i_{\min}(S), 0)$
4 $i := i_{\min}(S)$
5 **for** $j \in J_B$ **do** Assign$(S, j, i, 0)$
6 **return** S

Description of Subroutines. $\textsc{Assign}(S, j, i, t)$ schedules a job j into a (relaxed or unrelaxed) schedule S on machine i, starting at the earliest time point possible, but not before t. In contrast to the lower bound, where we did not state on which machine we schedule, here we always give a specific machine to schedule on, as it simplifies further analysis. The resource is assigned to j by consecutively considering the slots $s = t, t+1, \ldots$ and in each of these slots giving j the maximum possible resource. The given resource is only restricted by r_j, other already scheduled jobs (their given resources remain unchanged) and the remaining processing volume that j needs to schedule in s.

For a (relaxed) schedule S, we denote by $load^S(i)$ the earliest time point after which machine i remains idle for the rest of S. Furthermore, define $i_{\min}(S) \in \{1, 2\}$ to be a machine that has the lowest load in S. We assume that job sets will retain their ordering when passed as arguments or when elements are removed.

$\textsc{GetEndPoint2}(S, j)$ simulates scheduling j into S as $\textsc{Assign}(S, j, 2, 0)$ does, but instead returns $load^S(2)$ without altering S. This is useful so that we do not have to revert the scheduling of jobs.

$\textsc{BalanceLength}$ processes a given schedule S as follows: It only changes S if $\Delta := |load^S(2) - load^S(1)| \geq 2$. If so, it checks if the job j, scheduled in the last slot of S, is given r_j resource during $A_j^S \cup S^{(B,B)}$. As we have scheduled longest-first, there can be at most one stripe s with $J(s) = \{j, j'\}$ in $S^{(B,B)}$ during which j does not get r_j resource. The algorithm then gives j' less resource during s, pushing all other jobs scheduled with j after s to the right, shortening Δ. Simultaneously, $\textsc{BalanceLength}$ gives j more resource during s, shortening Δ even further. All jobs but j scheduled after s are given at most $1-r_j$ resource and as such their resource does not change when moved. Hence, it is straightforward to calculate how much volume has to be redistributed to shorten Δ until either $\Delta \leq 1$ or j is given r_j resource in s.

Outline of the Analysis. We will first analyze how ALG branches into $\text{ALG}_{\text{small}}$ or ALG_{big} and with which arguments. In the case that ALG branches into $\text{ALG}_{\text{small}}$, we can easily derive the 1.5-approximation factor in Theorem 4.

ALG_{big} basically consists of a first part where roughly the jobs from the (B, S)-region in $\hat{S}_R$ are scheduled and a second part where the remaining jobs are scheduled longest-first. To take care of the transition between both parts, we define the notion of a *bridge* job j_β:

Definition 6. For a structured relaxed schedule S_R, the *bridge* job j_β is the smallest job scheduled in $S_R^{(B,S)}$ that is not scheduled together with jobs $j \succ j_\beta$ in S_R (if it exists).

Using this definition, we are now able to give Lemma 4, which shows which sub-algorithm is executed with precisely which arguments by ALG.

Lemma 4. *Let J be a job set and S_R, $\hat{S}_R$ be the relaxed schedules given by Theorem 3 for J, with j_D being the disruptive job of $\hat{S}_R$. If property 1 of Theorem 3 holds, then calling* $\text{ALG}(J)$ *will execute* $\text{ALG}_{small}(J_S, J_B)$ *with*

J_S (J_B) being ascendingly (descendingly) sorted after $\prec$, respectively. Otherwise, $\mathrm{ALG}_{big}(J_S, H, J_B \setminus H, t)$ *will be executed with arguments* $t = \inf(A_{j_\beta}^{\hat{S}_R})$ *if there exists a bridge job j_β for $\hat{S}_R$, or $t = 0$ otherwise. Furthermore,* $H = \{\, j \in J_B \mid \inf(A_j^{\hat{S}_R}) \geq t \,\} \neq \varnothing$.

H is the set of all big jobs scheduled not before the bridge job. For the first part, $\mathrm{ALG}_{\mathrm{big}}$ schedules J_S and $J_B \setminus H$, mimicking the order as in $\hat{S}_R$ into the interval $[0, t' + \lfloor t \rfloor)$. The following observation guarantees that ALG can fit these jobs into said interval.

