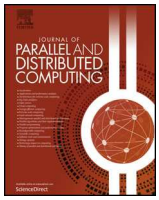
Journal of Parallel and Distributed Computing 184 (2024) 104792

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

journal homepage: www.elsevier.com/locate/jpdc



Multi-resource scheduling of moldable workflows ☆

Lucas Perotin [a], Sandhya Kandaswamy [b], Hongyang Sun [b,*], Padma Raghavan [c]

[a] Ecole Normale Superieure de Lyon, France
[b] University of Kansas, USA
[c] Vanderbilt University, USA

ARTICLE INFO

Keywords:
Makespan scheduling
Multi-resources scheduling
Moldable jobs
Workflows
Approximation ratio

ABSTRACT

Resource scheduling plays a vital role in High-Performance Computing (HPC) systems. Most scheduling research in HPC has focused on only a single type of resource (e.g., computing cores or I/O resource). With the advancement in hardware architectures and the increase in data-intensive HPC applications, there is a need to simultaneously consider a diverse set of resources (e.g., computing cores, cache, memory, I/O, and network resources) in the design of runtime schedulers for improving the overall application performance. In this paper, we study multi-resource scheduling to minimize the makespan of computational workflows comprised of moldable parallel jobs. Moldable jobs allow the scheduler to flexibly select a variable set of resources before execution, thus can adapt to the available system resources (as compared to rigid jobs) while staying easy to design and implement (as compared to malleable jobs). We propose a Multi-Resource Scheduling Algorithm (MRSA), which combines a novel resource allocation strategy and an extended list scheduling scheme to schedule the jobs. We prove that, on a system with d types of schedulable resources, MRSA achieves an approximation ratio of $1.619d + 2.545\sqrt{d} + 1$ for any $d \geq 1$, and a ratio of $d + 3\sqrt[3]{d^2} + O(\sqrt[3]{d})$ when d is large (i.e., $d \geq 22$). We also present improved approximation results for workflows comprised of jobs with special precedence constraints (e.g., series-parallel graphs, trees, and independent jobs). Further, we prove a lower bound of d on the approximation ratio of any list-based scheduling algorithm with local priority considerations. Finally, we conduct a comprehensive set of simulations to evaluate the performance of the algorithm using synthetic workflows of different structures and moldable jobs following different speedup models. The results show that MRSA fares better than the theoretical bound predicts, and that it consistently outperforms two baseline heuristics under a variety of parameter settings, illustrating its robust practical performance.

1. Introduction

Complex scientific workflows running in today's High-Performance Computing (HPC) systems are typically modeled as Directed Acyclic Graphs (DAGs), whose nodes represent the jobs of the workflows and whose edges represent the precedence constraints (or dependencies) among the jobs. Effective workflow scheduling plays a vital role in improving the performance of scientific applications. While HPC systems often rely on dynamic runtime schedulers, such as KAAPI [24], StarPU [2] or PaRSEC [7], to ensure the efficient execution of computational workflows, most existing schedulers focus only on a single type of resource (e.g., computing resource or I/O resource). With the advancement in hardware architectures and the increase in data-intensive HPC applications, there is a need to simultaneously consider multiple types of resources (e.g., computing, cache, memory, I/O, and network resources) in the design of runtime schedulers. Indeed, modern HPC systems are equipped with more levels of memory/storage (e.g., NVRAMs, SSDs, burst buffers [44]), all of which could potentially be partitioned among the system's concurrently running jobs. Advancements in architectural and software features (e.g., high-bandwidth memory [60], cache partitioning [69], bandwidth reservation [8]) can also be leveraged to facilitate the efficient scheduling of these multiple types of resources for enhancing the overall application and/or system performance.

In this paper, we study multi-resource scheduling for computational workflows comprised of moldable parallel jobs with DAG-based prece-

☆ A preliminary version [50] of this paper has appeared in the Proceedings of the 50th International Conference on Parallel Processing (ICPP), 2021.

* Corresponding author.
E-mail address: hongyang.sun@ku.edu (H. Sun).

https://doi.org/10.1016/j.jpdc.2023.104792
Received 4 January 2023; Received in revised form 3 July 2023; Accepted 16 October 2023
Available online 24 October 2023

dence constraints. The goal is to simultaneously explore the availability of multiple types of resources by designing effective scheduling solutions that minimize the overall completion time, or *makespan*, of a workflow. We focus on parallel jobs that are *moldable* [21], which allows the scheduler to select a variable set of resources for a job, but once the job has started its execution, the resource allocations cannot be changed. In contrast to *rigid* jobs, whose resource allocations are all static and hence fixed, moldable jobs can easily adapt to the different amounts of available resources, while in contrast to *malleable* jobs, whose resource allocations can be dynamically varied during runtime, moldable jobs are much easier to design and implement.

As the considered multi-resource scheduling problem contains the single-resource scheduling problem as a special case, it is known to be strongly NP-complete [16]. Thus, we focus on designing good approximation algorithms. In contrast to the single-resource problem, the multi-resource problem needs to consider the combined effect of multiple types of resources on the execution of the jobs, thus posing additional challenges to the scheduling problem. By adopting a two-phase approach [63] that is widely used for scheduling moldable jobs, we design a Multi-Resource Scheduling Algorithm (MRSA), which first computes an approximate resource allocation for all jobs on different resource types and then applies an extended list scheduling scheme to schedule the jobs. As list scheduling is easy to implement, the proposed algorithm can be readily applied to practical systems.

On the theoretical side, we prove, under reasonable assumptions on the job execution times and speedup functions, the following results for a system consisting of d types of schedulable resources:

- MRSA achieves an approximation ratio of $1.619d + 2.545\sqrt{d} + 1$ for any $d \geq 1$, and a ratio of $d + 3\sqrt[3]{d^2} + O(\sqrt[3]{d})$ for large d (i.e., $d \geq 22$);
- MRSA has improved approximations for some special graphs (e.g., series-parallel graphs, trees and independent jobs) with ratios of $1.619d + 1$ for any $d \geq 1$ and $d + O(\sqrt{d})$ for any $d \geq 4$;
- We prove a lower bound of d on the approximation ratio of any list-based scheduling algorithm with local priority considerations.

To the best of our knowledge, these are the first approximation results for scheduling moldable workflows with multiple resource requirements. They also improve upon the $2d$-approximation previously shown in [61] for scheduling independent moldable jobs. The results demonstrate that MRSA achieves the optimal asymptotic approximation up to the dominating factor (i.e., d) among the generic class of local list-based scheduling schemes, thus matching the same asymptotic performance for scheduling rigid [23] and malleable [29] jobs.

On the practical side, we conduct a comprehensive set of simulations using synthetic workflows generated by DAGGEN [62], which is a task graph generator capable of generating DAGs of different structures. We also generate moldable jobs that exhibit different runtime characteristics on multiple resource types by extending common speedup profiles that follow Amdahl's law [1] and power law [51] models. Our simulation results show that:

- MRSA fares better than the worst-case theoretical bound predicts;
- MRSA consistently outperforms two baseline multi-resource scheduling heuristics;
- MRSA performs comparably with some other approximation algorithms under the special case of a single resource type.

These simulation results nicely complement the theoretical analysis of MRSA, and they also illustrate MRSA's robust practical performance under a variety of parameter settings.

The rest of this paper is organized as follows. Section 2 reviews some related work on moldable and multi-resource scheduling. Section 3 formally introduces the scheduling model and shows a lower bound on the optimal makespan. Section 4 presents the multi-resource scheduling algorithm MRSA and analyzes its approximation ratios for general workflows. Section 5 proves improved approximation results for some special workflows, including series-parallel graphs, trees and independent jobs. Section 6 shows a lower bound on the performance of any local list-based scheduling algorithm. Section 7 presents a comprehensive set of simulation results for evaluating the performance of the proposed algorithm, and finally, Section 8 concludes the paper and briefly discusses future work.

2. Related work

This section reviews some related work on single-resource moldable job scheduling as well as multi-resource job scheduling to minimize the makespan. We will also review some multi-resource scheduling work under alternative job models and objectives.

2.1. Single-resource moldable job scheduling

Scheduling moldable jobs to minimize the makespan is a strong NP-complete problem on $P \geq 5$ processors [16], and it has been extensively studied in the literature. Most prior work, however, has only focused on a single type of resource while assuming different speedup models for the jobs. The following reviews some related work from the perspectives of both approximation algorithms and heuristic algorithms.

2.1.1. Approximation algorithms

For independent moldable jobs with arbitrary speedups, Turek et al. [63] presented a 2-approximation list-based algorithm and a 3-approximation algorithm based on building shelves. Ludwig and Tiwari [45] later improved the 2-approximation result with lower computational complexity. For monotonic jobs, whose execution time $t(p)$ is non-decreasing in the number p of allocated processors and whose work function $w(p) = p \cdot t(p)$ is non-decreasing in p, Mounié et al. [47] presented a $(1.5 + \epsilon)$-approximation algorithm using dual approximation. Jansen and Land [34] showed a faster algorithm that achieves the same $(1.5+\epsilon)$-approximation as well as a PTAS when the execution time functions of the jobs admit compact encodings.

For scheduling moldable jobs that have precedence constraints, Lepère et al. [42] presented a 5.236-approximation algorithm when jobs are assumed to be monotonic. Jansen and Zhang [36] improved the approximation ratio to around 4.73 for the same model, and recently, Chen [9] further improved it to around 3.42 using an iterative approximation method. Additionally, better approximation results have been obtained for jobs with special dependency graphs (e.g., series-parallel graphs and trees [42,41]) or special speedup models (e.g., concave speedup [35,10] and roofline speedup [65,22]).

2.1.2. Heuristic algorithms

A series of related work has also proposed heuristic algorithms for scheduling moldable jobs with precedence constraints under the distributed computing model where additional communication cost is incurred between two dependent jobs. Ramaswamy et al. [55] presented a Two Step Allocation and Scheduling (TSAS) algorithm that uses convex programming to allocate resources and list scheduling to schedule the jobs. Radulescu and van Gemund [54] took the same approach but proposed a low-cost heuristic, called Critical-Path and Allocation (CPA), that incrementally allocates the resources to the jobs. Bansal et al. [3] refined the CPA algorithm and proposed Modified CPA (MCPA) by limiting the total resource allocation for those jobs on the critical path but in the same layer of the graph. Hunold [32] further refined the MCPA algorithm by proposing MCPA2 that has been shown to work better for irregular workflows.

While the algorithms above use the two-level approach that separates the resource allocation and job scheduling decisions, Radulescu et al. [53] presented a one-step algorithm, called Critical Path Reduction (CPR), that iteratively improves the result of the list schedule by incrementally adding resource allocations to certain jobs. The algorithm

is shown to produce better schedules than some two-level algorithms at the expense of higher complexity. Huang et al. [31] also proposed a one-step algorithm, called Iterative Allocation Expanding and Shrinking (IAES), that allows the allocated resources of a job to shrink (as well as expand) based on the actual critical path in the schedule rather than the static one in the graph.

