Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Scheduling Parallel-Task Jobs Subject to Packing and Placement Constraints

Mehrnoosh Shafiee, Javad Ghaderi

To cite this article:
Mehrnoosh Shafiee, Javad Ghaderi (2022) Scheduling Parallel-Task Jobs Subject to Packing and Placement Constraints. Operations Research

Published online in Articles in Advance 18 Jan 2022

. https://doi.org/10.1287/opre.2021.2198

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

http://pubsonline.informs.org/journal/opre

OPERATIONS RESEARCH
Articles in Advance, pp. 1–17
ISSN 0030-364X (print), ISSN 1526-5463 (online)

Methods

Scheduling Parallel-Task Jobs Subject to Packing and Placement Constraints

Mehrnoosh Shafiee,[a] Javad Ghaderi[a]

[a] Department of Electrical Engineering, Columbia University, New York, New York 10027
Contact: s.mehrnoosh@columbia.edu, https://orcid.org/0000-0002-4558-4746 (MS); jghaderi@ee.columbia.edu (JG)

Received: May 8, 2020
Revised: June 6, 2020; January 12, 2021
Accepted: July 27, 2021
Published Online in Articles in Advance: January 18, 2022

Area of Review: Optimization

https://doi.org/10.1287/opre.2021.2198

Abstract. Motivated by modern parallel computing applications, we consider the problem of scheduling parallel-task jobs with heterogeneous resource requirements in a cluster of machines. Each job consists of a set of tasks that can be processed in parallel; however, the job is considered completed only when all its tasks finish their processing, which we refer to as the *synchronization* constraint. Furthermore, assignment of tasks to machines is subject to *placement* constraints, that is, each task can be processed only on a subset of machines, and processing times can also be machine dependent. Once a task is scheduled on a machine, it requires a certain amount of resource from that machine for the duration of its processing. A machine can process (*pack*) multiple tasks at the same time; however, the cumulative resource requirement of the tasks should not exceed the machine's capacity. Our objective is to minimize the weighted average of the jobs' completion times. The problem, subject to synchronization, packing, and placement constraints, is NP-hard, and prior theoretical results only concern much simpler models. For the case that migration of tasks among the placement-feasible machines is allowed, we propose a preemptive algorithm with an approximation ratio of $(6+\epsilon)$. In the special case that only one machine can process each task, we design an algorithm with an improved approximation ratio of four. Finally, in the case that migrations (and preemptions) are not allowed, we design an algorithm with an approximation ratio of 24. Our algorithms use a combination of linear program relaxation and greedy packing techniques. We present extensive simulation results, using a real traffic trace, that demonstrate that our algorithms yield significant gains over the prior approaches.

Funding: This work was supported by the National Science Foundation [Grants CNS-1652115 and CNS-1717867].
Supplemental Material: The online appendices are available at https://doi.org/10.1287/opre.2021.2198.

Keywords: scheduling algorithms • approximation algorithms • data-parallel computing • datacenters

1. Introduction

Modern parallel computing frameworks, for example, Hadoop and Spark (Apache 2018a, c), have enabled large-scale data processing in computing clusters. In such frameworks, the data are typically distributed across a cluster of machines and are processed in multiple stages. In each stage, a set of tasks is executed on the machines, and once all the tasks in the stage finish their processing, the job is finished or moved to the next stage. For example, in MapReduce (Dean and Ghemawat 2008), in the map stage, each map task performs local computation on a data block in a machine and writes the intermediate data to the disk. In the reduce stage, each reduce task pulls intermediate data from different maps, merges them, and computes its output. Although the reduce tasks can start pulling data as map tasks finish, the actual computation by the reduce tasks can only start once all the map tasks are done and their data pieces are received. Furthermore, the job is not completed unless all the reduce tasks finish. Similarly, in Spark (Zaharia et al. 2016), the computation is done in multiple stages. The tasks in a stage can run in parallel; however, the next stage cannot start unless the tasks in the preceding stage(s) are all completed.

We refer to such constraints as *synchronization* constraints, that is, a stage is considered completed only when all its tasks finish their processing. Such synchronizations could have a significant impact on the jobs' latency in parallel computing clusters (Cheatham et al. 1996, Zaharia et al. 2008, Ananthanarayanan et al. 2010, Kambatla et al. 2010, Zaharia et al. 2016). Intuitively, an efficient scheduler should complete all the (inhomogeneous) tasks of a stage more or less around

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

the same time while prioritizing the stages of different jobs in an order that minimizes the overall latency in the system. The scheduler can only make scheduling decisions for the stages that have been released from various jobs up to that point (i.e., those that their preceding stages have been completed). In our model, we use the terms stage and job interchangeability.

Another main feature of parallel computing clusters is that jobs can have diverse tasks and processing requirements. This has been further amplified by the increasing complexity of workloads, that is, from traditional batch jobs, to queries, graph processing, streaming, and machine learning jobs, that all need to share the same cluster. The cluster manager (*scheduler*) serves the tasks of various jobs by reserving their requested resources (e.g., CPU, memory). For example, in Hadoop (Apache 2018a), the resource manager reserves the tasks' resource requirements by launching *containers* in machines. Each container reserves required resources for processing of a task. To improve the overall latency, we therefore need a scheduler that *packs* as many tasks as possible in the machines while retaining their resource requirements.

In practice, there are further placement constraints for processing tasks on machines. For example, each task is preferred to be scheduled on one of the machines that has its required data block (Dean and Ghemawat 2008, Ananthanarayanan et al. 2011) (a.k.a. data locality); otherwise, processing can slow down because of data transfer. The data block might be stored in multiple machines for robustness and failure considerations. However, if all these machines are highly loaded, the scheduler might actually need to schedule the task in a less loaded machine that does not contain the data.

Despite the vast scheduling literature, scheduling algorithms with theoretical results (approximation ratios) are mainly based on simple models, where each machine processes one task at a time, each job is a single task, or tasks can be processed on any machine arbitrarily (see Section 1.1). Such models *do not* fully capture the *modern features* of data-parallel computing clusters, namely,

- *Packing*: each machine is capable of processing multiple tasks at a time subject to its capacity.
- *Synchronization*: tasks that belong to the same job have a collective completion time which is determined by the slowest task in the collection.
- *Placement constraint*: a task's processing time is machine dependent, and a task is typically preferred to be processed on a subset of machines (e.g., where its input data block is located). Furthermore, each task at each time can get processed on at most a single machine.

The goal of this paper is to design scheduling algorithms, with theoretical guarantees, under the previous features of modern parallel computing clusters. For simplicity, we consider one dimension for task resource requirement (e.g., memory). Although task resource requirements are in general multidimensional (CPU, memory), it has been observed that memory is typically the bottleneck resource (Apache 2018b, Nitu et al. 2018).

Our objective is to minimize the weighted sum of completion times of existing jobs in the system, where weights can encode different priorities for the jobs. Clearly, minimization of the average completion time is a special case of this problem with equal weights. We consider both preemptive and nonpreemptive scheduling. In a *nonpreemptive* schedule, a task cannot be preempted (and hence cannot be migrated among machines) once it starts processing on a machine until it is completed. In a *preemptive* schedule, a task may be preempted and resumed later in the schedule, and we further consider two cases depending on whether migration of a task among machines is allowed or not.

1.1. Related Work

Default cluster schedulers in Hadoop (Zaharia et al. 2010, Apache 2018a) focus primarily on fairness and data locality. Such schedulers can make poor scheduling decisions by not packing tasks well together or having a task running long without enough parallelism with other tasks in the same job. Several cluster schedulers have been proposed to improve job completion times (Jin et al. 2011, Schwarzkopf et al. 2013, Grandl et al. 2015, Verma et al. 2015, Grandl et al. 2016, Liu and Shen 2016, Rasley et al. 2016, Wang et al. 2016, Yekkehkhany et al. 2018). However, they either do not consider all aspects of packing, synchronization, and data locality or use heuristics that are not necessarily efficient.

We highlight four relevant papers (Grandl et al. 2015, Verma et al. 2015, Wang et al. 2016, Yekkehkhany et al. 2018) here. Tetris (Grandl et al. 2015) is a scheduler that assigns scores to tasks based on best-fit bin packing and shortest-remaining-time-first (SRPF) heuristic and gives priority to tasks with higher scores. The data locality is encoded in scores by imposing a remote penalty to penalize use of remote resources. Borg (Verma et al. 2015) packs multiple tasks of jobs in machines from high to low priority, modulated by a round-robin scheme within a priority to ensure fairness across jobs. The scheduler considers data locality by assigning tasks to machines that already have the necessary data stored. The papers by Wang et al. (2016) and Yekkehkhany et al. (2018) focus on single-task jobs and study the mean delay of tasks under a stochastic model where, if a task is scheduled on one of the remote servers that do not have the input data, its average processing time will be larger, by a multiplicative factor, compared with the case that it is processed on a local server that contains the data.

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

They propose algorithms based on join-the-shortest-queue and max-weight (JSQ-MW) to incorporate data locality in load balancing. This model is generalized by Yekkehkhany et al. (2018) to more levels of data locality. However, these models do not consider any task packing in servers or synchronization issue among multiple tasks of the same job.

From a theoretical perspective, our problem of scheduling parallel-task jobs with synchronization, packing, and placement constraints can be seen as a generalization of different scheduling problems in which only one or two of the constraints are considered. There is a vast amount of literature for the problems under each of the constraints in isolation. Here, we review some of these works.

The concurrent open shop (COS) problem (Ahmadi et al. 2005) considers the synchronization constraint. In COS, each job consists of a set of tasks, each machine processes one task at a time, and each task can be processed on a specific machine. Unlike COS, in our model a machine can process (pack) multiple tasks simultaneously subject to its capacity, and there are further task placement constraints for assigning tasks to machines. Minimizing the weighted sum of completion times in COS is known to be Approximation (APX)-hard (Garg et al. 2007), with several two-approximation algorithms (Chen and Hall 2007, Garg et al. 2007, Leung et al. 2007, Bansal and Khot 2010, Mastrolilli et al. 2010, Sachdeva and Saket 2013). At a higher level, the synchronization constraint in our setting can be seen as a special case of the precedence constraint in which there exists a partial ordering among tasks of each job. Scheduling problems under precedence constraint are widely studied in the literature (Coffman and Bruno 1976, Munier et al. 1998, Goldberg et al. 2001, Queyranne and Schulz 2006, Li 2020).

There is also a line of research on the parallel tasks scheduling (PTS) problem (Garey and Graham 1975) where the focus is on the packing constraint. In PTS, each job is only a single task that requires a certain amount of resource for its processing time and can be served by any machine subject to its capacity. This differs from our model where each job has multiple tasks, each task can be served by a set of machines, and the job's completion time is determined by its last task. Minimizing the weighted sum of completion times in the PTS is also NP-complete in the strong sense (Blazewicz et al. 1983). In the case of a single machine, Schwiegelshohn (2004) proposed a nonpreemptive algorithm that can achieve approximation ratio of 7.11, and a preemptive algorithm, called *PSRS*, that can achieve approximation ratio of 2.37. In the case of multiple machines, there is only one result in the literature, which is a 14.85-approximation nonpreemptive algorithm (Remy 2004).

Furthermore, our setting is closely related to unrelated machine scheduling. In the unrelated machine scheduling (Skutella 2001, Schulz and Skutella 2002, Im and Li 2016, Bansal et al. 2019), each job is only a single task that can be scheduled on a subset of machines, and its processing time is machine dependent. Both preemptive and nonpreemptive versions of the problem are APX-hard. When preemption is not allowed and all the jobs are present at time 0, the best result is a $(3/2-c)$-approximation algorithm for some fixed $c > 0$ (Bansal et al. 2019). For the preemptive case, if migration is allowed, Skutella (2001) gives a two-approximation solution, and if migration is not allowed, Schulz and Skutella (2002) provide a $(3/2+\epsilon)$-approximation solution.