Observation 2. Let S_R be a relaxed schedule for a job set J as obtained by Theorem 3, with $S_R^{(B,S)} \neq \varnothing$. Then Assign in line 6 of $\mathrm{ALG}_{\mathrm{big}}$ does not schedule jobs beyond $t' + \lfloor t \rfloor$.

For the second part, only big jobs remain to be scheduled. We can show they need at most $\approx 4/3 \cdot V$ slots, where V is their total volume, or the number of slots they require is dominated by one long job (where all other big jobs can fit onto the other machine). We can then guarantee the bound for one of the schedules procured by $\mathrm{ALG}_{\mathrm{big}}$, which then helps us to prove the overall bound.

In summary, we show the following theorem.

Theorem 4. *For any job set J we have $|\mathrm{ALG}(J)| \leq 1.5 OPT + \mathrm{O}(1)$, and this bound is tight for* ALG.

Proof. We only show the asymptotic lower bound here. Construct a job set with $k \in \mathbb{Z}_{2n}$ unit-size jobs j with $r_j = 1/3$ and $k - 1$ unit-size jobs j with $r_j = 2/3$. We can obviously construct a schedule of makespan k. The corresponding relaxed schedule obtained by Theorem 3 will have $j_D \in J_S$, so $\mathrm{ALG}_{\mathrm{small}}$ will be executed. It will first schedule all small jobs in $k/2$ slots. Afterwards, all big jobs will be scheduled on the same machine, using $k - 1$ slots. This gives the asymptotic lower bound of 3/2.

5 Additional Results

Note that ALG, as stated in Sect. 4, does not necessarily have a running time of $\mathrm{O}(n \log n)$. However, we prove this running time using a slightly modified algorithm in the following lemma.

Lemma 5. ALG *can be implemented to run in* $\mathrm{O}(n \log n)$ *time for a job set J with $|J| = n$.*

Our results for the two-machine case also imply an improved bound for the three-machine case, which is based on ignoring one of the three machines.

Corollary 1. *For $m = 3$ and any job set J with $|J| = n$, we have $ALG \leq 9/4 \cdot OPT + \mathrm{O}(1)$, where* ALG *runs in $\mathcal{O}(n \log n)$ time and OPT is the optimal solution for 3 machines.*

Lemma 6. *For any $P \in \mathbb{N}$ there exists a job set J with $\sum_{j \in J} p_j \geq P$ such that $3/2 \cdot |S_R| \leq |S|$, where S_R and S are optimal relaxed (unrelaxed) schedules for J, respectively. Furthermore, S_R can be chosen not to use preemption.*

In Lemma 6, S_R did not even use preemption. Thus, the slotted time is the reason for the 3/2-gap between relaxed and unrelaxed schedules. To beat the approximation ratio of 3/2, we would have to improve our understanding of how slotted time affects optimal schedules.

6 Conclusion and Open Problems

Using structural insights about the SRJS problem, we were able to improve approximation results from [10] for the cases of $m \in \{2, 3\}$ machines in the SRJS problem. As mentioned in Sect. 5, our (asymptotic) 3/2-approximation for $m = 2$ is the best possible result that can be achieved with the lower bound based on our definition of relaxed schedules and can be computed in time $O(n \log n)$. This leaves two natural research questions.

First, can a similar approach improve further improve the competitive ratio of [10] for larger values of m? While the lower bound we constructed in Sect. 3 is tailored towards $m = 2$, the underlying principle may be used to design improved algorithms for $m > 2$ machines. Indeed, a key insight behind our improvement stems from a worst case instance for the algorithm of [10]: Consider an instance with a job j that has small resource requirement r_j and processing volume $p_j \approx \text{OPT}$. An optimal schedule must begin to process p_j early, in parallel to the rest of the instance. However, the algorithm from [10] is *resource-focused*, in the sense that it orders jobs by resource requirement and basically ignores the processing volume when selecting jobs to be processed. This might result in j being processed at the very end, after all other jobs have been finished, possibly yielding an approximation ratio of roughly 2 (for large m). One could fix this problem using an algorithm focused on processing-volume, but that would run into similar issues caused by different resource requirements. Our algorithm for $m = 2$ basically identifies jobs like j (the disruptive job) and uses it to balance between these two extremes. A key question when considering such an approach for larger m is how to identify a suitable (set of) disruptive job(s).