Most of these algorithms are heuristic algorithms without performance guarantees, except for TSAS [55], which was shown to have an approximation ratio of 11.66.

2.2. Multi-resource job scheduling

The literature also contains some approximation algorithms on multi-resource scheduling to minimize makespan under different parallel job models, as well as works on multi-resource scheduling for alternative job models and objective functions.

2.2.1. Approximation algorithms to minimize makespan

Garey and Graham [23] considered scheduling n sequential jobs on m identical machines with d additional types of resources. Further, each job has a fixed resource requirement from each resource type, making it essentially a *rigid* job scheduling model. They presented a list-scheduling algorithm and proved three results: (1) an m-approximation for jobs with precedence constraints and when there is only one type of resource, i.e., $d = 1$; (2) a $(d + 1)$-approximation for independent jobs and when the number of machines is not a constraining factor, i.e., $m \geq n$; (3) a $(d + 2 - \frac{2d+1}{m})$-approximation for independent jobs with any $m \geq 2$. For the case of $d = 1$, Demirci et al. [14] presented an improved $O(\log n)$-approximation for jobs with precedence constraints, and Niemeier and Wiese [48] presented an improved $(2+\epsilon)$-approximation for independent jobs.

He et al. [29,28] considered parallel jobs that are represented as DAGs consisting of unit-size tasks, each of which requests a single type of resource from a total of d resource types. Further, the amount of resources allocated to a job can be dynamically changed during runtime, making it essentially a *malleable* job scheduling model. They showed that list scheduling achieves $(d + 1)$-approximation under this model. Shmoys et al. [58] considered a similar model while further restricting the tasks of each job to be processed sequentially. They called it the *DAG-shop* scheduling model, and presented a polylog approximation result in number of machines and job length.

Sun et al. [61] considered independent *moldable* jobs on d types of resources. They presented a $2d$-approximation list-based algorithm and a $(2d + 1)$-approximation shelf-based algorithm, thus generalizing the single-resource results in [63]. They also presented a technique to transform any c-approximation algorithm for a single resource type to a cd-approximation algorithm for d types of resources. This work is the closest to ours, while we consider moldable jobs with precedence constraints. When jobs are independent, our main approximation result also improves the one in [61] for a large number of resource types.

2.2.2. Multi-resource scheduling for alternative job models and objectives

Beaumont et al. [5] and Eyraud-Dubois and Kumar [17] considered scheduling sequential jobs on two alternative types of resources (CPU and GPU) to minimize the makespan. In their model, each job can be chosen to execute on either resource type with different processing rates. They analyzed an approximation algorithm, called HeteroPrio, for both independent jobs and jobs with precedence constraints. The approximation ratios depend on the relative amount of resources in the two resource types. A recent survey on this two-resource scheduling model can also be found in [4].

Ghodsi et al. [25] focused on the objective of resource allocation fairness in a multi-user and multi-resource setting. In their seminal paper, they proposed a Dominant Resource Fairness (DRF) algorithm that aims at maximizing the minimum dominant share across all users. Their work has subsequently been extended by many papers (e.g., [40,67,39,66,27,37,38]) that proposed improved or alternative algorithms by building on top of DRF for various multi-resource scheduling contexts.

Grandl et al. [26] considered scheduling malleable jobs under four specific resource types (CPU, memory, disk and network). They designed a heuristic algorithm, called Tetris, that schedules jobs by considering the correlation between the job's peak resource demands and the machine's resource availabilities, with the goal of minimizing resource fragmentation. NoroozOliaee et al. [49] studied a similar problem but for two resources (CPU and memory). They showed that a simple scheduling heuristic that uses Best Fit and Shortest Job First delivers good performance in terms of resource utilization and job queueing delays. Recently, Xu et al. [68] proposed DollyMP, a scheduling algorithm that considers two resource types (CPU and memory) and aims to minimize the total flow time of a set of dynamically arriving DAG-based jobs with stochastic execution times.

Sheikhalishahi et al. [57] used multi-dimensional bin-packing to schedule multiple resources in HPC and cloud environments. Their multi-resource scheduling algorithm is shown to outperform standard backfilling policies in terms of job slowdowns and wait times. Zhou et al. [71] applied a similar approach and presented P-Aware, a multi-resource scheduling strategy to map heterogeneous application jobs to server nodes in cloud data centers. Psychas and Ghaderi [52] showed that classical bin packing algorithms are not optimal in terms of throughput. They proposed a Randomized Multi-resource Scheduling (RMS) algorithm with provably optimal throughput.

Fan et al. [19] presented a scheduling algorithm, called BBSched, that targets two resources (CPU and burst buffer) for HPC. They proposed to use genetic algorithm to solve a formulated multi-objective optimization problem, which delivers fast scheduling decisions with improved resource utilization and average job wait time compared to existing HPC schedulers. They also extended their approach to dealing with multiple resource types, including memory and network resources [18], and provided a scheduling framework for the problem [20].

Recently, multi-resource scheduling has also been studied in the context of machine learning (ML) and deep learning (DL). On the one hand, some papers (e.g., [64,46,70]) have considered multiple resources when scheduling ML/DL jobs on computing clusters and reported improved job completion times. On the other hand, several efforts (e.g., [11,30,43]) have applied machine learning (in particular deep reinforcement learning) to the training of multi-resource schedulers, and they have led to better system and application performance compared to traditional schedulers.

3. Models

In this section, we formally present the multi-resource scheduling model and objective. We also derive a lower bound on the optimal makespan.

3.1. Scheduling model

We consider the problem of scheduling a set of n moldable jobs on d distinct types of resources (e.g., processor, memory, cache). Each resource type i has a total amount $P^{(i)}$ of available resource. The jobs are *moldable*, i.e., they can be executed using different amounts of resources from each resource type, but the resource allocation cannot be changed once a job has started executing. For each job j, where $1 \leq j \leq n$, its execution time $t_j(p_j)$ depends on the *resource allocation* $p_j = (p_j^{(1)}, p_j^{(2)}, \cdots, p_j^{(d)})$, which specifies the amount of resource $p_j^{(i)}$ allocated to the job for each resource type i, where $1 \leq i \leq d$. We make the following reasonable assumptions on the resource allocation and execution time of the jobs.

Assumption 1 *(Integral resources).* All resource allocations $p_j^{(i)}$'s for the jobs and the total amount of resources $P^{(i)}$'s for all resource types are non-negative integers.

This is a natural assumption for discrete resources, such as processors. Other resource types, such as memory or cache, are typically allocated in discrete chunks as well (e.g., memory blocks, cache lines) in practical systems.

Assumption 2 *(Known execution times)*. For each job j, its execution time function $t_j(p_j)$ is known for every possible resource allocation p_j.

In practice, the execution times of a job or an application could be obtained through one or more of the following approaches: application modeling or profiling, performance prediction or interpolation from historic data. Here, we assume these execution times are known while not concerning about how they are practically obtained.

Assumption 3 *(Monotonic jobs)*. Given two resource allocations p_j and q_j for a job j, we say that p_j is *at most* q_j, denoted by $p_j \preceq q_j$, if $p_j^{(i)} \leq q_j^{(i)}$ for all $1 \leq i \leq d$. The execution times of the job under these two allocations satisfy:

$$t_j(q_j) \leq t_j(p_j) \leq \left(\max_{i=1\ldots d} q_j^{(i)} / p_j^{(i)} \right) \cdot t_j(q_j) \ .$$

This generalizes the monotonic job assumption under a single resource type [42,47], which has been observed for many real-world applications. In particular, the first inequality specifies that the execution time of a job is *non-increasing* in the amount of resource allocated to the job,[1] and the second inequality restricts the job to have *non-superlinear* speedup with respect to any resource type.[2] Note that we do not make any assumptions on a job j's relative execution times under two resource allocations p_j and q_j that are *non-comparable*, i.e., $p_j \not\preceq q_j$ and $q_j \not\preceq p_j$.

Additionally, a set of *precedence constraints* is specified for the jobs, which form a directed acyclic graph (DAG), $G = (V, E)$. Each node $j \in V$ in the graph represents a job and a directed edge $(j_1 \to j_2) \in E$ requires that job j_2 cannot start executing until the completion of job j_1. In this case, j_1 is called an immediate *predecessor* of j_2, and j_2 is called an immediate *successor* of j_1.

3.2. Objective function

The objective is to find a schedule to minimize the maximum completion time, or the *makespan*. Specifically, a schedule is defined by the following two decisions:

- *Resource allocation decision*: $\mathbf{p} = (p_1, p_2, \ldots, p_n)$;
- *Starting time decision*: $\mathbf{s} = (s_1, s_2, \ldots, s_n)$.

Given a pair of scheduling decisions $\mathbf{p}$ and $\mathbf{s}$, the completion time of a job j is defined as $c_j = s_j + t_j(p_j)$, and the makespan of the jobs is given by $T = \max_j c_j$. A schedule must be *valid* by respecting the following constraints:

- For each resource type i, the amount of resource utilized by all running jobs at any time does not exceed the total amount $P^{(i)}$ of available resource;
- If two jobs j_1 and j_2 have a precedence constraint, i.e., $j_1 \to j_2$, then the starting time of j_2 is no earlier than the completion time of j_1, i.e., $s_{j_2} \geq c_{j_1}$.

[1] This assumption, however, is not restrictive, as we can discard any allocation that uses more resource than another allocation but results in a higher job execution time.

[2] Some parallel applications can achieve superlinear speedups with a combined effect of increased allocations in two or more resource types (e.g., the *cache effect* [56] when increasing both processor and cache allocations). We do not consider such superlinear speedup model in this paper.

The above multi-resource scheduling problem is NP-complete, as it contains the single-resource scheduling problem [36,42] as a special case. Thus, we aim at designing approximation algorithms with bounded performance guarantees. An algorithm is an *r-approximation* if its makespan for any set of jobs satisfies $\frac{T}{T_{\text{OPT}}} \leq r$, where T_{OPT} denotes the optimal makespan for the same set of jobs.

3.3. Lower bound on optimal makespan

We now derive a lower bound on the optimal makespan. To that end, we define the following concepts.

Definition 1. For each job j with resource allocation p_j:

- $w_j^{(i)}(p_j) = p_j^{(i)} \cdot t_j(p_j)$*:* work on resource type i;
- $a_j^{(i)}(p_j) = \frac{w_j^{(i)}(p_j)}{P^{(i)}}$*:* area (or normalized work) on resource type i;
- $a_j(p_j) = \frac{1}{d} \sum_{i=1}^{d} a_j^{(i)}(p_j)$*:* average area over all resource types.