We emphasize that our setting of parallel-task jobs, subject to synchronization, packing, and placement constraints, is more challenging than the previous problems, and algorithms from these problems *cannot* be applied to our setting. Although we use some of the classical techniques such as relaxed linear programming formulation (Lenstra et al. 1990, Qiu et al. 2015) and Slow-Motion (Schulz and Skutella 1997) in our setting, as we will see, we need to carefully adjust them and add new ideas to make them work in our setting. To the best of our knowledge, this is the first paper that provides constant-approximation algorithms for this problem subject to synchronization, packing, and placement constraints.

1.2. Main Contributions

We briefly summarize our main results and describe our techniques. We propose scheduling algorithms for three cases.

- **Task Migration Allowed.** When migration is allowed, a task might be preempted several times and resume possibly on a different machine within its placement-feasible set. Our algorithm in this case is based on greedy scheduling of task fractions (fraction of processing time of each task) on each machine, subject to capacity and placement constraints. The task fractions are found by solving a relaxed linear program (LP), which divides the time horizon into geometrically increasing time intervals and uses *interval-indexed variables* to indicate what fraction of each task is served at which interval on each machine. We show that our scheduling algorithm has an approximation ratio better than $(6+\epsilon)$, for any $\epsilon > 0$.
- **Task Migration Not Allowed.** When migration is not allowed, the schedule can be nonpreemptive or preemptive while all preemptions occur on the same machine. In this case, our algorithm is based on mapping tasks to proper time intervals on the machines. We use the interval-indexed variables to form a relaxed LP. We then use the LP's optimal solution to construct a *weighted bipartite graph* representing tasks on one side and

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

machine intervals on the other side and fractions of tasks completed in machine intervals as weighted edges. We then use an integral matching in this graph to construct a mapping of tasks to machine intervals. Finally, the tasks mapped to intervals of the same machine are packed in order and nonpreemptively using a greedy policy. We prove that this nonpreemptive algorithm has an approximation ratio better than 24. Furthermore, we show that the algorithm's solution is also a 24-approximation for the case that preemption on the same machine is allowed.

• **Preemption and Single-Machine Placement Set.** When preemption is allowed, and there is a specific machine for each task, we propose an algorithm with an improved approximation ratio of four. The algorithm first finds a proper ordering of jobs by solving a relaxed LP of our scheduling problem. Then, for each machine, it lists its tasks, with respect to the obtained ordering of jobs, and apply a simple greedy policy to pack tasks in the machine subject to its capacity. The methods of LP relaxation and list scheduling have been used in scheduling literature; however, the application and analysis of such techniques in presence of packing, placement, and synchronization is very different.

• **Empirical Evaluations.** We evaluate the performance of our preemptive and nonpreemptive algorithms compared with the prior approaches using a Google traffic trace (Wilkes 2011). We also present online versions of our algorithms that are suitable for handling dynamic job arrivals. Our 4− approximation preemptive algorithm outperforms PSRS (Schwiegelshohn 2004) and Tetris (Grandl et al. 2015) by up to 69% and 79%, respectively, when jobs' weights are determined using their priority information in the data set. Furthermore, our nonpreemptive algorithm outperforms JSQ-MW (Wang et al. 2016) and Tetris (Grandl et al. 2015) by up to 81% and 175%, respectively, under the same placement constraints. Because these algorithms do not consider all aspects of packing, synchronization, and data locality, we combined them with reasonable heuristics to enforce all the constraints in our settings.

2. Formal Problem Statement

2.1. Cluster and Job Model

Consider a collection of machines $\mathcal{M} = \{1, \ldots, M\}$, where machine i has capacity $m_i > 0$ on its available resource. We use $\mathcal{J} = \{1, \ldots, N\}$ to denote the set of existing jobs (stages) in the system that need to be served by the machines. Each job $j \in \mathcal{J}$ consists of a set of tasks $\mathcal{K}_j$, where we use (k, j) to denote task k of job j, $k \in \mathcal{K}_j$. Task (k, j) requires a specific amount a_{kj} of resource for the duration of its processing. Machine i can process multiple tasks at the same time, however, the sum of resource requirements of tasks running in machine i should not exceed its capacity m_i at any time.

2.2. Task Processing and Placement Constraint

Each task (k, j) can be processed on a machine from a specific set of machines $\mathcal{M}_{kj} \subseteq \mathcal{M}$. We refer to $\mathcal{M}_{kj}$ as the *placement set* of task (k, j). For generality, we let p^i_{kj} denote the processing time of task (k, j) on machine $i \in \mathcal{M}_{kj}$. Such placement constraints can model data locality. For example, we can set $\mathcal{M}_{kj}$ to be the set of machines that have task (k, j)'s data, and $p^i_{kj} = p_{kj}$, $i \in \mathcal{M}_{kj}$. Or, we can consider $\mathcal{M}_{kj}$ to be as large as $\mathcal{M}$, and incorporate the data transfer cost as a penalty in the processing time on machines that do not have the task's data.

Throughout the paper, we refer to a_{kj} as size or resource requirement of task (k, j), and to p^i_{kj} as its length, duration, or processing time on machine i. We also define the volume of task (k, j) on machine i as $v^i_{kj} = a_{kj}p^i_{kj}$. Without loss of generality, we assume processing times are nonnegative integers and duration of the smallest task is at least one. This can be done by defining a proper time unit (slot) and representing the task durations using integer multiples of this unit.

2.3. Synchronization Constraint

Tasks can be processed in parallel on their corresponding machines; however, a job is considered completed only when all of its tasks finish. Hence, using C_{kj} to denote the completion time of task (k, j), the completion time of job j, denoted by C_j, satisfies

$$C_j = \max_{k \in \mathcal{K}_j} C_{kj}. \tag{1}$$

Let $\mathbb{1}(i \in \mathcal{M}_{kj})$ be the indicator function, which is one if $i \in \mathcal{M}_{kj}$ and zero otherwise. Define

$$T = \max_{i \in \mathcal{M}} \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}_j} p^i_{kj} \mathbb{1}(i \in \mathcal{M}_{kj}), \tag{2}$$

which is clearly an upper bound on the time required for processing all the jobs. We define 0-1 variables $X^i_{kj}(t)$, $i \in \mathcal{M}$, $j \in \mathcal{J}$, $k \in \mathcal{K}_j$, $t \le T$, where $X^i_{kj}(t) = 1$ if task (k, j) is served at time slot t on machine i, and zero otherwise. We also make the following definition.

Definition 1 (Height of Machine i at Time t). The height of machine i at time t, denoted by $h_i(t)$, is the sum of resource requirements of the tasks running at time t in machine i, that is,

$$h_i(t) = \sum_{j \in \mathcal{J}, k \in \mathcal{K}_j} a_{kj} X^i_{kj}(t). \tag{3}$$

Given these definitions, a valid schedule $X^i_{kj}(t) \in \{0, 1\}$, $i \in \mathcal{M}$, $j \in \mathcal{J}$, $k \in \mathcal{K}_j$, $0 < t \le T$, must satisfy the following three constraints:

i. *Packing*: the sum of resource requirements of the tasks running in machine i at time t (i.e., tasks with $X^i_{kj}(t) = 1$) should not exceed machine i's capacity, that is, $h_i(t) \le m_i, \ \forall t \le T, \ \forall i \in \mathcal{M}$.

ii. *Placement*: each task at each time can get processed on at most a single machine selected from its feasible

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

placement set, that is, $\sum_{i\in\mathcal{M}_{kj}} X^i_{kj}(t) \le 1$, and $X^i_{kj}(t) = 0$ if $i \notin \mathcal{M}_{kj}$.

iii. *Processing*: each task must be processed completely. Noting that $X^i_{kj}(t)/p^i_{kj}$ is the fraction of task (k, j) completed on machine i in time slot t, we need $\sum_{i\in\mathcal{M}_{kj}}\sum_{t=1}^{T} X^i_{kj}(t)/p^i_{kj} = 1$.

2.4. Preemption and Migration

We consider three classes of scheduling policies. In a nonpreemptive policy, a task cannot be preempted (and hence cannot be migrated among machines) once it starts processing on its corresponding machine until it is completed. In a preemptive policy, a task may be preempted and resumed several times in the schedule, and we can further consider two subcases depending on whether migration of a task among machines is allowed or not. When migration is not allowed, the scheduler must assign each task (k, j) to one machine $i \in \mathcal{M}_{kj}$ on which the task is (preemptively or nonpreemptively) processed until completion.

2.5. Main Objective

Given positive weights w_j, $j \in \mathcal{J}$, our goal is to find valid nonpreemptive and preemptive (under with and without migrations) schedules of jobs (their tasks) in machines to minimize the sum of weighted completion times of jobs, that is,

$$\text{minimize} \sum_{j\in\mathcal{J}} w_j C_j. \tag{4}$$

The weights can capture different priorities for jobs. Clearly the case of equal weights reduces the problem to minimization of the average completion time.

Here, we use the three-field notation to specify our problems. Although we use some of the notations from the scheduling literature, we need to define new ones to capture all the constraints in our model. We consider the following problems:

- PRP|mgr|$\sum_j w_j C_j$
- PRP||$\sum_j w_j C_j$ and PRP|pmtn|$\sum_j w_j C_j$
- PDP|pmtn|$\sum_j w_j C_j$

In the first field of the notations, the first letter *P* stands for parallel and specifies the fact that the machines can process different tasks of a given job in parallel. The letter *R* means that the machines are unrelated, that is, a task has different processing times on different machines. The letter *D* stands for dedicated and shows that there is a dedicated machine for processing of each task. Finally, the last letter *P* stands for packing and shows that a machine can pack tasks subject to its capacity. In the second field, *pmts* and *mgr* indicate that processing of a task can be preempted and a task can migrate among machines, respectively. Finally, the objective function is specified in the third field.

3. Scheduling When Migration Is Allowed

We first consider the case that migration of tasks among machines is allowed. This is equivalent to PRP|mgr|$\sum_j w_j C_j$. In this case, we propose a preemptive algorithm, called SynchPack-1, with approximation ratio $(6+\epsilon)$ for any $\epsilon > 0$. We will use the construction ideas and analysis arguments for this algorithm to construct our preemptive and nonpreemptive algorithms when migration is prohibited in Section 4.

To describe SynchPack-1, we first present a relaxed linear program. We will utilize the optimal solution to this LP to schedule tasks in a preemptive fashion.

3.1. Relaxed Linear Program (LP1)

Recall that without loss of generality, the processing times of tasks are assumed to be integers (multiples of a time unit) and therefore $C_j \ge p^i_{kj} \ge 1$ for all $j \in \mathcal{J}$, $k \in \mathcal{K}_j$, and $i \in \mathcal{M}_{kj}$. We use interval indexed variables using geometrically increasing intervals (Lenstra et al. 1990, Queyranne and Sviridenko 2002, Qiu et al. 2015) to formulate a linear program for our problem.

Let $\epsilon > 0$ be a constant. We choose L to be the smallest integer such that $(1+\epsilon)^L \ge T$ (recall T in (2)). Subsequently, define

$$d_l = (1+\epsilon)^l, \quad \text{for} \quad l = 0, 1, \cdots, L, \tag{5}$$

and define $d_{-1} = 0$. We partition the time horizon into time intervals $(d_{l-1}, d_l]$, $l = 0, \ldots, L$. The length of the lth interval, denoted by Δ_l, is

$$\Delta_0 = 1, \ \Delta_l = \epsilon(1+\epsilon)^{l-1} \quad \forall l \ge 1. \tag{6}$$

We define z^{il}_{kj} to be the fraction of task (k, j) (*fraction of its required processing time*) that is processed in interval l on machine $i \in \mathcal{M}_{kj}$.