A second possible research direction is to beat the lower bound limit of our structural approach. Given its relation to other resource constrained scheduling problems and to bin packing variants, it seems possible that one can even find a PTAS for SRJS. While a PTAS has typically a worse runtime compared to more direct, combinatorial algorithms, it would yield solutions that are arbitrarily close to optimal schedules. A difficulty in constructing a PTAS seems to stem from the partition of time into *discrete* slots. An incautious approach might yield cases where all machines are stuck with finishing an ε-portion of work, forcing them to waste most of the available resource in such a time slot. If the average job processing time is small, this might have a comparatively large influence on the approximation factor. Previous work [10] reserved one of the m machines to

deal with such problems (which is wasteful for small m). Also augmenting the available resource in each slot by, e.g., a $1 + \varepsilon$ factor should help to circumvent such difficulties.

Acknowledgement. Peter Kling and Christoph Damerius were partially supported by the DAAD PPP with Project-ID 57447553. Minming Li is also from City University of Hong Kong Shenzhen Research Institute, Shenzhen, P.R. China. The work described in this paper was partially supported by Project 11771365 supported by NSFC.

References

1. Althaus, E., et al.: Scheduling shared continuous resources on many-cores. J. Sched. **21**(1), 77–92 (2017). https://doi.org/10.1007/s10951-017-0518-0
2. Epstein, L., Levin, A.: AFPTAS results for common variants of bin packing: a new method for handling the small items. SIAM J. Optim. **20**(6), 3121–3145 (2010)
3. Epstein, L., Levin, A., van Stee, R.: Approximation schemes for packing splittable items with cardinality constraints. Algorithmica **62**(1–2), 102–129 (2012)
4. Garey, M.R., Graham, R.L.: Bounds for multiprocessor scheduling with resource constraints. SIAM J. Comput. **4**(2), 187–200 (1975)
5. Garey, M.R., Johnson, D.S.: Complexity results for multiprocessor scheduling under resource constraints. SIAM J. Comput. **4**(4), 397–411 (1975)
6. Grigoriev, A., Sviridenko, M., Uetz, M.: Machine scheduling with resource dependent processing times. Math. Program. **110**(1), 209–228 (2007)
7. Grigoriev, A., Uetz, M.: Scheduling jobs with time-resource tradeoff via nonlinear programming. Discret. Optim. **6**(4), 414–419 (2009)
8. Jansen, K., Maack, M., Rau, M.: Approximation schemes for machine scheduling with resource (in-)dependent processing times. ACM Trans. Algorithms **15**(3), 31:1–31:28 (2019)
9. Kellerer, H.: An approximation algorithm for identical parallel machine scheduling with resource dependent processing times. Oper. Res. Lett. **36**(2), 157–159 (2008)
10. Kling, P., Mäcker, A., Riechers, S., Skopalik, A.: Sharing is caring: multiprocessor scheduling with a sharable resource. In: Scheideler, C., Hajiaghayi, M.T. (eds.) Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, 24–26 July 2017, pp. 123–132. ACM (2017)
11. Leung, J.Y. (ed.): Handbook of Scheduling - Algorithms, Models, and Performance Analysis. Chapman and Hall/CRC (2004)
12. Niemeier, M., Wiese, A.: Scheduling with an orthogonal resource constraint. Algorithmica **71**(4), 837–858 (2015)
13. Rózycki, R., Weglarz, J.: Improving the efficiency of scheduling jobs driven by a common limited energy source. In: 23rd International Conference on Methods & Models in Automation & Robotics, MMAR 2018, Międzyzdroje, Poland, 27–30 August 2018, pp. 932–936. IEEE (2018)
14. Trinitis, C., Weidendorfer, J., Brinkmann, A.: Co-scheduling: prospects and challenges. In: Trinitis, C., Weidendorfer, J. (eds.) Co-Scheduling of HPC Applications [Extended Versions of All Papers from COSH@HiPEAC 2016, Prague, Czech Republic, 19 January 2016]. Advances in Parallel Computing, vol. 28, pp. 1–11. IOS Press (2016)