Definition 2. For a set of jobs with resource allocation $\mathbf{p} = (p_1, p_1, \ldots, p_n)$:

- $W^{(i)}(\mathbf{p}) = \sum_{j=1}^{n} w_j^{(i)}(p_j)$*:* total work on resource type i;
- $A^{(i)}(\mathbf{p}) = \frac{W^{(i)}(\mathbf{p})}{P^{(i)}} = \sum_{j=1}^{n} a_j^{(i)}(p_j)$*:* total area on resource type i;
- $A(\mathbf{p}) = \frac{1}{d} \sum_{i=1}^{d} A^{(i)}(\mathbf{p}) = \sum_{j=1}^{n} a_j(p_j)$*:* average total area over all resource types;
- $C(\mathbf{p}, f) = \sum_{j \in f} t_j(p_j)$*:* total execution time of all the jobs along a particular path f in the graph[3];
- $C(\mathbf{p}) = \max_f C(\mathbf{p}, f)$*:* critical-path length, i.e., total execution time of the jobs along a critical (longest) path in the graph;
- $L(\mathbf{p}) = \max(A(\mathbf{p}), C(\mathbf{p}))$*:* maximum of average total area $A(\mathbf{p})$ and critical-path length $C(\mathbf{p})$.

We further define $L_{\min} = \min_{\mathbf{p}} L(\mathbf{p})$ to be the minimum value of $L(\mathbf{p})$ among all possible resource allocations, and let $\mathbf{p}^*$ denote a resource allocation such that $L(\mathbf{p}^*) = L_{\min}$. The following lemma shows that $L_{\min}$ serves as a lower bound on the optimal makespan.

Lemma 1. $T_{\text{OPT}} \geq L_{\min}$.

Proof. We first show that, given any resource allocation $\mathbf{p}$, the makespan produced by any schedule must satisfy $T \geq L(\mathbf{p}) = \max(A(\mathbf{p}), C(\mathbf{p}))$. The bound $T \geq C(\mathbf{p})$ is trivial, since the jobs along the critical path must be executed sequentially, so the makespan is at least $C(\mathbf{p})$. To derive the bound $T \geq A(\mathbf{p})$, we observe that the average total area $A(\mathbf{p})$ in any valid schedule with makespan T must satisfy:

$$\begin{aligned} A(\mathbf{p}) &= \frac{1}{d} \sum_{i=1}^{d} \sum_{j=1}^{n} \frac{w_j^{(i)}(p_j)}{P^{(i)}} \\ &= \frac{1}{d} \sum_{i=1}^{d} \frac{1}{P^{(i)}} \sum_{j=1}^{n} w_j^{(i)}(p_j) \\ &\leq \frac{1}{d} \sum_{i=1}^{d} \frac{1}{P^{(i)}} \cdot (P^{(i)} \cdot T) = T \ . \end{aligned}$$

The inequality $\sum_{j=1}^{n} w_j^{(i)}(p_j) \leq P^{(i)} \cdot T$ is because $P^{(i)} \cdot T$ is the maximum amount of work that can be allocated to the jobs within time T on any resource type i with total amount of resource $P^{(i)}$.

Suppose the optimal schedule uses a resource allocation $\mathbf{p}_{\text{OPT}}$. Then, its makespan must satisfy:

[3] A path is a sequence of jobs with linear precedence, i.e., $f = (j_{\pi(1)} \to j_{\pi(2)} \to \cdots \to j_{\pi(v)})$, where the first job $j_{\pi(1)}$ does not have any predecessor in the graph and the last job $j_{\pi(v)}$ does not have any successor.

$$T_{\text{OPT}} \geq \max\left(A(\mathbf{p}_{\text{OPT}}), C(\mathbf{p}_{\text{OPT}})\right)$$
$$= L(\mathbf{p}_{\text{OPT}})$$
$$\geq L(\mathbf{p}^*) = L_{\min} .$$

The last inequality is because $L(\mathbf{p}^*)$ is the minimum $L(\mathbf{p})$ among all possible resource allocations including $\mathbf{p}_{\text{OPT}}$. □

4. Multi-Resource Scheduling Algorithm (MRSA) and approximation results for general DAGs

In this section, we present the Multi-Resource Scheduling Algorithm (MRSA) and analyze its approximation ratio for general DAGs. The algorithm adopts the *two-phase approach* that has been widely used for scheduling moldable jobs on a single type of resource [63,42,36,55,54].

4.1. Phase 1: resource allocation

The first phase concerns finding a resource allocation decision $\mathbf{p} = (p_1, p_2, \dots, p_n)$ for all the jobs.

Discrete time-cost tradeoff (DTCT) problem. We first consider a relevant discrete time-cost tradeoff problem [13], which has been studied in the literature of operations research and project management.

Definition 3 *(Discrete time-cost tradeoff (DTCT)).* Suppose a project consists of n tasks with precedence constraints. Each task j can be executed using several different alternatives and each alternative i takes time $t_{j,i}$ and has cost $c_{j,i}$. Further, for any two alternatives i_1 and i_2, if i_1 is faster than i_2, then i_1 is more costly than i_2, i.e.,

$$t_{j,i_1} \leq t_{j,i_2} \Rightarrow c_{j,i_1} \geq c_{j,i_2} . \tag{1}$$

Given a project realization σ that specifies which alternative is chosen for each task, the total project duration $D(\sigma)$ is defined as the sum of times of the tasks along the critical path, and the total cost $B(\sigma)$ is defined as the sum of costs of all tasks. The objective is to find a realization σ^* that minimizes the total project duration $D(\sigma^*)$ or the total cost $B(\sigma^*)$.

The above DTCT problem is obviously bicriteria, and a tradeoff exists between the total project duration and the total cost. Two problem variants have been commonly studied, both of which are shown to be NP-complete [12]:

- *Budget Problem*: Given a total cost budget B, minimize the project duration $D(\sigma)$ subject to $B(\sigma) \leq B$;
- *Deadline Problem*: Given a project deadline D, minimize the total cost $B(\sigma)$ subject to $D(\sigma) \leq D$.

For both problems, Skutella [59] presented a polynomial-time approximation algorithm, which, given any feasible budget-deadline pair (B, D) and any $\rho \in (0, 1)$, finds a realization σ for the project that satisfies[4]:

$$D(\sigma) \leq \frac{D}{\rho} ,$$
$$B(\sigma) \leq \frac{B}{1-\rho} .$$

[4] In essence, Skutella's approximation algorithm first transforms each task of the project into a set of virtual tasks, and then constructs a relaxed Linear Program (LP) for the transformed problem. The relaxed LP either minimizes $D(\sigma)$ subject $B(\sigma) \leq B$ or minimizes $B(\sigma)$ subject $D(\sigma) \leq D$. In either case, the result can be obtained by rounding the optimal fractional solution to the relaxed LP based on the parameter ρ.

Allocating resources to jobs. We can now transform our resource allocation problem to the DTCT problem and solve it using the approximation result in [59]. To that end, a task j is created for each job j in the graph, with the set of alternatives for the task corresponding to the set of resource allocations for the job. The execution time $t_{j,i}$ of task j with alternative i is then defined as the execution time $t_j(p_j)$ of job j with the corresponding resource allocation p_j, and the cost $c_{j,i}$ is defined as the average area $a_j(p_j)$.

Let $\mathcal{S}$ denote the set of all $Q = \prod_{i=1}^{d} P^{(i)}$ possible resource allocations for a job. To ensure that Condition (1) in Definition 3 is satisfied, we discard, for each job j, the subset $\mathcal{D}_j \subset \mathcal{S}$ of *dominated* allocations, which is defined as:

$$\mathcal{D}_j = \{p_j \mid \exists q_j, t_j(q_j) < t_j(p_j) \text{ and } a_j(q_j) \leq a_j(p_j)\} , \tag{2}$$

and only use the remaining set of *non-dominated* allocations, denoted by $\mathcal{N}_j = \mathcal{S} \backslash \mathcal{D}_j$, to create the alternatives of the task. Thus, a realization σ for the project corresponds to a resource allocation decision $\mathbf{p}$ for the jobs. The total project duration $D(\sigma)$ then corresponds to the critical-path length $C(\mathbf{p})$ of the DAG, and the total cost $B(\sigma)$ corresponds to the average total area $A(\mathbf{p})$ of the jobs.

A resource allocation decision $\mathbf{p} = (p_1, p_2, \dots, p_n)$ is said to be *non-dominated* if the allocation for every job is non-dominated, i.e., $p_j \in \mathcal{N}_j$ for all $j = 1, \dots, n$. The following lemma shows that the makespan lower bound $L_{\min}$ can be achieved by a non-dominated resource allocation.

Lemma 2. *There exists a non-dominated resource allocation* $\mathbf{p}^* = (p_1^*, p_2^*, \dots, p_n^*)$ *that achieves* $L(\mathbf{p}^*) = L_{\min}$.

Proof. Suppose a resource allocation $\mathbf{q}^* = (q_1^*, q_2^*, \dots, q_n^*)$ achieves $L(\mathbf{q}^*) = L_{\min}$, and it contains a dominated allocation $q_j^* \in \mathcal{D}_j$ for a job j. Then, by replacing q_j^* with a non-dominated allocation $q_j'^* \in \mathcal{N}_j$ that dominates q_j^*, i.e., $t_j(q_j'^*) \leq t_j(q_j^*)$ and $a_j(q_j'^*) \leq a_j(q_j^*)$, we get a new resource allocation $\mathbf{q}'^* = (q_1^*, \dots, q_{j-1}^*, q_j'^*, q_{j+1}^*, \dots, q_n^*)$, which satisfies $A(\mathbf{q}'^*) \leq A(\mathbf{q}^*)$ and $C(\mathbf{q}'^*) \leq C(\mathbf{q}^*)$. This implies $L(\mathbf{q}'^*) \leq L(\mathbf{q}^*) = L_{\min}$. Repeating the process above for every job with a dominated allocation results in an overall non-dominated allocation $\mathbf{p}^*$ that achieves $L(\mathbf{p}^*) = L_{\min}$. □

We can now find a resource allocation $\mathbf{p}'$ for the jobs (or equivalently a realization σ' in the corresponding DTCT problem), with the following property.