To measure completion time of job j, for each interval l, we define an integer variables x_{jl}, which is one if job j finishes in interval l and zero otherwise. Consider the following constraints, $\forall j \in \mathcal{J}$:

$$\sum_{l'=0}^{l} x_{jl'} \le \sum_{l'=0}^{l} \sum_{i\in\mathcal{M}_{kj}} z^{il'}_{kj}, \ k \in \mathcal{K}_j, \ l = 0, \ldots, L, \tag{7a}$$

$$\sum_{l=0}^{L} x_{jl} = 1, \ x_{jl} \in \{0, 1\}, \ l = 0, \cdots, L. \tag{7b}$$

Note that (7b) implies that only one of the variables $\{x_{jl}\}_{l=0}^{L}$ can be nonzero (equal to one). (7a) implies that x_{jl} can be one only for one of the intervals $l \ge l^\star$, where $l^\star$ is the interval in which the last task of job j finishes its processing. Now define

$$C_j = \sum_{l=0}^{L} d_{l-1} x_{jl} \ j \in \mathcal{J}. \tag{8}$$

If we can guarantee that $x_{jl^\star} = 1$ for $l^\star$ as defined previously, then C_j will be equal to the starting point $d_{l^\star-1}$ of that interval, and the actual completion time of job j

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

will be bounded above by $d_{l^*} = (1+\epsilon)C_j$, thus implying that C_j is a reasonable approximation for the actual completion time of job j. This can be done by minimizing the objective function in the following linear program:

$$\min \sum_{j\in\mathcal{J}} w_j C_j \quad (\mathbf{LP1}), \tag{9a}$$

$$\sum_{l=0}^{L} \sum_{i\in\mathcal{M}_{kj}} z_{kj}^{il} = 1,\ k\in\mathcal{K}_j, j\in\mathcal{J}, \tag{9b}$$

$$\sum_{l'=0}^{l} \sum_{i\in\mathcal{M}_{kj}} z_{kj}^{il'} p_{kj}^{i} \le d_l,\ k\in\mathcal{K}_j, j\in\mathcal{J}, l=0,\dots,L, \tag{9c}$$

$$\sum_{l'=0}^{l} \sum_{(k,j):i\in\mathcal{M}_{kj}} z_{kj}^{il'} p_{kj}^{i} a_{kj} \le m_i d_l,\ i\in\mathcal{M}, l=0,\dots,L, \tag{9d}$$

$$z_{kj}^{il} \ge 0,\ k\in\mathcal{K}_j, j\in\mathcal{J}, i\in\mathcal{M}_{kj}, l=0,\dots,L, \tag{9e}$$

$$\sum_{l'=0}^{l} x_{jl'} \le \sum_{l'=0}^{l} \sum_{i\in\mathcal{M}_{kj}} z_{kj}^{il'}, k\in\mathcal{K}_j, j\in\mathcal{J}, l=0,\dots,L, \tag{9f}$$

$$C_j = \sum_{l=0}^{L} d_{l-1} x_{jl},\ j\in\mathcal{J}, \tag{9g}$$

$$\sum_{l=0}^{L} x_{jl} = 1,\ x_{jl}\ge 0,\ l=0,\dots,L, j\in\mathcal{J}. \tag{9h}$$

Constraint (9b) means that each task must be processed completely. (9c) is because during the first l intervals, a task cannot be processed for more than d_l, the end point of interval l, which itself is because of requirement (ii) of Section 2. (9d) bounds the total volume of the tasks processed by any machine i in the first l intervals by $d_l \times m_i$. (9e) indicates that z variables must be nonnegative.

Constraints (9f), (9h), and (9g) are the relaxed version of (7a), (7b), and (8), respectively, where the integral constraint in (7b) has been relaxed to (9h). To give more insight, (9f) has the interpretation of keeping track of the fraction of the job processed by the end of each time interval, which is bounded from above by the fraction of any of its tasks processed by the end of that time interval. We should finish processing of all jobs as indicated by (9h). Also (9g) computes a relaxation of the job completion time C_j, as a convex combination of the intervals' left points, with coefficients x_{jl}.

3.2. Scheduling Algorithm: SynchPack – 1

In the following, a *task fraction* (k,j,i,l) of task (k,j) corresponding to interval l, is a task with size a_{kj} and duration $z_{kj}^{il} p_{kj}^{i}$ that needs to be processed on machine i.

The SynchPack-1 (*Synchronized Packing*-1) algorithm has three main steps:

Step 1: Solve (LP1). We first solve (LP1) and obtain the optimal solution of $\{z_{kj}^{il}\}$, which we denote by $\{\tilde{z}_{kj}^{il}\}$.

Step 2: Pack task fractions greedily to construct schedule $\mathcal{S}$. To schedule task fractions, we use a greedy list scheduling policy as follows:

Consider an ordered list of the task fractions such that task fractions corresponding to interval l appear before the task fractions corresponding to interval l', if $l<l'$. Task fractions within each interval and corresponding to different machines are ordered arbitrarily. Let t denote a time at which the algorithm makes some scheduling decision. The algorithm scans the list starting from the first task fraction and schedules task fraction (k,j,i,l) on machine i,if some fraction of task (k,j) is not already scheduled on some other machine at time t and machine i has sufficient capacity, that is, $h_i(t)+a_{kj}\le m_i$ (recall $h_i(t)$ in Definition 1). It then moves to the next task fraction in the list, repeats the same procedure, and so on. Upon completion of a task fraction, it preempts the task fractions corresponding to higher indexed intervals on all the machines if there is some unscheduled task fraction of a lower-indexed interval in the list. It then removes the completed task fraction(s) from the list, updates the remaining processing times of the task fractions in the list, and starts scheduling the updated list. The set of times at which scheduling decisions are made consists of time 0 and task fractions' completion times. This greedy list scheduling algorithm schedules task fractions in a preemptive fashion. We refer to the constructed schedule as $\mathcal{S}$.

As an illustration, Figure 1 shows execution of Step 2 in a system with three machines and three jobs.

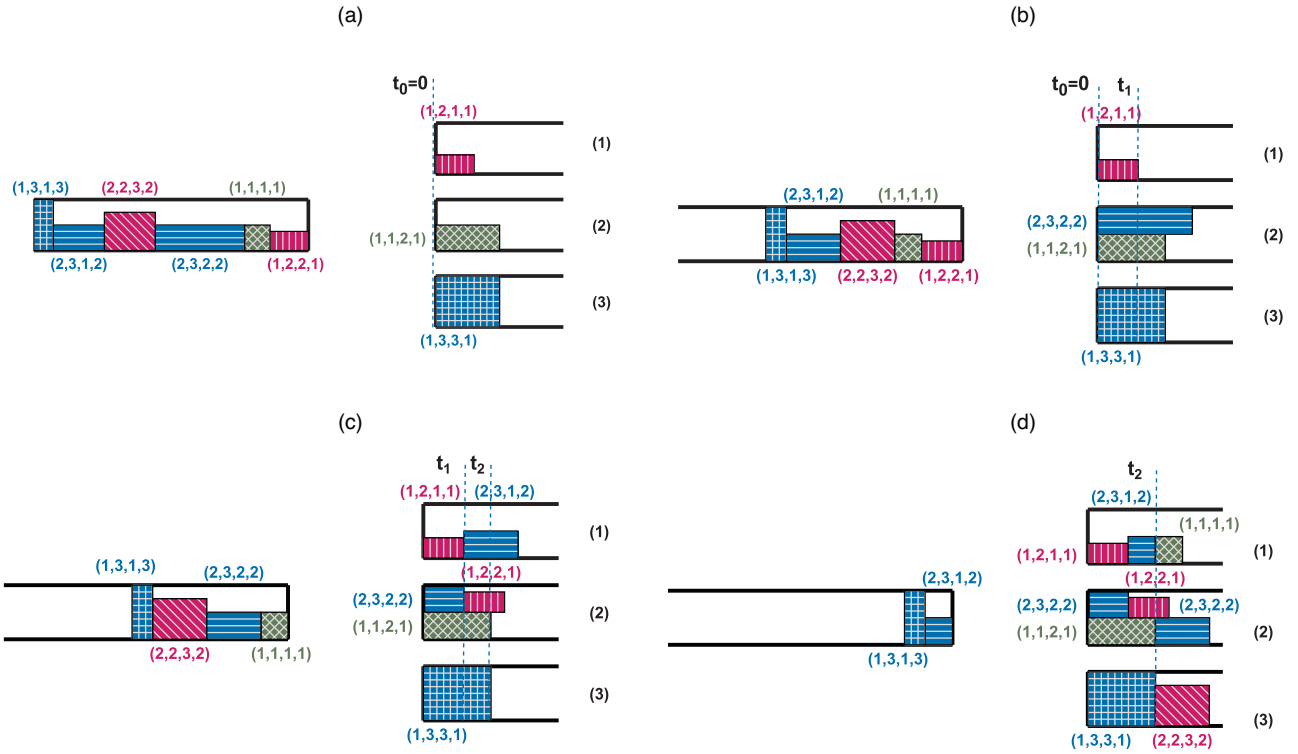
Step 3: Apply Slow-Motion technique to construct schedule $\bar{\mathcal{S}}$. Unfortunately, we cannot bound the value of Objective Function (9a) for schedule $\mathcal{S}$, because completion times of some jobs in $\mathcal{S}$ can be very long compared with the completion times returned by (LP1).[1]

Therefore, we construct a new feasible schedule $\bar{\mathcal{S}}$, by stretching $\mathcal{S}$, for which we can bound the value of its objective function. This method is referred to as *Slow-Motion* technique (Schulz and Skutella 1997). Let $\tilde{Z}_{kj}^{i} = \sum_{l=0}^{L} \tilde{z}_{kj}^{il}$ denote the total fraction of task (k,j) that is scheduled in machine i according to the optimal solution to (LP1). We refer to $\tilde{Z}_{kj}^{i}$ as the *total task fraction* of task (k,j) on machine i. The Slow-Motion technique works by choosing a parameter $\lambda\in(0,1]$ randomly drawn according to the probability density function $f(\lambda)=2\lambda$. It then stretches schedule $\mathcal{S}$ by a factor $1/\lambda$. If a task is scheduled in $\mathcal{S}$ during an interval $[\tau_1,\tau_2)$, the same task is scheduled in $\bar{\mathcal{S}}$ during $[\tau_1/\lambda,\tau_2/\lambda)$ and *the machine is left idle if it has already processed its total task fraction* $\tilde{Z}_{kj}^{i}$ completely. We may also shift back future tasks' schedules as far as the machine capacity allows and placement constraint is respected.

Figure 2 shows the execution of this step on the example of Figure 1 for $\lambda = 1/2$.

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

Figure 1. (Color online) Example for Execution of Step 2 of SynchPack-1 for Three Jobs in a System with Three Machines



Notes. Different tasks of a job have the same color and different patterns. Task fraction $(1,2,2,1)$, which is at the head of the list in (a), cannot get scheduled on machine 2 as task fraction $(1,2,1,1)$ (of the same task $(1,2)$) is already scheduled on machine 1. At time t_1, task fraction $(1,2,1,1)$ is finished processing as shown in (b). At this time, while task fraction $(2,3,2,2)$ is running on machine 2 (whose corresponding interval is 2), two task fractions, namely $(1,2,2,1)$ and $(1,1,1,1)$ (whose corresponding intervals are 1), have remained unscheduled in the list. Therefore, task fraction $(2,3,2,2)$ is preempted and its remaining duration is updated. Then, the algorithm scans the list and schedules the task fractions as shown in (c). The next time that a completion occurs is denoted by t_2. (d) Schedule at this time. The rest of the schedule can be determined in a similar fashion. (a) A list of task fractions is given. The first three task fractions in the list are already scheduled on the machines at time $t_0 = 0$. (b) Because of placement constraint, task fractions $(1,2,2,1)$ and $(1,1,1,1)$ cannot get scheduled. However, machine 2 can accommodate task fraction $(2,3,2,2)$. (c) At t_1 task fraction $(1,2,1,1)$ completes and task fraction $(2,3,2,2)$ is preempted. Task fractions $(1,2,2,1)$ and $(2,3,1,2)$ are scheduled. (d) Both task fractions $(1,1,2,1)$ and $(1,3,3,1)$ complete, and task fraction $(2,3,1,2)$ is preempted at time t_2. Then task fractions $(1,1,1,1)$, $(2,3,2,2)$, and $(2,2,3,2)$ are scheduled.