Lemma 3. *For any* $\rho \in (0, 1)$, *a resource allocation* $\mathbf{p}' = (p_1', p_2', \dots, p_n')$ *can be found in polynomial time that satisfies:*

$$C(\mathbf{p}') \leq \frac{T_{\text{OPT}}}{\rho} , \tag{3}$$
$$A(\mathbf{p}') \leq \frac{T_{\text{OPT}}}{1-\rho} . \tag{4}$$

Proof. The result can be obtained by adapting the algorithm in [59], which minimizes the project duration (or total cost) subject to a known budget B (or deadline D) for the DTCT problem. Without knowing the value of the constraint a priori, we can still achieve the same approximations by adopting the technique used in [36] for the problem with a single resource type. Specifically, the relaxed LP originally formulated in [59] can be modified and applied to our problem as follows: minimize the lower bound $L(\mathbf{p})$ instead, subject to two additional constraints $C(\mathbf{p}) \leq L(\mathbf{p})$ and $A(\mathbf{p}) \leq L(\mathbf{p})$. Then, by rounding the optimal fractional solution $\bar{\mathbf{p}}^*$ to this modified LP, we can get a resource allocation $\mathbf{p}'$ that satisfies: $C(\mathbf{p}') \leq \frac{C(\bar{\mathbf{p}}^*)}{\rho} \leq \frac{L(\bar{\mathbf{p}}^*)}{\rho}$ and $A(\mathbf{p}') \leq \frac{A(\bar{\mathbf{p}}^*)}{1-\rho} \leq \frac{L(\bar{\mathbf{p}}^*)}{1-\rho}$. Since the optimal fractional solution $\bar{\mathbf{p}}^*$ must result in an objective not greater than the one achieved by any (non-dominated) integral solution $\mathbf{p}^*$, and based on Lemma 2, we have $L(\bar{\mathbf{p}}^*) \leq L(\mathbf{p}^*) = L_{\min}$. The result then directly follows by applying the makespan lower bound in Lemma 1. □

Adjusting resource allocation. Lastly, we adjust the resource allocation $\mathbf{p}'$ (obtained through Lemma 3 above) to get the final resource allocation $\mathbf{p}$ for the jobs. The aim is to limit the maximum resource utilization of any job under any resource type, thus facilitating more efficient list scheduling (described in Section 4.2). As with the case for a single type of resource [42,36], we choose a parameter $\mu \in (0, 0.5)$, and obtain the final resource allocation for each job j on each resource type i as follows:

$$p_j^{(i)} = \begin{cases} \lceil \mu P^{(i)} \rceil, & \text{if } p'^{(i)}_j > \lceil \mu P^{(i)} \rceil \\ p'^{(i)}_j, & \text{otherwise} \end{cases} \tag{5}$$

where $p'^{(i)}_j$ is the corresponding resource allocation in $\mathbf{p}'$.

A job j is *adjusted* if its final resource allocation p_j is reduced from the initial allocation p'_j in any resource type; otherwise, the job is *unadjusted*. The following lemma shows the properties of any adjusted job.

Lemma 4. *For any adjusted job j, its execution time satisfies:*

$$t_j(p_j) \le \frac{t_j(p'_j)}{\mu} . \tag{6}$$

Further, if the total amount of resource type i satisfies $P^{(i)} \ge \frac{1}{\mu^2}$, the job's area on resource type i satisfies:

$$a_j^{(i)}(p_j) \le d \cdot a_j(p'_j) . \tag{7}$$

Proof. For any adjusted job j, let $x_j^{(i)} = \frac{p'^{(i)}_j}{p_j^{(i)}}$ denote its resource reduction factor on any resource type i, and let $k = \arg\max_{i=1\dots d} x_j^{(i)}$ denote the resource type with the largest reduction factor.

Since the job's final resource allocation p_j is at most its initial allocation p'_j, i.e., $p_j \le p'_j$, and according to the adjustment procedure in Equation (5), we have $x_j^{(k)} \le \frac{P^{(k)}}{\lceil \mu P^{(k)} \rceil} \le \frac{1}{\mu}$. Thus, based on Assumption 3, we can prove the time bound:

$$t_j(p_j) \le \left(\max_{i=1\dots d} x_j^{(i)} \right) \cdot t_j(p'_j) = x_j^{(k)} \cdot t_j(p'_j) \le \frac{t_j(p'_j)}{\mu} .$$

To prove the area bound, we distinguish three cases.

Case (1): For resource type k with the largest reduction factor, we have $w_j^{(k)}(p_j) = p_j^{(k)} \cdot t_j(p_j) \le \frac{p'^{(k)}_j}{x_j^{(k)}} \cdot (x_j^{(k)} \cdot t_j(p'_j)) = p'^{(k)}_j \cdot t_j(p'_j) = w_j^{(k)}(p'_j)$. Thus, the area of the job on resource type k satisfies:

$$a_j^{(k)}(p_j) = \frac{w_j^{(k)}(p_j)}{P^{(k)}} \le \frac{w_j^{(k)}(p'_j)}{P^{(k)}} \le \sum_{\ell=1}^{d} \frac{w_j^{(\ell)}(p'_j)}{P^{(\ell)}} = d \cdot a_j(p'_j) .$$

Case (2): For any resource type $i \ne k$ with $p_j^{(i)} \le \lfloor \mu P^{(i)} \rfloor \le \mu P^{(i)}$, and since $p_j^{(k)} = \lceil \mu P^{(k)} \rceil \ge \mu P^{(k)}$, we have:

$$\begin{aligned} a_j^{(i)}(p_j) &= \frac{p_j^{(i)} \cdot t_j(p_j)}{P^{(i)}} \le \frac{\mu P^{(i)} \cdot t_j(p_j)}{P^{(i)}} \\ &\le \mu \cdot x_j^{(k)} \cdot t_j(p'_j) = \mu \cdot \frac{p'^{(k)}_j \cdot t_j(p'_j)}{p_j^{(k)}} \\ &\le \mu \cdot \frac{w_j^{(k)}(p'_j)}{\mu P^{(k)}} = \frac{w_j^{(k)}(p'_j)}{P^{(k)}} \\ &\le \sum_{\ell=1}^{d} \frac{w_j^{(\ell)}(p'_j)}{P^{(\ell)}} = d \cdot a_j(p'_j) . \end{aligned}$$

Case (3): For any resource type $i \ne k$ with $p_j^{(i)} = \lceil \mu P^{(i)} \rceil \le \mu P^{(i)} + 1$, by following the same derivation steps as in Case (2), we can get:

$$\begin{aligned} a_j^{(i)}(p_j) &\le \left(1 + \frac{1}{\mu P^{(i)}}\right) \frac{w_j^{(k)}(p'_j)}{P^{(k)}} \\ &= \frac{w_j^{(k)}(p'_j)}{P^{(k)}} + \frac{w_j^{(k)}(p'_j)}{\mu P^{(i)} P^{(k)}} \\ &\le \sum_{\ell=1}^{d} \frac{w_j^{(\ell)}(p'_j)}{P^{(\ell)}} - \frac{w_j^{(i)}(p'_j)}{P^{(i)}} + \frac{w_j^{(k)}(p'_j)}{\mu P^{(i)} P^{(k)}} \\ &= \sum_{\ell=1}^{d} \frac{w_j^{(\ell)}(p'_j)}{P^{(\ell)}} + \frac{t_j(p'_j)}{P^{(i)}} \left(\frac{p'^{(k)}_j}{\mu P^{(k)}} - p'^{(i)}_j \right) . \end{aligned}$$

Since $p'^{(k)}_j \le P^{(k)}$ and $p'^{(i)}_j \ge \lceil \mu P^{(i)} \rceil \ge \mu P^{(i)}$, we have $\frac{p'^{(k)}_j}{\mu P^{(k)}} - p'^{(i)}_j \le \frac{1}{\mu} - \mu P^{(i)}$, which is at most 0 when $P^{(i)} \ge \frac{1}{\mu^2}$. In this case, we have:

$$a_j^{(i)}(p_j) \le \sum_{\ell=1}^{d} \frac{w_j^{(\ell)}(p'_j)}{P^{(\ell)}} = d \cdot a_j(p'_j) .$$

This completes the proof of the lemma. □

Algorithm 1: Resource Allocation (Phase 1).

Input: For each job j, execution time $t_j(p_j)$ and average area $a_j(p_j)$ under all possible resource allocations; values of parameters ρ and μ.
Output: Resource allocation decision $\mathbf{p}=(p_1, p_2, \dots, p_n)$ for all jobs.
begin
(Step 1): For each job j, discard the subset $\mathcal{D}_j \subset \mathcal{S}$ of dominated resource allocations as defined in Equation (2);
(Step 2): Transform the resource allocation problem to the DTCT problem and adapt the algorithm in [59] to obtain an initial allocation decision $\mathbf{p}'$ that satisfies Equations (3) and (4);
(Step 3): For each job j and each resource type i, adjust the initial allocation in $\mathbf{p}'$ based on Equation (5) to obtain a final resource allocation decision $\mathbf{p}$ that satisfies Equations (6) and (7).
end

Algorithm 1 summarizes all the three steps involved in this first phase of the algorithm.

4.2. Phase 2: job scheduling

The second phase schedules the jobs by making a starting time decision $\mathbf{s} = (s_1, s_2, \dots, s_n)$, given the resource allocation decision $\mathbf{p}$ determined by the first phase.

List scheduling strategy. Jobs are scheduled through a list scheduling strategy, as shown in Algorithm 2, which extends the classical list scheduling for a single type of resource to multiple resource types.

A job is said to be *ready* if all of its immediate predecessors in the precedence graph have been completed or if the job has no immediate predecessor. The algorithm starts by inserting all ready jobs into a queue $\mathcal{Q}$ in any order. Then, at time 0 or whenever a running job k completes and hence releases resources, the algorithm inserts, into the queue $\mathcal{Q}$, any new job k' that becomes ready due to the completion of job k. It then goes through the list of all ready jobs in $\mathcal{Q}$ and schedules each job j that can be executed at the current time if its resource allocation p_j can be met by the amount of available resources in all resource types.

Properties of list scheduling. We now analyze some properties of list scheduling, which will be used later in the derivation of MRSA's approximation ratio. The analysis follows the same framework as in [42,36] for single-resource scheduling.

We start by defining some notations. Let T denote the makespan of a list schedule. We note that the algorithm only allocates and de-allocates resources upon job completions. Hence, the entire schedule's duration $[0, T]$ can be partitioned into a set $\mathcal{I} = \{I_1, I_2, \dots\}$ of non-overlapping intervals, where jobs only start (or complete) at the beginning (or end) of an interval, and the amount of utilized resource for any resource

Algorithm 2: List Scheduling (Phase 2).

Input: Resource allocation decision $\mathbf{p}=(p_1, p_2, \dots, p_n)$ for all jobs, and their precedence constraints.
Output: A list schedule for the jobs with starting time decision $\mathbf{s}=(s_1, s_2, \dots, s_n)$.

```
begin
    insert all ready jobs into a queue Q;
    P_avail^(i) ← P^(i), ∀i;
    when at time 0 or a job k completes execution do
        curr_time ← getCurrentTime();
        P_avail^(i) ← P_avail^(i) + p_k^(i), ∀i;
        for each job k' that becomes ready do
            insert job k' into queue Q;
        end
        for each job j ∈ Q do
            if P_avail^(i) ≥ p_j^(i), ∀i then
                s_j ← curr_time and execute job j now;
                P_avail^(i) ← P_avail^(i) − p_j^(i), ∀i;
                remove job j from queue Q;
            end
        end
    end
end
```

type does not change during an interval. For any resource type i, let $P_{util}^{(i)}(I)$ denote the total amount of utilized resources from all jobs that are running during interval $I \in \mathcal{I}$. We further classify the set of intervals into the following three categories.

- $\mathcal{I}_1$: set of intervals during which the amount of utilized resources is at most $\lceil \mu P^{(i)} \rceil - 1$ for all resource type i, i.e., $\mathcal{I}_1 = \{I \mid \forall i, P_{util}^{(i)}(I) \leq \lceil \mu P^{(i)} \rceil - 1\}$.
- $\mathcal{I}_2$: set of intervals during which there exists a resource type k that utilizes at least $\lceil \mu P^{(k)} \rceil$ amount of resources, but the amount of utilized resources is at most $\lceil (1-\mu) P^{(i)} \rceil - 1$ for all resource type i, i.e., $\mathcal{I}_2 = \{I \mid \exists k, P_{util}^{(k)}(I) \geq \lceil \mu P^{(k)} \rceil$ and $\forall i,\ P_{util}^{(i)}(I) \leq \lceil (1-\mu) P^{(i)} \rceil - 1\}$.
- $\mathcal{I}_3$: set of intervals during which there exists a resource type k that utilizes at least $\lceil (1-\mu) P^{(k)} \rceil$ amount of resources, i.e., $\mathcal{I}_3 = \{I \mid \exists k, P_{util}^{(k)}(I) \geq \lceil (1-\mu) P^{(k)} \rceil\}$.

Let $|I|$ denote the duration of an interval I, and let $T_1 = \sum_{I \in \mathcal{I}_1} |I|$, $T_2 = \sum_{I \in \mathcal{I}_2} |I|$ and $T_3 = \sum_{I \in \mathcal{I}_3} |I|$ be the total durations of the three categories of intervals, respectively. Since $\mathcal{I}_1$, $\mathcal{I}_2$ and $\mathcal{I}_3$ are obviously disjoint and partition $\mathcal{I}$, we have:

$$T = T_1 + T_2 + T_3 \ . \tag{8}$$

Furthermore, for each job j and each interval I, we define $\beta_{j,I}$ to be the *fraction* of the job executed during that interval. For instance, if one third of job j is executed in interval I and two thirds of the job are executed in interval I', we have $\beta_{j,I} = 1/3$ and $\beta_{j,I'} = 2/3$. Note that the fraction is defined in terms of either the execution time or the area (work) of the job, which are equivalent here since the resource allocation of the job has been fixed. Thus, for each job j, we have $\sum_{I \in \mathcal{I}} \beta_{j,I} = 1$.

The following lemma bounds the durations of the first two categories of intervals in terms of the execution time along the critical path of the initial resource allocation $\mathbf{p}'$.

Lemma 5 *(Critical-path bound). For any choice of $\mu \in (0, 0.5)$, we have:*

$$T_1 + \mu T_2 \leq C(\mathbf{p}') \ .$$

Proof. For any interval $I \in \mathcal{I}_1 \cup \mathcal{I}_2$, the amount of utilized resource for any resource type i is at most $\lceil (1-\mu) P^{(i)} \rceil - 1$, so the amount of available resource is at least $P^{(i)} + 1 - \lceil (1-\mu) P^{(i)} \rceil \geq \lceil \mu P^{(i)} \rceil$. According to the resource allocation algorithm, any job is allocated at most $\lceil \mu P^{(i)} \rceil$ amount of resource for resource type i. Thus, there is sufficient resource available to execute any additional job (if one is ready) during any interval $I \in \mathcal{I}_1 \cup \mathcal{I}_2$. This implies that there is no ready job in the queue $\mathcal{Q}$, since otherwise the list scheduling algorithm would have scheduled the job.

In list scheduling, it is known that there exists a path f in the graph such that whenever there is no ready job in the queue, some job along that path is running [22,42,36]. Thus, during any interval $I \in \mathcal{I}_1 \cup \mathcal{I}_2$, some job along path f is running, and we let $j(I) \in f$ denote such a job.

Now, consider the initial resource allocation $\mathbf{p}'$. During any interval $I \in \mathcal{I}_1$, the amount of utilized resource for any resource type i is at most $\lceil \mu P^{(i)} \rceil - 1$, so job $j(I)$ must be unadjusted. Thus, we have $t_{j(I)}(p_{j(I)}) = t_{j(I)}(p'_{j(I)})$. However, during any interval $I \in \mathcal{I}_2$, job $j(I)$ could be adjusted, and thus, according to Lemma 4 (Inequality (6)), we have $\mu \cdot t_{j(I)}(p_{j(I)}) \leq t_{j(I)}(p'_{j(I)})$. We can then derive:

$$\begin{aligned}
&T_1 + \mu T_2 \\
&= \sum_{I \in \mathcal{I}_1} t_{j(I)}(p_{j(I)}) \cdot \beta_{j(I),I} + \mu \sum_{I \in \mathcal{I}_2} t_{j(I)}(p_{j(I)}) \cdot \beta_{j(I),I} \\
&\leq \sum_{I \in \mathcal{I}_1} t_{j(I)}(p'_{j(I)}) \cdot \beta_{j(I),I} + \sum_{I \in \mathcal{I}_2} t_{j(I)}(p'_{j(I)}) \cdot \beta_{j(I),I} \\
&\leq \sum_{j \in f} \left(t_j(p'_j) \cdot \sum_{I \in \mathcal{I}_1 \cup \mathcal{I}_2} \beta_{j,I} \right) \\
&\leq \sum_{j \in f} t_j(p'_j) = C(\mathbf{p}', f) \leq C(\mathbf{p}') \ . \quad \square
\end{aligned}$$

The following lemma bounds the durations of the last two categories of intervals in terms of the average total area of the initial resource allocation $\mathbf{p}'$.

Lemma 6 *(Area bound). For any choice of $\mu \in (0, 0.5)$, if $P^{\min} = \min_i P^{(i)} \geq \frac{1}{\mu^2}$, we have:*

$$\mu T_2 + (1-\mu) T_3 \leq d \cdot A(\mathbf{p}') \ .$$

Proof. For any interval $I \in \mathcal{I}_2$, there exists a resource type i such that the amount of utilized resource is at least $\lceil \mu P^{(i)} \rceil$ based on the definition of $\mathcal{I}_2$. Therefore, the total work done on resource type i from all jobs during this interval satisfies: $\sum_{j=1}^{n} \beta_{j,I} \cdot w_j^{(i)}(p_j) \geq |I| \cdot \lceil \mu P^{(i)} \rceil \geq |I| \cdot \mu P^{(i)}$. Thus, we have: $\mu \cdot |I| \leq \sum_{j=1}^{n} \beta_{j,I} \cdot \frac{w_j^{(i)}(p_j)}{P^{(i)}} = \sum_{j=1}^{n} \beta_{j,I} \cdot a_j^{(i)}(p_j) \leq d \sum_{j=1}^{n} \beta_{j,I} \cdot a_j(p'_j)$. The last inequality is due to Lemma 4 (Inequality (7)), if $P^{(i)} \geq \frac{1}{\mu^2}$. Note that Inequality (7) was proven for any adjusted job but it obviously holds for unadjusted jobs as well. Thus, if $P^{\min} = \min_{i=1\dots d} P^{(i)} \geq \frac{1}{\mu^2}$, we can derive:

$$\begin{aligned}
\mu T_2 &= \mu \sum_{I \in \mathcal{I}_2} |I| \\
&\leq d \sum_{I \in \mathcal{I}_2} \sum_{j=1}^{n} \beta_{j,I} \cdot a_j(p'_j) \\
&= d \sum_{j=1}^{n} \left(a_j(p'_j) \cdot \sum_{I \in \mathcal{I}_2} \beta_{j,I} \right) \ .
\end{aligned} \tag{9}$$

For any interval $I \in \mathcal{I}_3$, there exists a resource type i such that the amount of utilized resource is at least $\lceil (1-\mu) P^{(i)} \rceil$. Using the same argument, we can derive:

$$(1-\mu) T_3 \leq d \sum_{j=1}^{n} \left(a_j(p'_j) \cdot \sum_{I \in \mathcal{I}_3} \beta_{j,I} \right) \ . \tag{10}$$

Thus, combining Inequalities (9) and (10), we can get:

$$\mu T_2 + (1-\mu)T_3 \le d \sum_{j=1}^{n} \left(a_j(p'_j) \cdot \sum_{I \in I_2 \cup I_3} \beta_{j,I} \right) \le d \sum_{j=1}^{n} a_j(p'_j) = d \cdot A(\mathbf{p}') \ . \quad \square$$

4.3. Approximation results

We now derive the approximation ratio of MRSA, which combines the resource allocation phase (Algorithm 1) and the list scheduling phase (Algorithm 2). The following theorem shows the result for any number d of resource types.

Theorem 1. *For any $d \ge 1$ and if $P^{\min} \ge 7$, the makespan of MRSA satisfies:*

$$\frac{T}{T_{\text{OPT}}} \le \phi d + 2\sqrt{\phi d} + 1 \le 1.619d + 2.545\sqrt{d} + 1 \ ,$$

where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. The result is achieved at $\mu^* = 1 - \frac{1}{\phi} \approx 0.382$ and $\rho^* = \frac{1}{\sqrt{\phi d}+1} \approx \frac{1}{1.272\sqrt{d}+1}$.

We point out that $P^{\min} \ge 7$ represents a reasonable condition on the total amount of most discrete resource types (e.g., processors, memory blocks, cache lines).

Proof. Based on the analysis of the list scheduling algorithm, by substituting T_1 from Lemma 5 and T_3 from Lemma 6 into $T = T_1 + T_2 + T_3$, and if $P^{\min} \ge \frac{1}{\mu^2}$, we get:

$$T \le C(\mathbf{p}') + \frac{d}{1-\mu} A(\mathbf{p}') + \left(1 - \mu - \frac{\mu}{1-\mu}\right) T_2 \ .$$

Applying the bounds for $C(\mathbf{p}')$ and $A(\mathbf{p}')$ in Lemma 3 from the resource allocation algorithm, and when $(1-\mu)^2 \le \mu$, i.e., $\mu \ge \frac{3-\sqrt{5}}{2} = 1 - \frac{1}{\phi}$, which makes the last term above at most zero, we can derive:

$$T \le \left(\frac{1}{\rho} + \frac{d}{(1-\mu)(1-\rho)} \right) T_{\text{OPT}} \triangleq f_d(\mu, \rho) \cdot T_{\text{OPT}} \ .$$

Clearly, $f_d(\mu, \rho)$ is an increasing function of μ. Thus, to minimize the function, we can set $\mu^* = 1 - \frac{1}{\phi}$. In this case, we require $P^{\min} \ge \frac{1}{(\mu^*)^2} \approx 6.854$ and we define $f_d(\rho) \triangleq f_d(\mu^*, \rho) = \frac{1}{\rho} + \frac{\phi d}{1-\rho}$. By setting $f'_d(\rho) = -\frac{1}{\rho^2} + \frac{\phi d}{(1-\rho)^2} = 0$ and by checking that $f''_d(\rho) > 0$ for all ρ, we get $\rho^* = \frac{1}{\sqrt{\phi d}+1}$ that minimizes $f_d(\rho)$. Thus, the approximation ratio is given by $f_d(\mu^*, \rho^*) = \phi d + 2\sqrt{\phi d} + 1$. $\square$

Remarks. When there is only one type of resource (i.e., $d = 1$), Theorem 1 gives an approximation ratio of 5.164, which improves upon the ratio of 5.236 by Lepère et al. [42]. In fact, Jansen and Zhang [36] proved an even better ratio of 4.73 by deriving a tighter critical-path bound than the one shown in Lemma 5. Unfortunately, their analysis cannot be generalized to work for the case with more than one type of resources.