A pseudocode for SynchPack-1 can be found in Online Appendix G. The obtained algorithm is a randomized algorithm; however, we will show in Online Appendix C how we can de-randomize it to get a deterministic algorithm.

3.3. Performance Guarantee

We now analyze the performance of SynchPack-1. The result is stated by the following proposition.

Theorem 1. *For any $\epsilon > 0$, the sum of weighted completion times of jobs, for the problem of parallel-task job scheduling with packing and placement constraints, under* SynchPack-1, *is at most $(6+\epsilon) \times OPT$.*

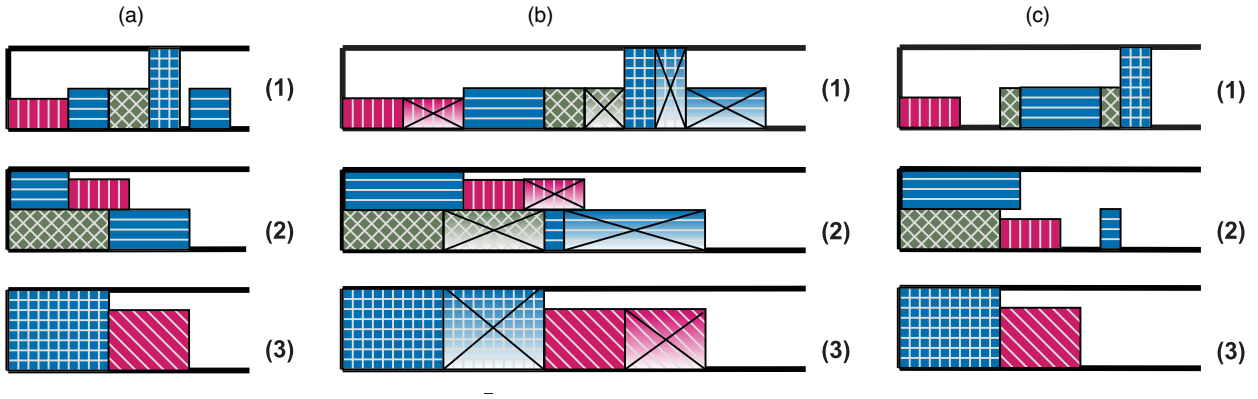
The rest of the section is devoted to the proof of Theorem 1. We use $\tilde{C}_j$ to denote the optimal solution to (LP1) for completion time of job $j \in \mathcal{J}$. The optimal objective value of (LP1) is a lower bound on the optimal value of our scheduling problem as stated in the following lemma whose proof is provided in Online Appendix B.1.

Lemma 1. *The optimal objective value of (LP1) is a lower bound on the optimal value of our scheduling problem, i.e., $\sum_{j=1}^{N} w_j \tilde{C}_j \le \sum_{j=1}^{N} w_j C_j^{\star} = OPT$.*

Constraint (9d) bounds the volume of all the task fractions corresponding to the first l intervals on machine i by $d_l \times m_i$. However, the (LP1)'s solution does not directly provide a feasible schedule as task fractions of the same task on different machines might overlap during the same interval, and machines' capacity constraints might be also violated as Constraint (9d) in (LP1) bounds the total volume of the processed tasks and ignores their sizes and durations. Next, we show under the greedy list scheduling policy (Step 2 in SynchPack-1), the completion time of task fraction

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

Figure 2. (Color online) Example for Execution of Slow-Motion Technique in Step 3 of SynchPack-1



Notes. (a) Final schedule of the example in Figure 1 is shown. (b) Result after applying Slow-Motion with $\lambda = 1/2$. If a machine has already processed total task fraction of a task completely, it is left idle. For instance, consider the task fraction (2,3,2,2) on machine 2, that is the task with horizontal strips. Some portion of its schedule in the second part is shadowed and crossed and machine 2 is left idle, because machine 2 has already processed this task fraction for the total time that it does originally in (a). (c) Result after shifting back future tasks' schedules while respecting the constraints. For instance, see the task fraction with vertical strips (i.e., (1, 2, 2, 1) on machine 2 and part of the task fraction with crossed pattern (i.e., (1, 1, 1, 1)) on machine 1. The idle times on the machines are left blank in (c). This last action (shifting back future tasks' schedules) is optional. (a) Schedule $\mathcal{S}$ for the example of Figure 1. (b) Schedule $\bar{\mathcal{S}}$ for stretch factor $\lambda = 1/2$. The machines are left idle in the shadowed crossed parts. (c) Final schedule after shifting back future tasks' schedules while respecting the constraints.

(k,j,i,l) is bounded from above by $3 \times d_l$, that is, we need a factor of three to guarantee a feasible schedule.

Lemma 2. *Let τ_l denote the time that all the task fractions (k,j,i,l'), for $l' \le l$, are completed in schedule $\mathcal{S}$. Then, $\tau_l \le 3d_l$.*

Proof. Consider the nonzero task fractions (k,j,i,l'), $i \in \mathcal{M}, l' \le l$ (according to an optimal solution to (LP1)). Without loss of generality, we normalize the processing times of task fractions to be positive integers, by defining a proper time unit and representing the task durations using integer multiples of this unit. Let D_l and T_l be the value of d_l and τ_l using the new unit. Let $i^\star$ denote the machine that schedules the last task fraction among the nonzero task fractions of the first l intervals. Note that T_l is the time that this task fraction completes. If $T_l \le D_l$, then $T_l \le 3D_l$, and the lemma is proved. Hence, consider the case that $T_l > D_l$.

Define $h_{il}(t)$ to be the height of machine i at time t in schedule $\mathcal{S}$ considering only the task fractions of the first l intervals. First, we note that

$$\sum_{l'=0}^{l} \sum_{(k,j):i\in\mathcal{M}_{kj}} z_{kj}^{il'} p_{kj}^{i} a_{kj} \overset{(a)}{=} \sum_{t=1}^{T_l} h_{il}(t) \overset{(b)}{\le} m_i D_l, \quad \forall i \in \mathcal{M}. \tag{10}$$

Using the definition of $h_{il}(t)$, the right-hand side of equality (a) is the total volume of task fractions corresponding to the first l intervals that are processed during the interval $(0, T_l]$ on machine i, which is the left-hand side. Furthermore, inequality (b) is by Constraint (9d).

Let $S_{il}(\theta)$ denote the set of tasks whose some task fraction is running at time θ, $\theta \in \{1,\dots,T_l\}$, on machine i. Consider machine $i^\star$. Consider machine $i^\star$. We construct a bipartite graph G=(U ∪ V, E)[2] as follows. With a slight abuse of notations, for each time $\theta \in \{1,\dots,T_l\}$, we consider a node θ, and define $V = \{\theta | 1 \le \theta \le T_l - D_l\}$, and $U = \{\theta | T_l - D_l + 1 \le \theta \le T_l\}$. For any $s \in U$ and $t \in V$, we add an edge (s, t) if $h_{i^\star l}(s) + h_{i^\star l}(t) \ge m_{i^\star}$. This implies that if $h_{i^\star l}(s) + h_{i^\star l}(t) < m_{i^\star}$, then there is no edge between s and t, and we can write

$$\left(\bigcup_{i\in\mathcal{M}} S_{il}(s)\right) \backslash \left(\bigcup_{i\in\mathcal{M}} S_{il}(t)\right) = \emptyset. \tag{11}$$

This is because otherwise SynchPack-1 would have scheduled the task(s) in $S_{i^\star l}(s)$ at time t (note that $t < s$).

Let $|\cdot|$ denote set cardinality (size). For any set of nodes $\tilde{U} \subseteq U$, we define set of its neighbor nodes as $N_{\tilde{U}} = \{t \in V | \exists s \in \tilde{U} : (s,t) \in E\}$. There are $T_l - D_l - |N_{\tilde{U}}|$ nodes in V, which do not have any edge to some node in $\tilde{U}$. We consider two cases.

Case i: There exists a set $\tilde{U}$ for which $|N_{\tilde{U}}| < |\tilde{U}|$. Consider a node $s \in \tilde{U}$ and a task with duration p running at time slot s. Let p_U denote the amount of time that this task is running on time slots of set U. Note that $p_U \ge 1$. By Equation (11), a task that is running at time s is also running at $T_l - D_l - |N_{\tilde{U}}|$ many other time slots whose corresponding nodes are in V.

$$p = T_l - D_l - |N_{\tilde{U}}| + p_U \le D_l,$$

where the inequality is by Constraint (9c). Therefore,

$$T_l \le 2D_l + |N_{\tilde{U}}| - p_U < 2D_l + |\tilde{U}| \le 3D_l.$$

Case ii: For any $\tilde{U} \subseteq U$, $|\tilde{U}| \le |N_{\tilde{U}}|$. Hence, $|V| \ge |U|$, which implies that $T_l \ge 2D_l$. Furthermore, Hall's theorem (Hall 1935) states that a perfect matching[3] of nodes in U to nodes in V always exists in G in this case. The

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

existence of such a matching then implies that any time slot $s \in (T_l - D_l, T_l]$ can be matched to a time slot $t_s \in (0, T_l - D_l]$ and $h_{i^\star l}(s) + h_{i^\star l}(t_s) \geq m_{i^\star}$. This implies that

$$\sum_{s \in U} (h_{i^\star l}(s) + h_{i^\star l}(t_s)) \geq m_{i^\star} D_l \overset{(c)}{\geq} \sum_{t=1}^{T_l} h_{i^\star l}(t), \tag{12}$$

where inequality (c) is by Equation (10). From this, one can conclude that no nonzero task fraction $(k, j, i^\star, l')$, $i^\star, l' \leq l$ is processed at time slots $V' = V \backslash \cup_{s \in U} \{t_s\}$. This is because the right-hand side of inequality (c) is the total amount of task fractions that is processed up to time T_l. Hence, $V' = \emptyset$, because otherwise SynchPack-1 would have scheduled some of the tasks running at time slots of set U at V'. We then can conclude that $T_l = 2D_l < 3D_l$. This completes the proof. □

Recall that schedule $\bar{\mathcal{S}}$ is formed by stretching schedule $\mathcal{S}$ by factor $1/\lambda$. Let $\bar{C}_j^\lambda$ denote the completion time of job j in $\bar{\mathcal{S}}$. Next, we need to relate $\bar{C}_j^\lambda$ and $\tilde{C}_j$, the optimal solution to (LP1) for completion time of job j. For this purpose, we first make the following definition regarding schedule $\mathcal{S}$.

Definition 2. We define $C_j(\alpha)$, for $0 < \alpha \leq 1$, to be the time at which α-fraction of job j is completed in schedule $\mathcal{S}$ (i.e., at least α-fraction of each of its tasks has been completed.).

The following lemma shows the relationship between $C_j(\alpha)$ and $\tilde{C}_j$. The proof is provided in Online Appendix B.2.

Lemma 3. *The following inequality holds,* $\int_{\alpha=0}^{1} C_j(\alpha) d\alpha \leq 3(1+\epsilon)\tilde{C}_j$.

Now, we can show that the following lemma holds.

Lemma 4. *We can bound the expected completion time of job j in $\bar{S}$ as follows,* $\mathbb{E}[\bar{C}_j^\lambda] \leq 6(1+\epsilon)\tilde{C}_j$.

Proof. The proof is based on Lemma 3 and taking expectation with respect to probability density function of λ. The details can be found in Online Appendix B.3. □

In constructing $\bar{\mathcal{S}}$, we may shift scheduling time of some of the tasks on each machine to the left and construct a better schedule. Nevertheless, we have the performance guarantee of Theorem 1 even without this shifting.