While Theorem 1 proves the approximation ratio of the algorithm for any d, the following theorem shows an improved result for large d.

Theorem 2. *For any $d \ge 22$ and if $P^{\min} \ge d^{2/3}$, the makespan of MRSA satisfies:*

$$\frac{T}{T_{\text{OPT}}} \le d + 3\sqrt[3]{d^2} + O(\sqrt[3]{d}) \ ,$$

which is achieved at $\mu^* \approx \frac{1}{\sqrt[3]{d}}$ and $\rho^* = \frac{\sqrt{1-2\mu^*}}{\sqrt{1-2\mu^*}+\sqrt{d\mu^*}}$.

Proof. Following the proof of Theorem 1 but by substituting T_2 and T_3 into Equation (8), and if $P^{\min} \ge \frac{1}{\mu^2}$, we get:

$$T \le \frac{1-2\mu}{\mu(1-\mu)} C(\mathbf{p}') + \frac{d}{1-\mu} A(\mathbf{p}') + \left(1 - \frac{1-2\mu}{\mu(1-\mu)}\right) T_1 \ .$$

Applying the bounds for $C(\mathbf{p}')$ and $A(\mathbf{p}')$ in Lemma 3, and when $1 - \frac{1-2\mu}{\mu(1-\mu)} \le 0$, i.e., $\mu \le \frac{3-\sqrt{5}}{2} = 1 - \frac{1}{\phi}$, which makes the last term above at most zero, we can derive:

$$T \le \left(\frac{1-2\mu}{\mu(1-\mu)\rho} + \frac{d}{(1-\mu)(1-\rho)} \right) T_{\text{OPT}} \triangleq g_d(\mu, \rho) \cdot T_{\text{OPT}} \ .$$

Let $X_\mu = \frac{1-2\mu}{\mu(1-\mu)} = \frac{1}{\mu} - \frac{1}{1-\mu}$ and $Y_\mu = \frac{1}{1-\mu}$. We can then write: $g_d(\mu, \rho) = \frac{X_\mu}{\rho} + \frac{dY_\mu}{1-\rho}$. By deriving $g_d(\mu, \rho)$ with respect to ρ and setting the derivative to zero, we can get the best choice for ρ to be $\rho^*(\mu) = \frac{\sqrt{X_\mu}}{\sqrt{X_\mu}+\sqrt{dY_\mu}}$. As $X_\mu, Y_\mu > 0$, clearly $\rho^*(\mu) \in (0,1)$, thus is a valid choice. By substituting $\rho^*(\mu)$ back into $g_d(\mu, \rho)$ and simplifying, we can get:

$$g_d(\mu, \rho^*(\mu)) = \left(\sqrt{X_\mu} + \sqrt{dY_\mu}\right)^2 \triangleq g_d(\mu)^2 \ .$$

We will now minimize $g_d(\mu) = \sqrt{\frac{1}{\mu} - \frac{1}{1-\mu}} + \sqrt{\frac{d}{1-\mu}}$. By deriving $g_d(\mu)$ with respect to μ and factoring, we can get:

$$g'_d(\mu) = -\frac{h_d(\mu)}{r_d(\mu)} \ ,$$

where

$$h_d(\mu) = (2d+4)\mu^4 - (d+8)\mu^3 + 8\mu^2 - 4\mu + 1 \ ,$$

$$r_d(\mu) = 2\mu(1-\mu)\sqrt{\mu(1-\mu)(1-2\mu)}\left(\mu\sqrt{d\mu(1-2\mu)} + (2\mu^2 - 2\mu + 1)\right) \ .$$

As $2\mu^2 - 2\mu + 1 = \mu^2 + (1-\mu)^2 > 0$ for any $\mu \in (0, 0.5)$, $r_d(\mu)$ is always positive. Thus, the sign of $g'_d(\mu)$ is the opposite of the sign of $h_d(\mu)$.

In the following, we will show that, if $d \le 21$, $h_d(\mu)$ is always positive for any $\mu \in (0, \frac{3-\sqrt{5}}{2}]$, and thus the optimal choice is $\mu^* = \frac{3-\sqrt{5}}{2}$, which gives the same result as in Theorem 1. Otherwise, if $d \ge 22$, there is a unique optimal choice $\mu^* \in (0, \frac{3-\sqrt{5}}{2})$, which satisfies $h_d(\mu^*) = 0$. For convenience, we define $\mu^A = \frac{3-\sqrt{5}}{2}$ and $\mu^B = \frac{3}{8} < \mu^A$.

First, we can compute, for any $\mu \in (0, \mu^B]$, that:

$$\begin{aligned} h'_d(\mu) &= 4(2d+4)\mu^3 - 3(d+8)\mu^2 + 16\mu - 4 \\ &= d\mu^2(8\mu - 3) + 4(2\mu - 1)\left(\mu^2 + (1-\mu)^2\right) < 0 \ . \end{aligned}$$

We can also compute, for any $\mu \in [\mu^B, \mu^A]$, that:

$$\begin{aligned} h''_d(\mu) &= 12(2d+4)\mu^2 - 6(d+8)\mu + 16 \\ &\ge 12(2d+4) \cdot \left(\frac{3}{8}\right)^2 - 6(d+8) \cdot \left(\frac{3-\sqrt{5}}{2}\right) + 16 \\ &\approx 1.083d + 4.416 > 0 \end{aligned}$$

Thus, we can conclude the following:

- In $(0, \mu^B]$, $h_d(\mu)$ is a strictly decreasing function of μ;
- In $[\mu^B, \mu^A]$, $h_d(\mu)$ is a strictly convex function of μ, and $h'_d(\mu)$ is a strictly increasing function of μ.

We can verify that, when $d \le 21$, the value of μ^* as suggested in Theorem 1 remains the optimal choice that minimizes $g_d(\mu)$, and it yields the same approximation result of Theorem 1.

We now focus on the case with $d \ge 22$. For any fixed μ in $(0, \mu^A]$, we can easily show that $h_d(\mu)$ is a decreasing function of d (by deriving $h_d(\mu)$ with respect to d). Thus, we have $h_d(\mu^B) \le h_{22}(\mu^B) \approx -0.008 < 0$. Further, we have $h_d(0) = 1 > 0$. Since $h_d(\mu)$ is a strictly decreasing function of μ in $(0, \mu^B]$, we know that $h_d(\mu) = 0$ admits a unique solution μ^*

in this interval. Moreover, since $h_d(\mu)$ is a convex function in $[\mu^B, \mu^A]$, we have, for any $\mu \in [\mu^B, \mu^A]$, that:

$$h_d(\mu) \le h_{22}(\mu) \le \max\left(h_{22}(\mu^B), h_{22}(\mu^A)\right) \approx \max(-0.008, -0.01) < 0 .$$

This shows that $h_d(\mu) > 0$ in $(0, \mu^*)$ and $h_d(\mu) < 0$ in $(\mu^*, \mu^A]$. Since $h_d(\mu)$ and $g'_d(\mu)$ have opposite signs, we get that $g_d(\mu)$ is a strictly decreasing function of μ in $(0, \mu^*)$ and a strictly increasing function in $(\mu^*, \mu^A]$. Thus, the optimal μ to minimize $g_d(\mu)$ is given by μ^*.

As μ^* is the solution to a fourth-degree equation (i.e., $h_d(u) = 0$), its closed form, although exists, is too complicated to express. However, observing that when d increases and if μ is small enough, the dominating negative term of $h_d(\mu)$ is $d\mu^3$ and the dominating positive term is 1. We can then get an estimate of $\mu^* \approx \frac{1}{\sqrt[3]{d}}$, which gives the following approximation ratio:

$$g_d(\mu^*)^2 \approx \frac{d\sqrt[3]{d} + 2d\sqrt{1 - \frac{2}{\sqrt[3]{d}}} + \sqrt[3]{d^2} - 2\sqrt[3]{d}}{\sqrt[3]{d} - 1} = d + 3\sqrt[3]{d^2} + O(\sqrt[3]{d}) .$$

This completes the proof of the theorem. □

Remarks. Theorem 2 holds for a relatively large number of resource types (i.e., $d \ge 22$), which is unlikely to be practical in today's resource management systems. However, the theoretical result offers the first approximation ratio whose dominating factor (i.e., d) matches the lower bound for local list-based scheduling (see Theorem 6). Thus, the result is asymptotically tight for this class of multi-resource moldable job scheduling algorithms.

5. Improved approximation results of MRSA for some special graphs

In the preceding section, we have derived the approximation ratios of MRSA for general graphs. In this section, we will show improved approximation results for some special graphs, namely, series-parallel graphs or trees, and independent jobs without any precedence constraints.

5.1. Results for SP graphs or trees

We first consider jobs whose precedence constraints form a series-parallel graph or a tree. A directed acyclic graph is a *Series-Parallel (SP) graph* [6] if it has only two nodes (i.e., a source and a sink) connected by an edge, or can be constructed (recursively) by a series composition or a parallel composition of two SP graphs.[5] Trees are simply special cases of general SP graphs.

In this case, we rely on an FPTAS (Fully Polynomial-Time Approximation Scheme) proposed in [42] to find a near-optimal resource allocation. The algorithm was proposed in the context of single-resource scheduling, but can be readily adapted to work for multiple types of resources (while first discarding the subset of dominated resource allocations as shown in Step 1 of Algorithm 1).[6] The following lemma shows the result. More details about the algorithm can be found in [42].

[5] Given two SP graphs G_1 and G_2, the *parallel composition* is the union of the two graphs while merging their sources to create the new source and merging their sinks to create the new sink, and the *series composition* merges the sink of G_1 with the source of G_2 and uses the source of G_1 as the new source and the sink of G_2 as the new sink.

[6] In essence, the FPTAS first decomposes an SP graph into atomic parts, then uses dynamic programming to decide if an allocation $\mathbf{p}'$ that satisfies $L(\mathbf{p}') \le X$ can be found for a positive integer X, and finally performs a binary search on X.

Lemma 7. *For any set of jobs whose precedence constraints form an SP graph or a tree, and for any $\epsilon \ge 0$, there exists an FPTAS (with running time polynomial in $1/\epsilon$), which computes a resource allocation $\mathbf{p}'$ that satisfies:*

$$L(\mathbf{p}') = \max(A(\mathbf{p}'), C(\mathbf{p}')) \le (1+\epsilon) \cdot L_{\min} \le (1+\epsilon) \cdot T_{\text{OPT}} .$$

We can now use the above FPTAS to replace Step 2 in resource allocation (Algorithm 1) and combine it with list scheduling (Algorithm 2). The following theorem shows the approximation ratio for any number d of resource types.