Proof of Theorem 1. Let C_j denote the completion time of job j under SynchPack-1. Then

$$\mathbb{E}\left[\sum_{j \in \mathcal{J}} w_j C_j\right] \leq \mathbb{E}\left[\sum_{j \in \mathcal{J}} w_j \bar{C}_j^\lambda\right] \overset{(a)}{\leq} 6(1+\epsilon) \sum_{j \in \mathcal{J}} w_j \tilde{C}_j \overset{(b)}{\leq} 6(1+\epsilon) \sum_{j \in \mathcal{J}} w_j C_j^\star,$$

where (a) is by Lemma 4, and (b) is by Lemma 1. In Online Appendix C, we discuss how to derandomize the random choice of $\lambda \in (0, 1]$, which is used to construct schedule $\bar{\mathcal{S}}$ from schedule $\mathcal{S}$. Therefore, the proof is complete. □

4. Scheduling When Migration Is Not Allowed

The algorithm in Section 3 is preemptive, and tasks can be migrated across the machines in the same placement set. Implementing such an algorithm can be complex and costly in practice. In this section, we consider the case that migration of tasks among machines is not allowed. We propose a nonpreemptive scheduling algorithm for this case. Using the three-field notation, this case is represented by PRP$||\sum_j w_j C_j$. We also show that its solution provides a bounded solution for the case that preemption of tasks (in the same machine, without migration) is allowed (PRP|pmtn$|\sum_j w_j C_j$).

Our algorithm is based on a relaxed LP that is very similar to (LP1) of Section 3; however, a different constraint is used to ensure that each task is scheduled entirely by the end point of some time interval of a machine. Next, we introduce this LP and describe how to generate a nonpreemptive schedule based on its solution.

4.1. Relaxed Linear Program (LP2)

We partition the time horizon into intervals $(d_{l-1}, d_l]$ for $l = 0, \ldots, L$, as defined in (5) by replacing ϵ by one. Define 0-1 variable z_{kj}^{il} to indicate whether task (k, j) is completed on machine i by the end point of interval l, that is, by d_l. The interpretation of variables z_{kj}^{il} is slightly different from their counterparts in (LP1). By relaxing integrality of z variables, we formulate (LP2):

$$\min \sum_{j \in \mathcal{J}} w_j C_j \quad (\mathbf{LP2}), \tag{13a}$$

$$z_{kj}^{il} = 0 \text{ if } p_{kj}^i > d_l, \quad j \in \mathcal{J},\ k \in \mathcal{K}_j,\ i \in \mathcal{M}_{kj},\ l = 0, \ldots, L, \tag{13b}$$

$$\text{Constraints (9b)–(9h).} \tag{13c}$$

Constraint (13b) allows z_{kj}^{il} to be positive only if the end point of the l-th interval is at least as long as task (k, j)'s processing time on machine $i \in \mathcal{M}_{kj}$. We would like to emphasize that this is a valid constraint for both the preemptive and nonpreemptive cases when migration is not allowed. We will see shortly how this constraint helps us construct our nonpreemptive algorithm. We interpret *fractional* values of z_{kj}^{il} as the fraction of task (k, j) that is processed in interval l of machine i (as in Section 3).

4.2. Scheduling Algorithm: SynchPack – 2

Our nonpreemptive algorithm, which we refer to as SynchPack-2, has three main steps:

Step 1: Solve (LP2). We first solve the linear program (LP2) to obtain the optimal solution of $\{z_{kj}^{il}\}$ denoted by $\{\tilde{z}_{kj}^{il}\}$.

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

Step 2: Apply Slow-Motion. Before constructing the actual schedule of tasks, the algorithm applies the Slow-Motion technique (see Section 3.2). We pause here to clarify the connection between $\tilde{z}^{il}_{kj}$ and those obtained after applying Slow-Motion, which we denote by $\bar{z}^{il}_{kj}$.

Recall that $\tilde{z}^{il}_{kj}$ is the fraction of task (k, j) that is scheduled in interval l of machine i in the optimal solution to (LP2), and Δ_l is the length of the lth interval. Also, recall that $\tilde{Z}^{i}_{kj} = \sum_{l=0}^{L} \tilde{z}^{il}_{kj}$ is the total task fraction to be scheduled on machine i corresponding to task (k, j). Similarly, we define $\bar{\Delta}_l$ and $\bar{d}_l$ to be the length and the end point of the lth interval after applying the Slow-Motion using a stretch parameter $\lambda \in (0, 1]$, respectively. Therefore,

$$\bar{\Delta}_l = \frac{\Delta_l}{\lambda}, \ \bar{d}_l = \frac{d_l}{\lambda}. \tag{14}$$

Furthermore, we define $\bar{z}^{il}_{kj}$ to be the fraction of task (k, j) to be scheduled during the lth interval on machine i after applying Slow-Motion. Then it holds that

$$\bar{z}^{il}_{kj} = \begin{cases} \dfrac{\tilde{z}^{il}_{kj}}{\lambda}, & \text{if } \displaystyle\sum_{l'=0}^{l} \frac{\tilde{z}^{il'}_{kj}}{\lambda} < \tilde{Z}^{i}_{kj} \\ \max\left\{0, (\tilde{Z}^{i}_{kj} - \displaystyle\sum_{l'=0}^{l-1} \frac{\tilde{z}^{il'}_{kj}}{\lambda})\right\}, & \text{otherwise.} \end{cases} \tag{15}$$

To see (15), note that in Slow-Motion, both variables and intervals are stretched by factor $1/\lambda$, and after stretching, the machine is left idle if it has already processed its total task fraction completely. Hence, as long as $\tilde{Z}^{i}_{kj}$ fraction of task (k, j) is not completely processed by the end of the lth interval in the stretched solution, it is processed for $\tilde{z}^{il}_{kj} p^{i}_{kj}/\lambda$ amount of time in the lth interval of length $\bar{\Delta}_l = \Delta_l/\lambda$. Hence, $\bar{z}^{il}_{kj} = \tilde{z}^{il}_{kj}/\lambda$. Now suppose $l^\star$ is the first interval for which $\sum_{l'=0}^{l^\star} \tilde{z}^{il'}_{kj}/\lambda \geq \tilde{Z}^{i}_{kj}$. Then, the remaining processing time of task (k, j) to be scheduled in the $l^\star$ th interval of machine i in the stretched schedule is $p^{i}_{kj}(\tilde{Z}^{i}_{kj} - \sum_{l'=0}^{l^\star-1} \tilde{z}^{il'}_{kj}) = p^{i}_{kj}(\tilde{Z}^{i}_{kj} - \sum_{l'=0}^{l^\star-1} \tilde{z}^{il'}_{kj}/\lambda) > 0$. Therefore, the second part of (15) holds for $l^\star$ and for intervals $l > l^\star$, $\bar{z}^{il}_{kj}$ will be zero, because $\tilde{Z}^{i}_{kj} - \sum_{l'=0}^{l-1} \tilde{z}^{il'}_{kj}/\lambda \leq 0$. Observe that $\sum_{i \in \mathcal{M}_{kj}} \sum_{l=0}^{L} \bar{z}^{il}_{kj} = 1$.

Step 3: Construct a nonpreemptive schedule. According to variables $\bar{z}^{il}_{kj}$, a task possibly is set to get processed in different intervals and machines. The last step of SynchPack-2 is the procedure of constructing a nonpreemptive schedule using these variables. This procedure involves two substeps: (1) *mapping of tasks to machine-intervals* and (2) *nonpreemptive scheduling of tasks mapped to each machine-interval using a greedy scheme*. We now describe each of these substeps in detail.

Substep 3.1: Mapping of tasks to machine intervals. For each task (k, j), the algorithm uses a mapping procedure to find *a machine and an interval in which it can schedule the task entirely in a nonpreemptive fashion*. The mapping procedure is based on constructing a weighted bipartite graph $\mathcal{G} = (U \cup V, E)$, followed by an *integral matching* of nodes in U to nodes in V on edges with nonzero weights, as described here:

i. *Construction of Graph* $\mathcal{G} = (U \cup V, E)$: For each task (k, j), $j \in \mathcal{J}$, $k \in \mathcal{K}_j$, we consider a node in U. Therefore, there are $\sum_{j \in \mathcal{J}} |\mathcal{K}_j|$ nodes in U. Furthermore, $V = \bigcup_{i \in \mathcal{M}} V_i$, where V_i is the set of nodes that we add for machine i to represent intervals. To construct graph $\mathcal{G}$, we start from the first machine, say machine i, and sort tasks in nonincreasing order of their volume $v^{i}_{kj} = a_{kj} p^{i}_{kj}$ in machine i. Let N_i denote the number of tasks on machine i with nonzero volumes. Without loss of generality, suppose

$$v^{i}_{k_1 j_1} \geq v^{i}_{k_2 j_2} \geq \ldots v^{i}_{k_{N_i} j_{N_i}} > 0. \tag{16}$$

For each interval l, we consider $\lceil \bar{z}^{il} \rceil = \lceil \sum_{j \in J} \sum_{k \in \mathcal{K}_j} \bar{z}^{il}_{kj} \rceil$ (recall the definition of $\bar{z}^{il}_{kj}$ in (15)) consecutive nodes in V_i, which we call *copies of interval* l.

Starting from the first task in the ordering (16), we draw edges from its corresponding node in U to the interval copies in V_i in the following manner. Assume we reach at task (k, j) in the process of adding edges. For each interval l, if $\bar{z}^{il}_{kj} > 0$, first set $R = \bar{z}^{il}_{kj}$. Consider the first copy of interval l for which the total weight of its current edges is strictly less than one and set W to be its total weight. We draw an edge from the node of task (k, j) in U to this copy node in V_i and assign a weight equal to $\min\{R, 1 - W\}$ to this edge. Then we update $R \leftarrow R - \min\{R, 1 - W\}$, consider the next copy of interval l, and apply the same procedure, until $R = 0$ (or equivalently, the sum of edge weights from node (k, j) to copies of interval l becomes equal to $\bar{z}^{il}_{kj}$). We use w^{ilc}_{kj} to denote the weight of edge that connects task (k, j) to copy c of interval l of machine i, and if there is no such edge, $w^{ilc}_{kj} = 0$. We then move to the next machine and apply the similar procedure and so on. See Figure 3 for an illustrative example.

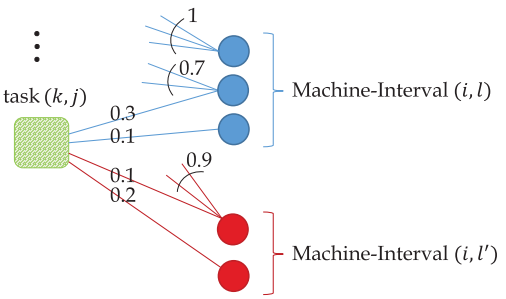
In $\mathcal{G}$, the weight of any node $u \in U$ (the sum of weights of its edges) is equal to one (because $\sum_{l=0}^{L} \bar{z}^{il}_{kj} = 1$, for any task (k, j)), whereas the weight of any node $v \in V$ is at most one.

ii. *Integral Matching:* Finally, we find an integral matching on the nonzero edges of $\mathcal{G}$, such that each nonzero task is matched to some interval copy. As we will show shortly in Section 4.3, we can always find an integral matching of size $\sum_{j \in \mathcal{J}} |\mathcal{K}_j|$, the total number of tasks, in $\mathcal{G}$, in polynomial time, in which each task is matched to a copy of some interval.

A pseudocode for the mapping procedure can be found in Online Appendix H.

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

Figure 3. (Color online) Illustrative Example for Construction of Graph $\mathcal{G}$ in Substep 3.1



Notes. Task (k, j) requires $\bar{z}_{kj}^{il} = 0.4$ and $\bar{z}_{kj}^{il'} = 0.3$. When we reach at task (k, j), the total weight of the first copy of interval l is 1 and that of its second copy is 0.7. Also, the total weight of the first copy of interval l' is 0.9. Hence, the procedure adds two edges to copies of interval l with weights 0.3 and 0.1 and two edges to copies of interval l' with weights 0.1 and 0.2.

Substep 3.2: Greedy packing of tasks in machine intervals. We use a greedy packing to schedule all the tasks that are mapped to a machine-interval *non-preemptively*. More precisely, on each machine, the greedy algorithm starts from the first interval and considers an arbitrary ordered list of its corresponding tasks. Starting from the first task, the algorithm schedules it, and moves to the second task. If the machine has sufficient capacity, it schedules the task; otherwise, it checks the next task and so on. Once it is done with all the tasks of the first interval, it considers the second interval, applies the similar procedure, and so on. We may also shift back future tasks' schedules as far as the machine capacity allows.

This greedy algorithm is simpler than the one described in Section 3, because it does not need to consider requirement (ii) of Section 2 as here each task only appears in one feasible machine.

As we prove in the next section, we can bound the total volume of tasks mapped to interval l on machine i in the mapping phase by $m_i\bar{\Delta}_l$. Furthermore, by Constraint (13b) and the fact that the integral matching in Substep 3.1 was constructed on nonzero edges, the processing time of any task mapped to an interval is not greater than the interval's end point, which is twice the interval length. Hence, we can bound the completion time of each job and find the approximation ratio that our algorithm provides. A pseudocode for the SynchPack-2 algorithm can be found in Online Appendix H.

4.3. Performance Guarantee

In this section, we analyze the performance of our nonpreemptive algorithm SynchPack-2. The main result of this section is as follows.

Theorem 2. *The scheduling algorithm* SynchPack-2 *is a 24-approximation algorithm for the problem of parallel-task jobs scheduling with packing and placement constraints, when preemption and migration is not allowed.*

Because the constraints of (LP2) also hold for the preemptive case when migration is not allowed, the optimal solution of this case is also lower bounded by the optimal solution to the LP. Therefore, the algorithm' solution is also a bounded solution for the case that preemption is allowed (while still migration is not allowed).

Corollary 1. *The scheduling algorithm* SynchPack-2, *in Section 4.2, is a 24-approximation algorithm for the problem of parallel-task jobs scheduling with packing and placement constraints, when preemption is allowed and migration is not.*

The rest of this section is devoted to the proof of Theorem 2. With a minor abuse of notation, we use $\tilde{C}_{kj}$ and $\tilde{C}_j$ to denote the completion time of task (k, j) and job j, respectively, in the optimal solution to (LP2). Also, let $C^\star_{kj}$ and $C^\star_j$ denote the completion time of task (k, j) and job j, respectively, in the optimal nonpreemptive schedule. We can bound the optimal value of (LP2) as stated here. The proof is provided in Online Appendix D.1.

Lemma 5. *The following inequality holds,* $\sum_{j=1}^{N} w_j\tilde{C}_j \le \sum_{j=1}^{N} w_j C^\star_j = OPT$.

Definition 3. Given $0 < \alpha \le 1$, define $\hat{C}_j(\alpha)$ to be the starting point of the earliest interval l for which $\alpha \le \tilde{x}_{jl}$, where $\tilde{x}_{jl}$ is solution of (LP2).

Note that $\hat{C}_j(\alpha)$ is slightly different from Definition 2, because we do not construct an actual schedule yet. We then have the following corollary that is a counterpart of Lemma 3. See Online Appendix D.2 for the proof.

Corollary 2. *The following equality holds,* $\int_{\alpha=0}^{1} \hat{C}_j(\alpha)d\alpha = \tilde{C}_j$.

Consider the mapping procedure where we construct bipartite graph $\mathcal{G}$ and match each task to a copy of some machine-interval. We state a lemma that ensures that indeed we can find an integral (i.e., zero or one) matching in $\mathcal{G}$. The proof can be found in Online Appendix D.3.

Lemma 6. *Consider graph $\mathcal{G}$ constructed in the mapping procedure. There exists an integral matching on the nonzero edges of $\mathcal{G}$ in which each task is matched to some interval copy. Furthermore, this matching can be found in polynomial time.*

Let $\mathcal{V}_{il}$ denote the total volume of the tasks mapped to all the copies of interval l of machine i. The following lemma bounds $\mathcal{V}_{il}$, whose proof is provided in Online Appendix D.4.

Lemma 7. *For any machine-interval (i, l), we have*

$$\mathcal{V}_{il} \le \bar{d}_l m_i + \sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}_j} v_{kj}^i \bar{z}_{kj}^{il}. \tag{17}$$

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

The second term on the right side of (17) can be bounded by $\bar{d}_l m_i$, which results in the inequality $\mathcal{V}_{il} \le 2\bar{d}_l m_i$. However, the provided bound is tighter and allows us to prove a better bound for the algorithm. We next show that, using the greedy packing algorithm, we can schedule all the tasks of an interval l in a bounded time.

In the case of packing single tasks in a single machine, the greedy algorithm by Garey and Graham (1975) is known to provide a two-approximation solution for minimizing makespan. The situation is slightly different in our setting because we require bounding of the completion time of the last task as a function of the total volume of tasks, when the maximum duration of all tasks in each interval is bounded. We state the following lemma and its proof in Online Appendix D.5 for completeness.

Lemma 8. *Consider a machine with a capacity of one and a set of tasks $J = \{1, 2, \ldots, n\}$. Suppose each task j has size $a_j \le 1$, processing time $p_j \le 1$, and $\sum_{j\in J} a_j p_j \le v$. Then, we can schedule all the tasks within the interval $(0, 2\max\{1, v\}]$ using the greedy algorithm.*

Now consider a machine-interval (i, l). Lemma 7 bounds the total volume of tasks while Constraint (13b) ensures that duration of each task is less than d_l. Thus, by applying Lemma 8 on the normalized instance, in which size and length of tasks are normalized by m_i and d_l, respectively, we guarantee that we can schedule all the task within a time interval of length $2d_l + 2\sum_{j\in\mathcal{J}}\sum_{k\in\mathcal{K}_j} v^i_{kj}\bar{z}^{il}_{kj}/m_i$. Moreover, the factor of two is tight as stated in the following lemma whose proof can be found in Online Appendix D.6.

Lemma 9. *We need an interval of length at least $2\max(1, v)$ to be able to schedule any list of tasks as in Lemma 8 using any algorithm.*

Hence, Lemmas 8 and 9 imply that applying the greedy algorithm to schedule the tasks of each machine interval provides a tight bound with respect to the total volume of tasks in that machine interval. Let C_{kj} denote the completion time of task (k, j) under SynchPack-2. Then we have the following lemma, whose proof can be found in Online Appendix D.7.

Lemma 10. *Suppose that task (k, j) is mapped to the lth interval of machine i at the end of Substep 3.1. Then, $C_{kj} \le 6\bar{d}_l$.*

Proof of Theorem 2. Let l denote the end point of the interval in which task (k, j) has the last nonzero fraction according to $\bar{z}^{il}_{kj}$. Then,

$$\bar{d}_l = 2^l/\lambda \overset{(\star)}{\le} 2\hat{C}_j(\lambda)/\lambda. \tag{18}$$

First note that ϵ is replaced by one in Equation (5). Furthermore, inequality $(\star)$ follows from the definition of $\hat{C}_j(\lambda)$ (Definition 3), and the fact that d_ls are multiplied by $1/\lambda$. Therefore, $\hat{C}_j(\lambda)/\lambda$ is the start point of the interval in which job j is completed, and, accordingly, $2\hat{C}_j(\lambda)/\lambda$ is the end point of that interval. Thus, $2^l/\lambda$, the end point of the interval in which task (k, j) is completed, has to be at most $2\hat{C}_j(\lambda)/\lambda$, the end point of the interval in which job j is completed.

Let C_{kj} and C_j be the completion time of task (k, j) and job j under SynchPack-2. Recall that in the mapping procedure, we only map a task to some interval l' in which part of the task is assigned to that interval after Slow-Motion applied (in other words, $\bar{z}^{il'}_{kj} > 0$). Thus, task (k, j) that has its last nonzero fraction in interval l (by our assumption) is mapped to some interval $l' \le l$, because $\bar{z}^{il''}_{kj} = 0$ for intervals $l'' > l$. Suppose task (k_j, j) is the last task of job j and finishes in interval l_j in our nonpreemptive schedule. Then, by Lemma 10 and Equation (14), we have $C_j = C_{i_j j} \le 6\bar{d}_l = \frac{6}{\lambda}2^{l_j}$. Recall that $\tilde{C}_j$ denotes the completion time of job j in an optimal solution of (LP2). Hence,

$$\mathbb{E}\left[\sum_{j\in\mathcal{J}} w_j C_j\right] \le \mathbb{E}\left[\sum_{j\in\mathcal{J}} w_j \frac{6}{\lambda} 2^{l_j}\right] \overset{(a)}{\le} 12 \times \mathbb{E}\left[\sum_{j\in\mathcal{J}} w_j \hat{C}_j(\lambda)/\lambda\right]$$
$$\overset{(b)}{=} 12 \times \sum_{j\in\mathcal{J}} w_j \int_{\lambda=0}^{1} \frac{\hat{C}_j(\lambda)}{\lambda} 2\lambda d\lambda \overset{(c)}{\le} 24 \times \sum_{j\in\mathcal{J}} w_j \tilde{C}_j,$$

where (a) is by the second part of (18) for $l = l_j$, (b) is by definition of expectation with respect to λ, with pdf $f(\lambda) = 2\lambda$, and (c) is by Corollary 2. Using the previous inequality and Lemma 5,

$$\mathbb{E}\left[\sum_{j\in\mathcal{J}} w_j C_j\right] \le 24 \times \sum_{j\in\mathcal{J}} w_j C^\star_j = 24 \times \mathrm{OPT}. \tag{19}$$

By applying the derandomization procedure (see Online Appendix C), we can find $\lambda = \lambda^\star$ in polynomial time for which the total weighted completion time is less that its expected value in (19). This completes the proof of Theorem 2. □

5. Special Case: Preemption and Single-Machine Placement Set

In previous sections, we studied the parallel-task job scheduling problem for both cases when migration of tasks (among machines in its placement set) is allowed or not and provided $(6+\epsilon)$- and 24-approximation algorithms, respectively. In this section, we consider a special case when only one machine is in the placement set of each task (e.g., it is the only machine that has the required data for processing the task), and preemption is allowed. Using the three-field notation, this case is represented by PDP|pmtn|$\sum_j w_j C_j$.

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

Corollary 3. *Consider the parallel-task job scheduling problem when there is a specific machine to process each task and preemption is allowed. For any $\epsilon > 0$, the sum of the weighted completion times of jobs under* SynchPack-1, *in Section 3.2, is at most $(4+\epsilon) \times OPT$.*

Proof. The proof is straight forward and similar to proof of Theorem 1. Specifically, the factor of three needed to bound the solution of the greedy policy is reduced to two because placement constraint is not needed to be enforced here, because there is only one machine for each task. □

We can show that there is a slightly better approximation algorithm to solve the problem in this special case that has an approximation ratio of four. The algorithm uses a relaxed LP, based on linear ordering variables (Gandhi et al. 2008, Mastrolilli et al. 2010, Shafiee and Ghaderi 2018) to find an efficient ordering of jobs. Then it applies a simple list scheduling to pack their tasks in machines subject to capacity constraints. The details are as follows.

5.1. Relaxed Linear Program (LP3)

Each task must be processed in a specific machine. Each job consists of up to M (number of machines) different tasks. We use $\mathcal{M}_j$ to denote the set of machines that have tasks for job j. Task i of job j, denoted as task (i, j), requires a specific amount a_{ij} of machine i's resource ($a_{ij} \le m_i$) for a specific time duration $p_{ij} > 0$. We also define its volume as $v_{ij} = a_{ij}p_{ij}$. The results also hold in the case that a job has multiple tasks on the same machine.