Theorem 3. *For any $d \ge 1$ and if $P^{\min} \ge 7$, the makespan of MRSA for SP graphs or trees satisfies:*

$$\frac{T}{T_{\text{OPT}}} \le (1+\epsilon) \cdot (\phi d + 1) \le (1+\epsilon) \cdot (1.619d + 1) ,$$

where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. The result is achieved at $\mu^* = 1 - \frac{1}{\phi} \approx 0.382$.

Proof. Following the proof of Theorem 1 by substituting T_1 from Lemma 5 and T_3 from Lemma 6 into $T = T_1 + T_2 + T_3$, and if $P^{\min} \ge \frac{1}{\mu^2}$, we get:

$$T \le C(\mathbf{p}') + \frac{d}{1-\mu} A(\mathbf{p}') + \left(1 - \mu - \frac{\mu}{1-\mu}\right) T_2 .$$

Then, by applying the bounds in Lemma 7, and when $(1-\mu)^2 \le \mu$, i.e., $\mu \ge \frac{3-\sqrt{5}}{2} = 1 - \frac{1}{\phi}$, we can derive:

$$T \le (1+\epsilon) \cdot \left(1 + \frac{d}{(1-\mu)}\right) T_{\text{OPT}} \triangleq f_d(\mu) \cdot T_{\text{OPT}} .$$

Clearly, $f_d(\mu)$ is an increasing function of μ. Thus, the minimum value is obtained by setting $\mu^* = 1 - \frac{1}{\phi}$. In this case, the approximation ratio is given by $f_d(\mu^*) = (1+\epsilon) \cdot (\phi d + 1)$, with the condition $P^{\min} \ge \frac{1}{(\mu^*)^2} \approx 6.854$. □

The approximation ratio can be further improved for $d \ge 4$ resource types, as shown in the following theorem.

Theorem 4. *For any $d \ge 4$ and if $P^{\min} \ge d + 2\sqrt{d-1}$, the makespan of MRSA for SP graphs or trees satisfies:*

$$\frac{T}{T_{\text{OPT}}} \le (1+\epsilon) \cdot \left(d + 2\sqrt{d-1}\right) ,$$

which is achieved at $\mu^* = \frac{1}{\sqrt{d-1}+1}$.

Proof. Following the proof of Theorem 1 but by substituting T_2 and T_3 into $T = T_1 + T_2 + T_3$, and if $P^{\min} \ge \frac{1}{\mu^2}$, we get:

$$T \le \frac{1-2\mu}{\mu(1-\mu)} C(\mathbf{p}') + \frac{d}{1-\mu} A(\mathbf{p}') + \left(1 - \frac{1-2\mu}{\mu(1-\mu)}\right) T_1 .$$

Applying the bounds in Lemma 7, and when $1 - \frac{1-2\mu}{\mu(1-\mu)} \le 0$, i.e., $\mu \le \frac{3-\sqrt{5}}{2}$, we can derive:

$$T \le (1+\epsilon) \cdot \left(\frac{1-2\mu}{\mu(1-\mu)} + \frac{d}{1-\mu}\right) T_{\text{OPT}} = (1+\epsilon) \cdot \left(\frac{1}{\mu} + \frac{d-1}{1-\mu}\right) T_{\text{OPT}} \triangleq g_d(\mu) \cdot T_{\text{OPT}} .$$

By setting $g'_d(\mu) = -\frac{1}{\mu^2} + \frac{d-1}{(1-\mu)^2} = 0$ and by checking that $g''_d(\mu) > 0$, we get $\mu^* = \frac{1}{\sqrt{d-1}+1}$, which is at most $\frac{3-\sqrt{5}}{2}$ for $d \ge 4$. Thus, with the con-

dition $P^{\min} \geq \frac{1}{(\mu^*)^2} = d + 2\sqrt{d-1}$ and $d \geq 4$, we get the approximation ratio:

$$g_d(\mu^*) = (1+\epsilon) \cdot \left(\sqrt{d-1} + 1 + \frac{d-1}{1 - \frac{1}{\sqrt{d-1}+1}} \right)$$
$$= (1+\epsilon) \cdot \left(d + 2\sqrt{d-1} \right) . \quad \square$$

5.2. Results for independent jobs

We finally consider a set of independent jobs without any precedence constraints. For this case, Sun et al. [61] presented a $2d$-approximation algorithm for any $d \geq 1$, while we show improved results for $d \geq 3$. Here, we rely on an optimal multi-resource allocation algorithm proposed in [61] as Step 2 of our Algorithm 1. The algorithm computes the resource allocation in polynomial time as shown in the lemma below. More details of the algorithm can be found in [61].

Lemma 8. *For any set of independent jobs, a resource allocation $\mathbf{p}'$ can be computed in polynomial time that satisfies:*

$$L(\mathbf{p}') = \max(A(\mathbf{p}'), C(\mathbf{p}')) = L_{\min} \leq T_{\text{OPT}} ,$$

where $C(\mathbf{p}') = \max_{j=1\ldots n} t_j(p'_j)$ denotes the maximum execution time of any job under allocation $\mathbf{p}'$, which becomes the critical path when there is no precedence constraint.

For a set of independent jobs, while the area bound (Lemma 6) remains unchanged, in the following we provide a modified critical-path bound.

Lemma 9 *(Modified critical-path bound). For any choice of $\mu \in (0, 0.5)$, we have:*

- *If $\mathcal{I}_1 = \emptyset$, $\mu T_2 \leq C(\mathbf{p}')$;*
- *If $\mathcal{I}_1 \neq \emptyset$, $T_1 + T_2 \leq C(\mathbf{p}')$.*

Proof. Recall that there are three categories of intervals $\mathcal{I}_1$, $\mathcal{I}_2$ and $\mathcal{I}_3$. Based on the proof of Lemma 5, during any interval $I \in \mathcal{I}_1 \cup \mathcal{I}_2$, there is no ready job in the queue. Since all jobs are independent, it means that all jobs have been scheduled. This implies that all intervals in $\mathcal{I}_2$ happen before all intervals in $\mathcal{I}_1$, since there is no new job arrival and jobs only complete. Further, all intervals in $\mathcal{I}_3$ happen before all intervals in $\mathcal{I}_2$ using the same argument. Now, consider a job j that completes the last in the schedule. We know that j must have started during $\mathcal{I}_3$ or at the beginning of $\mathcal{I}_2$. We consider two cases.

Case (1): $\mathcal{I}_1 = \emptyset$. In this case, job j is executed during all intervals in $\mathcal{I}_2$ and it could be adjusted. Thus, according to Lemma 4 (Inequality (6)), we have $\mu T_2 \leq \mu \cdot t_j(p_j) \leq t_j(p'_j) \leq \max_{j=1\ldots n} t_j(p'_j) = C(\mathbf{p}')$.

Case (2): $\mathcal{I}_1 \neq \emptyset$. In this case, job j is executed during all intervals in $\mathcal{I}_2$ and all intervals in $\mathcal{I}_1$, so it must be unadjusted (since it is executed during $\mathcal{I}_1$). Thus, we have $T_1 + T_2 \leq t_j(p_j) = t_j(p'_j) \leq \max_{j=1\ldots n} t_j(p'_j) = C(\mathbf{p}')$. $\square$

The following theorem summarizes the best approximation ratios that could be obtained for independent jobs with different number d of resource types.

Theorem 5. *The makespan of multi-resource scheduling for any set of independent jobs satisfies $T/T_{\text{OPT}} \leq r$, where:*

$$r = \begin{cases} 2d, & \text{if } d = 1, 2, \text{ and } P^{\min} \geq 1 \\ 1.619d + 1, & \text{if } d = 3, \text{ and } P^{\min} \geq 7 \\ d + 2\sqrt{d-1}, & \text{if } d \geq 4, \text{ and } P^{\min} \geq d + 2\sqrt{d-1} \end{cases}$$

Proof. When $d = 1, 2$, we can just apply the multi-resource scheduling algorithm in [61] to get $2d$-approximation. Otherwise, we consider both cases as stated in Lemma 9.

Case (1): $\mathcal{I}_1 = \emptyset$. In this case, the makespan is given by $T = T_2 + T_3$. Substituting $\mu T_2 \leq C(\mathbf{p}')$ from Lemma 9 and $\mu T_2 + (1-\mu)T_3 \leq d \cdot A(\mathbf{p}')$ from Lemma 6 into T, we get:

$$T \leq \frac{1-2\mu}{\mu(1-\mu)} C(\mathbf{p}') + \frac{d}{1-\mu} A(\mathbf{p}')$$
$$\leq \left(\frac{1-2\mu}{\mu(1-\mu)} + \frac{d}{1-\mu} \right) \cdot T_{\text{OPT}} \qquad \text{(by Lemma 8)}$$
$$\triangleq g_d(\mu) \cdot T_{\text{OPT}} .$$

Case (2): $\mathcal{I}_1 \neq \emptyset$. In this case, the makespan is given by $T = T_1 + T_2 + T_3$. Substituting $T_1 + T_2 \leq C(\mathbf{p}')$ from Lemma 9 and $\mu T_2 + (1-\mu)T_3 \leq d \cdot A(\mathbf{p}')$ from Lemma 6 into T, we get:

$$T \leq C(\mathbf{p}') + \frac{d}{1-\mu} A(\mathbf{p}') - \frac{\mu}{1-\mu} T_2$$
$$\leq \left(1 + \frac{d}{1-\mu} \right) \cdot T_{\text{OPT}} \qquad \text{(by Lemma 8)}$$
$$\triangleq f_d(\mu) \cdot T_{\text{OPT}} .$$

Based on the two cases above, the overall approximation ratio is thus given by $\max(f_d(\mu), g_d(\mu))$, with the condition $P^{\min} \geq \frac{1}{\mu^2}$. Thus, when $d = 3$, by following the proof of Theorem 3 and setting $\mu^* \approx 0.382$, the ratio is $f_d(\mu^*) \leq 1.619d + 1$. When $d \geq 4$, we can follow the proof of Theorem 4 by setting $\mu^* = \frac{1}{\sqrt{d-1}+1}$. In this case, the ratio is $g_d(\mu^*) = d + 2\sqrt{d-1}$. $\square$

6. Lower bound for local list-based scheduling

In this section, we prove a lower bound on the approximation ratio of any deterministic algorithm that, in the second phase, uses local list-based scheduling to schedule the jobs. This means that the algorithm will only consider the *local* characteristics of the jobs, such as their execution times or areas, when prioritizing them in the list/queue and will not take any *global* characteristics, such as the jobs' relative positions in the precedence graph, into consideration. The following theorem shows this lower bound, which holds regardless of the resource allocation scheme for the first phase.