For each pair of jobs, we define $\delta_{jj'} \in \{0,1\}$ such that $\delta_{jj'} = 1$ if job j is completed before job j', and $\delta_{jj'} = 0$ otherwise. Note that by the synchronization constraint (1), the completion of a job is determined by its last task. If both jobs finish at the same time, we set either one of $\delta_{jj'}$ or $\delta_{j'j}$ to one and the other one to zero, arbitrarily. By relaxing the integral constraint on binary variables, we formulate the following LP:

$$\min \sum_{j\in\mathcal{J}} w_j C_j \quad (\textbf{LP3}), \tag{20a}$$

$$m_i C_j \ge v_{ij} + \sum_{j'\in\mathcal{J}, j'\ne j} v_{ij'}\delta_{j'j},\ j\in\mathcal{J}, i\in\mathcal{M}_j, \tag{20b}$$

$$C_j \ge p_{ij},\ j\in\mathcal{J}, i\in\mathcal{M}_j, \tag{20c}$$

$$\delta_{jj'} + \delta_{j'j} = 1,\ j\ne j', j,j'\in\mathcal{J}, \tag{20d}$$

$$\delta_{jj'} \ge 0,\ j,j'\in\mathcal{J}. \tag{20e}$$

Recall the definition of job completion time C_j and task completion time C_{ij} in Section 2. In (LP3), (20b) follows from the definition of $\delta_{jj'}$, and the fact that the tasks that need to be served on machine i are processed by a single machine of capacity m_i. It states that the total volume of tasks that can be processed during the time period $(0, C_j]$ by machine i is *at most* $m_i C_j$. This total volume is given by the right-hand side of (20b), which basically sums the volumes of the tasks on machine i that finish before job j finishes its corresponding tasks at time C_j, plus the volume of task (i, j) itself. Constraint (20c) is because $C_j \ge C_{ij}$ and each task cannot be completed before its processing time p_{ij}. (20d) indicates that for each two jobs, one precedes the other. Furthermore, we relax the binary ordering variables to be fractional in (20e).

The optimal solution to (LP3) might be an infeasible schedule because (LP3) replaces the tasks by sizes of their volumes, and it might be impossible to pack the tasks in a way that matches the obtained completion times from (LP3).

Remark 1. (LP3) can be easily modified to allow each job to have multiple tasks on the same machine. We omit the details to focus on the main ideas.

5.2. Scheduling Algorithm: SynchPack – 3

The SynchPack-3 algorithm has two steps:

Step 1: Solve (LP3) to find an ordering of jobs. Let $\tilde{C}_j$ denote the optimal solution to (LP3) for completion time of job $j \in \mathcal{J}$. We order jobs based on their $\tilde{C}_j$ values in a nondecreasing order. Without loss of generality, we reindex the jobs such that

$$\tilde{C}_1 \le \tilde{C}_2 \le \ldots \le \tilde{C}_N. \tag{21}$$

Ties are broken arbitrarily.

Step 2: List scheduling based on the obtained ordering. For each machine i, the algorithm maintains a list of tasks such that for every two tasks (i, j) and (i, j') with $j < j'$ (according to ordering (21)), task (i, j) appears before task (i, j') in the list. On machine i, the algorithm scans the list starting from the first task. It schedules a task (i, j) from the list if the machine has sufficient remaining resource to accommodate it. Upon completion of a task, the algorithm preempts the schedule, removes the completed task from the list and updates the remaining processing time of the tasks in the list, and starts scheduling the tasks in the updated list. Observe that this list scheduling is slightly different from the greedy scheme used in SynchPack-1. A pseudocode for the algorithm can be found in Online Appendix I.

5.3. Performance Guarantee

Theorem 3. *The scheduling algorithm* SynchPack-3 *is a four-approximation algorithm for the problem of parallel-task jobs scheduling with packing and single-machine placement constraints.*

The proof of the theorem, and any supporting lemmas, is presented in Online Appendix E.

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

Table 1. Performance Ratio of SynchPack-3 with Respect to (LP3) and SynchPack-2 with Respect to (LP2)

Jobs' weights	Equal	Random	Priority based
Ratio for SynchPack-2	2.87	2.90	2.98
Ratio for SynchPack-3	1.34	1.35	1.31

6. Complexity of Algorithms

The complexity of our algorithms is mainly dominated by solving their corresponding LPs, which can be solved in polynomial time using efficient linear programming solvers. The rest of the operations have low complexity and can be parallelized on the machines. We provided a detailed discussion of the complexity in Online Appendix A.

7. Evaluation Results

In this section, we evaluate the performance of our algorithms using a real traffic trace from a large Google cluster (Wilkes 2011) and compare with prior algorithms. The original data set only contains the machine to which each task is assigned by the resource manager, and the information regarding the placement constraints (data locality) is missing. The setting is then similar to our model for preemptive algorithm SynchPack-3 in Section 5. To incorporate placement constraints, we modify the data set as follows. For each task, we randomly choose three machines and assume that processing time of the task on these machines is equal to the processing time given in the data set. We allow the task to be scheduled on other machines; however, its processing time will be penalized by a factor $\alpha > 1$. This is consistent with the data locality models in previous work (Grandl et al. 2015, Wang et al. 2016). The details of the data set can be found in Online Appendix F.

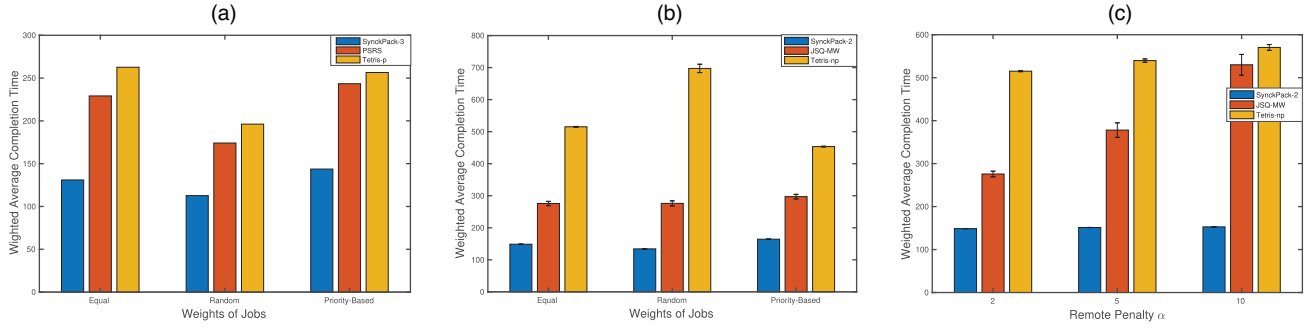
We consider three prior algorithms, PSRS (Schwiegelshohn 2004), Tetris (Grandl et al. 2015), and JSQ-MW (Wang et al. 2016), to compare with our algorithms SynchPack-2 and SynchPack-3. PSRS is a preemptive algorithm for the parallel task scheduling problem (see Section 1.1) on a single machine. Tetris is a heuristic that schedules tasks on each machine according to an ordering based on their scores (Section 1.1). In our evaluations, we consider two versions of Tetris, preemptive (Tetris-p) and nonpreemptive (Tetris-np). Finally, JSQ-MW is a nonpreemptive algorithm in presence of data locality (Section 1.1). An overview of these algorithms can be found in Online Appendix F.

7.1. Results in Offline Setting

We use SynchPack-3, Tetris-p, and PSRS to schedule tasks of the original data set preemptively and use SynchPack-2, Tetris-np, and JSQ-MW to schedule tasks of the modified data set (with placement constraints) nonpreemptively. We then compare the weighted average completion time of jobs, $\sum_j w_j C_j / \sum_j w_j$, under these algorithms for the three weight cases, that is, equal, random, and priority-based weights. Weighted average completion time is equivalent to the total weighted completion time (up to the normalization $\sum_j w_j$). We first report the ratio between the total weighted completion time obtained from SynchPack-2 (for $\alpha = 2$) and SynchPack-3 and their corresponding optimal value of their relaxed LPs (13) and (20) (which are lower bounds on the optimal total weighted competition times) to verify Theorems 2 and 3. Table 1 shows this performance ratio for the three cases of job weights. All ratios are within our theoretical results of 24 and 4. In fact, the approximation ratios are much smaller.

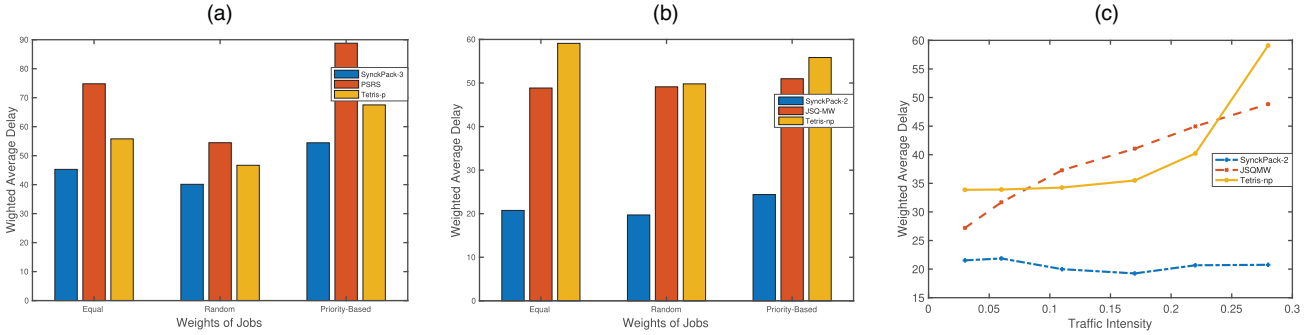
Figure 4(a) shows the performance of SynchPack-3, Tetris-p, and PSRS in the offline setting. As we see, SynchPack-3 outperforms the other two algorithms in all the cases and performance gain varies from 33% to 132%. Furthermore, Figure 4(b) depicts performance of SynchPack-2, Tetris-np, and JSQ-MW for different

Figure 4. (Color online) Performance of Algorithms in the Offline Setting



Notes. (a) Performance of SynchPack-3, Tetris – p, and **PSRS** for different weights. (b) Performance of SynchPack – 2, Tetris – np, and JSQ – MW for different weights and remote penalty $\alpha = 2$. (c) Performance of SynchPack – 2, Tetris – np, and JSQ – MW for different remote penalties and equal weights.

Figure 5. (Color online) Performance of Algorithms in the Online Setting



Notes. (a) Performance of SynchPack – 3, Tetris – p, and **PSRS** for different weights. (b) Performance of SynchPack – 2, Tetris – np, and JSQ – MW for different weights. (c) Performance of SynchPack – 2, Tetris – np, and JSQ – MW for different traffic intensities.

weights, when $\alpha = 2$. The performance gain of SynchPack-2 varies from 81% to 420%. Figure 4(c) shows the effect of remote penalty α in the performance of SynchPack-2, Tetris-np, and JSQ-MW. As we see, SynchPack-2 outperforms the other algorithms by 85%–273%.

7.2. Results in Online Setting

In the online setting, jobs arrive dynamically over time, according to the arrival time information in the data set, and we are interested in the weighted average *delay* of jobs. The delay of a job is measured from the time that it arrives to the system until its completion. See Online Appendix F for details on implementation of the algorithms in the online setting.

Figure 5(a) shows the performance results, in terms of the weighted average *delay* of jobs, under SynchPack-3, Tetris-p, and PSRS. Performances of Tetris-p is worse than our algorithm by 11%–27%, whereas PSRS presents the poorest performance and has 36%–65% larger weighted average delay compared with SynchPack-3. Moreover, performance of SynchPack-2, Tetris-np, and JSQ-MW for different weights is depicted in Figure 5(b). As we see, SynchPack-2 outperforms the other two algorithms in all the cases, and performance gain varies from 109% to 189%. Furthermore, by multiplying arrival times by constant values, we can change the traffic intensity and study its effect on algorithms' performance. Figure 5(c) shows the results for equal job weights. As we can see, SynchPack-2 outperforms the other algorithms and the performance gain increases as traffic intensity grows.

8. Conclusions

We studied the problem of scheduling jobs, each job with multiple resource constrained tasks, in a cluster of machines. We proposed the first constant-approximation algorithms for minimizing the total weighted completion time of such jobs. The model and analysis in our setting of tasks with packing, synchronization, and placement constraints are new. The approximation results are upper bounds on the algorithms' performance, and in fact, our simulation results showed that the approximation ratios are very close to one in practice.

As we showed, applying our simple greedy packing to schedule tasks mapped to each interval in SynchPack-2 provides a tight bound on the total volume of tasks and its relation to the associated linear program. Therefore, we cannot improve the final result by replacing this step with more intelligent bin packing algorithms like BestFit (Coffman et al. 1980). However, in practice, applying such bin packing schemes can give a better performance. Improving the performance bound of 24 requires a more careful and possibly different analysis. We leave further improvement of the result as a future work. Extension of our model to capture multidimensional task resource requirements and analysis of online algorithms for our problem are also interesting and challenging topics for future work.

Acknowledgments

The authors thank the editor-in-chief John Birge, area editor Samuel Burer, anonymous associate editor, and anonymous referees for comments and suggestions.

Endnotes

[1] This is because of how C_j is defined as a convex combination of the interval left points in Constraint (9g). More specifically, assume job j consists of one task and completes at interval l_j; however, only a very small fraction of its task is scheduled in l_j, that is, x_{jl_j} is very small. Furthermore, assume the rest of the task is scheduled at some interval l where $l << l_j$. Then, we can choose x_{jl_j} such that $C_j \sim d_l$ (according to (9g)), while the actual completion time of job j in schedule $\mathcal{S}$ can be $\sim d_{l_j}$.

[2] $G = (U \cup V, E)$ is a bipartite graph iff for any edge $e = (u, v) \in E$, we have $u \in U$ and $v \in V$.

[3] A perfect matching in G (with size $|U|$) is a subset of E such that every node in set U is matched to one and only one node in set V by an edge in the subset.

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

References

Ahmadi R, Bagchi U, Roemer TA (2005) Coordinated scheduling of customer orders for quick response. *Naval Res. Logist.* 52(6):493–512.

Ananthanarayanan G, Ghodsi A, Shenker S, Stoica I (2011) Disk-locality in datacenter computing considered irrelevant. *HotOS'13: Proc. 13th USENIX Conf. Hot Topics Operating Systems* (USENIX Association, Berkeley, CA), 1–12.

Ananthanarayanan G, Kandula S, Greenberg AG, Stoica I, Lu Y, Saha B, Harris E (2010) Reining in the outliers in map-reduce clusters using mantri. *OSDI '10: Proc. 9th USENIX Conf. Operating Systems Design and Implementation*, (USENIX Association, Berkeley, CA), 265–278.

Apache SF (2018a) Apache hadoop. Accessed January 5, 2022, http://hadoop.apache.org.

Apache SF (2018b) Apache hadoop yarn. Accessed January 5, 2022, http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

Apache SF (2018c) Apache spark. Accessed January 5, 2022, https://spark.apache.org/docs/latest/index.html.

Bansal N, Khot S (2010) Inapproximability of hypergraph vertex cover and applications to scheduling problems. *Proc. Internat. Colloquium on Automata, Languages, and Programming* (Springer, Berlin), 250–261.

Bansal N, Srinivasan A, Svensson O (2019) Lift-and-round to improve weighted completion time on unrelated machines. *SIAM J. Comput.* STOC16–STOC138.

Blazewicz J, Lenstra JK, Rinnooy Kan A (1983) Scheduling subject to resource constraints: Classification and complexity. *Discrete Appl. Math.* 5(1):11–24.

Cheatham T, Fahmy A, Stefanescu D, Valiant L (1996) Bulk synchronous parallel computing—A paradigm for transportable software. *Tools and Environments for Parallel and Distributed Systems* (Springer, Berlin), 61–76.

Chen ZL, Hall NG (2007) Supply chain scheduling: Conflict and cooperation in assembly systems. *Oper. Res.* 55(6):1072–1089.

Coffman EG, Bruno JL (1976) *Computer and Job-Shop Scheduling Theory* (John Wiley & Sons, Hoboken, NJ).

Coffman EG Jr, Garey MR, Johnson DS, Tarjan RE (1980) Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM J. Comput.* 9(4):808–826.

Dean J, Ghemawat S (2008) Mapreduce: Simplified data processing on large clusters. *Comm. ACM* 51(1):107–113.

Gandhi R, Halldórsson MM, Kortsarz G, Shachnai H (2008) Improved bounds for scheduling conflicting jobs with minsum criteria. *ACM Trans. Algorithms* 4(1):11.

Garey MR, Graham RL (1975) Bounds for multiprocessor scheduling with resource constraints. *SIAM J. Comput.* 4(2):187–200.

Garg N, Kumar A, Pandit V (2007) Order scheduling models: hardness and algorithms. *Proc. Internat. Conf. on Foundations of Software Tech. and Theoretical Comput. Sci.* (Springer, Berlin), 96–107.

Goldberg LA, Paterson M, Srinivasan A, Sweedyk E (2001) Better approximation guarantees for job-shop scheduling. *SIAM J. Discrete Math.* 14(1):67–92.

Grandl R, Ananthanarayanan G, Kandula S, Rao S, Akella A (2015) Multi-resource packing for cluster schedulers. *Comput. Comm. Rev.* 44(4):455–466.

Grandl R, Kandula S, Rao S, Akella A, Kulkarni J (2016) {GRAPHENE}: Packing and dependency-aware scheduling for data-parallel clusters. *Proc. 12th Sympos. Operating Systems Design Implementation* (USENIX Association, Berkeley, CA), 81–97.

Hall P (1935) On representatives of subsets. *J. London Math. Soc.* 1(1):26–30.

Im S, Li S (2016) Better unrelated machine scheduling for weighted completion time via random offsets from non-uniform distributions. *Proc. IEEE 57th Annual Sympos. Foundations of Comput. Sci.* (IEEE, New York), 138–147.

Jin J, Luo J, Song A, Dong F, Xiong R (2011) Bar: An efficient data locality driven task scheduling algorithm for cloud computing. *Proc. 11th IEEE/ACM Internat. Sympos. Cluster, Cloud and Grid Comput* (IEEE, New York), 295–304.

Kambatla K, Rapolu N, Jagannathan S, Grama A (2010) Asynchronous algorithms in mapreduce. *Proc. IEEE Internat. Conf. Cluster Comput.* (IEEE, New York), 245–254.

Lenstra JK, Shmoys DB, Tardos E (1990) Approximation algorithms for scheduling unrelated parallel machines. *Math. Programming* 46(1-3):259–271.

Leung JYT, Li H, Pinedo M (2007) Scheduling orders for multiple product types to minimize total weighted completion time. *Discrete Appl. Math.* 155(8):945–970.

Li S (2020) Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations. *SIAM J. Comput.* 49(4):FOCS17–FOCS409.

Liu J, Shen H (2016) Dependency-aware and resource-efficient scheduling for heterogeneous jobs in clouds. *Proc. IEEE Internat. Conf. on Cloud Comput. Tech. and Sci.*, 110–117.

Mastrolilli M, Queyranne M, Schulz AS, Svensson O, Uhan NA (2010) Minimizing the sum of weighted completion times in a concurrent open shop. *Oper. Res. Lett.* 38(5):390–395.

Munier A, Queyranne M, Schulz AS (1998) Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. *Proc. Internat. Conf. on Integer Programming and Combinatorial Optimization* (Springer, Berlin), 367–382.

Nitu V, Kocharyan A, Yaya H, Tchana A, Hagimont D, Astsatryan H (2018) Working set size estimation techniques in virtualized environments: One size does not fit all. *Proc. ACM on Measurement and Analysis of Comput. Systems* 2(1):1–22.

Qiu Z, Stein C, Zhong Y (2015) Minimizing the total weighted completion time of coflows in datacenter networks. *Proc. 27th ACM Sympos. on Parallelism in Algorithms and Architectures* (ACM, New York), 294–303.

Queyranne M, Schulz AS (2006) Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. *SIAM J. Comput.* 35(5):1241–1253.

Queyranne M, Sviridenko M (2002) A (2+ ε)-approximation algorithm for the generalized preemptive open shop problem with minsum objective. *J. Algorithms* 45(2):202–212.

Rasley J, Karanasos K, Kandula S, Fonseca R, Vojnovic M, Rao S (2016) Efficient queue management for cluster scheduling. *Proc. 11th Eur. Conf. Computer Systems* (ACM, New York), 1–15.

Remy J (2004) Resource constrained scheduling on multiple machines. *Inform. Processing Lett.* 91(4):177–182.

Sachdeva S, Saket R (2013) Optimal inapproximability for scheduling problems via structural hardness for hypergraph vertex cover. *Proc. IEEE Conf. on Computational Complexity* (IEEE, New York), 219–229.

Schulz AS, Skutella M (1997) Random-based scheduling new approximations and lp lower bounds. *Proc. Internat. Workshop on Randomization and Approximation Techniques in Comput. Sci.* (Springer, Berlin), 119–133.

Schulz AS, Skutella M (2002) Scheduling unrelated machines by randomized rounding. *SIAM J. Discrete Math.* 15(4):450–469.

Schwarzkopf M, Konwinski A, Abd-El-Malek M, Wilkes J (2013) Omega: Flexible, scalable schedulers for large compute clusters. *Proc. 8th ACM Eur. Conf. on Comput Systems* (ACM, New York), 351–364.

Schwiegelshohn U (2004) Preemptive weighted completion time scheduling of parallel jobs. *SIAM J. Comput.* 33(6):1280–1308.

Shafiee M, Ghaderi J (2018) An improved bound for minimizing the total weighted completion time of coflows in datacenters. *IEEE/ACM Trans. Networks* 26(4):1674–1687.

Skutella M (2001) Convex quadratic and semidefinite programming relaxations in scheduling. *J. ACM* 48(2):206–242.

Verma A, Pedrosa L, Korupolu M, Oppenheimer D, Tune E, Wilkes J (2015) Large-scale cluster management at google with borg. *Proc. 10th Eur. Conf. on Comput. Systems* (ACM, New York), 1–17.

Published in Operations Research on January 18, 2022 as DOI: 10.1287/opre.2021.2198.
This article has not been copyedited or formatted. The final version may differ from this version.

Wang W, Zhu K, Ying L, Tan J, Zhang L (2016) Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality. *IEEE/ACM Trans. Networks* 24(1):190–203 (TON).

Wilkes J (2011) More Google cluster data. Accessed January 5, 2022, http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.

Yekkehkhany A, Hojjati A, Hajiesmaili MH (2018) GB-pandas: Throughput and heavy-traffic optimality analysis for affinity scheduling. *Performance Evaluation Rev.* 45(3):2–14.

Zaharia M, Konwinski A, Joseph AD, Katz RH, Stoica I (2008) Improving mapreduce performance in heterogeneous environments. *Open Systems & Information Dynamics*, vol. 8, 7.

Zaharia M, Borthakur D, Sen Sarma J, Elmeleegy K, Shenker S, Stoica I (2010) Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. *Proc. 5th Eur. Conf. on Comput. Systems* (ACM, New York), 265–278.

Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, et al (2016) Apache spark: A unified engine for big data processing. *Comm. ACM* 59(11):56–65.

Mehrnoosh Shafiee received her PhD in 2020 from the Department of Electrical Engineering at Columbia University. Her PhD research was broadly in optimization and network algorithms. Her thesis title is "Resource Allocation in Large-Scale Distributed Systems," in which she designed and analyzed resource allocation and scheduling algorithms for data centers. She is currently a quantitative analyst at Citigroup.

Javad Ghaderi is an associate professor of electrical engineering at Columbia University. His research interests include network algorithms, control, and optimization. He is the recipient of several awards including the Mac Van Valkenburg Graduate Research Award at University of Illinois at Urbana–Champaign, Best Paper Award at ACM CoNEXT 2016, National Science Foundation CAREER Award in 2017, Best Paper Award at IEEE INFOCOM 2020, and Best Student Paper Award at IFIP Performance 2020.