Theorem 6. *Any deterministic list scheduling algorithm with local job priority considerations is no better than d-approximation for the multi-resource scheduling problem.*

Proof. The lower bound is constructed by using a set of jobs whose precedence constraints form a tree. Each job takes unit-time to complete, and only requires a unit resource allocation from a single resource type. For each resource type i, there is a total amount $P^{(i)} = 2$ of available resource. Fig. 1 illustrates our lower bound instance with $n = 2Md$ jobs, where M is an integer multiple of 3. The nodes represent the jobs, the arrows represent the precedence constraints, and the color of a node represents the single resource type the corresponding job requires. The jobs form a grid, and the job in the j-th row and k-th column is denoted by (j, k). They satisfy the following properties:

- The number of jobs in each row of the grid is defined as follows. For all $j \in [1, 2(d-1)]$, we have $k \in [1, M]$. For all $j \in [2d-1, 2d+1]$, we have $k \in [1, \frac{2M}{3}]$.
- For all $j \geq 1$ and $k > 1$, job (j, k) requires resource type $i = \min(\lceil \frac{j}{2} \rceil, d)$. For all $j > 2$, job $(j, 1)$ requires resource type $i = \lceil \frac{j}{2} \rceil - 1$, while jobs $(1, 1)$ and $(1, 2)$ require resource type $i = 1$.
- The jobs have the following dependencies. For all $j \geq 1$ and $k \geq 1$, we have $(j, k) \to (j, k+1)$. For $j \in [1, d-1]$, we have $(2j, 1) \to (2j+1, 1)$ and $(2j, 1) \to (2j+2, 1)$. Finally, we have $(2d, 1) \to (2d+1, 1)$.

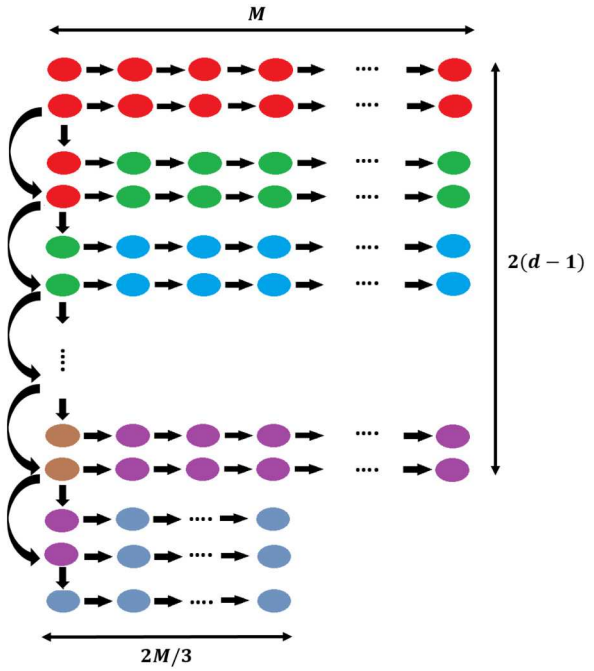


Fig. 1. Lower bound instance with an approximation ratio of d for any deterministic list-based scheduling algorithm with local job priority considerations.

The optimal schedule can be obtained by prioritizing the job dependencies going downward, thus enabling more resource types to be utilized concurrently. This results in a makespan of $T_{\text{OPT}} = M + d - 1$. Any deterministic list scheduling algorithm with only local priority considerations cannot distinguish jobs that require the same resource type. Hence, in the worst-case, it could only utilize one type of resource at any time by prioritizing horizontal job dependencies. This results in a makespan of $T = M(d-1) + \frac{4M}{3} = Md + \frac{M}{3}$. Choosing $M > 3(d^2 - d)$, the worst-case approximation ratio is given by:

$$\frac{T}{T_{\text{OPT}}} = \frac{Md + \frac{M}{3}}{M + d - 1} = \frac{d + \frac{1}{3}}{1 + \frac{d-1}{M}} > \frac{d + \frac{1}{3}}{1 + \frac{1}{3d}} = d \ .$$

This completes the proof of the theorem. □

Remarks. Theorem 6 implies that MRSA, when handling a large number of resource types as shown in Theorem 2, essentially achieves a tight approximation ratio up to the dominating factor (i.e., d) among the generic class of local list-based scheduling algorithms.

7. Simulation results

This section presents the results obtained by using simulations for evaluating the performance of MRSA and comparing it against two heuristic algorithms. For reproducibility purpose, the simulation code is publicly available at: https://gitlab.inria.fr/luperoti/mrsa.

7.1. Simulation setup

Workflow generation. In our simulations, we focus on evaluating the algorithms under general graphs/workflows (not special graphs such as trees or SP graphs). For that purpose, we use DAGGEN [62], a synthetic task graph generator capable of generating DAGs of different structures. DAGGEN has been previously used for evaluating single-resource scheduling algorithms for moldable jobs [33,15]. The graphs generated by DAGGEN have their jobs organized in layers, and important parameters that influence the structure of the graphs are described below.

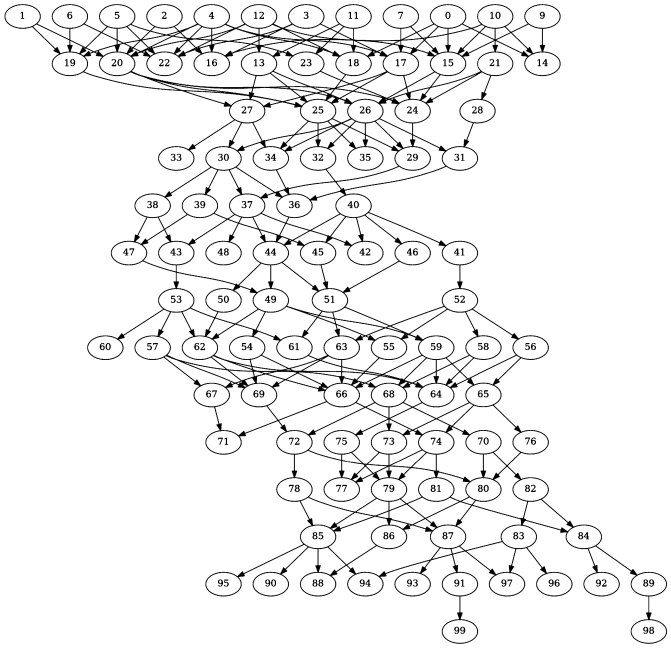


Fig. 2. A graph consisting of 100 jobs generated by DAGGEN with fat = 0.5, density = 0.5, regular = 0.5, and jump = 1.

- *fat*: controls the width of the DAG, i.e., the maximum number of jobs that can be executed concurrently;
- *density*: determines the number of dependencies between jobs of two consecutive layers of the DAG;
- *regular*: specifies the regularity of the distribution of jobs between different layers of the DAG;
- *jump*: controls the maximum number of layers that can be spanned by the edges of the DAG.

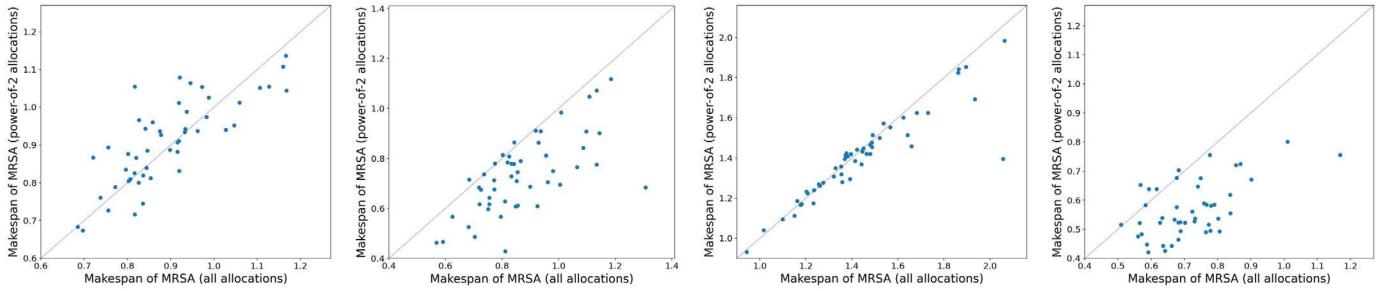
The range of possible values for fat, density, and regular is between 0 and 1, and jump can take any integer at least 1. In our default simulation setting, we will choose fat = 0.5, density = 0.5, regular = 0.5, and jump = 1. In Section 7.4, we will also vary these parameters to evaluate their impacts on the performance of the algorithms. Fig. 2 shows a graph consisting of 100 jobs generated by DAGGEN under this setting.

Job speedup models. We extend some common speedup models to define how resources of different types interact and contribute to the overall speedup of a moldable job. In particular, we adopt the following execution time functions for the jobs proposed in [61] that extend the classical Amdahl's law [1] and power law [51] speedup models.

- *Amdahl-Sum*: $t(p) = W\left(s_0 + \sum_{i=1}^{d} \frac{s_i}{p^{(i)}}\right)$;
- *Amdahl-Max*: $t(p) = W\left(s_0 + \max_{i=1..d} \frac{s_i}{p^{(i)}}\right)$;
- *Power-Sum*: $t(p) = W\left(\sum_{i=1}^{d} \frac{s_i}{(p^{(i)})^{\alpha_i}}\right)$;
- *Power-Max*: $t(p) = W\left(\max_{i=1..d} \frac{s_i}{(p^{(i)})^{\alpha_i}}\right)$.

In all the models above, W denotes the total amount of work to be completed by the job, and s_i denotes the fraction of work for the resource type i. For the two Amdahl models, s_0 denotes the sequential fraction that is not affected by the resource allocations. For the two power models, α_i denotes the efficiency factor for the utilization of the resource type i. We note that all of these speedup models above satisfy the monotonic job assumption stated in Section 3.1.

In our simulations, the sequential fraction s_0 is uniformly generated in $(0, 0.2]$, and the fraction s_i for each resource type i is uniformly



(a) Amdahl-Sum (b) Amdahl-Max (c) Power-Sum (d) Power-Max

Fig. 3. Scatter plots showing the makespans of MRSA for 50 workflows with $n = 30$ jobs, $d = 3$ resource types, and $P^{(i)} = 64$ for each resource type under the four speedup models. Each point represents a workflow, the x-axis represents the makespan when considering all possible resource allocations, and the y-axis represents the corresponding makespan when using only power-of-2 allocations.

generated in $(0, 1]$ and then normalized such that $\sum_{i=1}^{d} s_i = 1 - s_0$. The efficiency factor α_i is uniformly generated in $[0.3, 1)$. Finally, the total work W is uniformly generated in $(0, 1]$. These follow the same parameter choices as in [61]. In Section 7.5, we will also vary the ranges for the sequential fraction s_0 and the efficiency factor α_i to evaluate their impacts on the performance of the evaluated algorithms.

Comparing algorithms. To the best of our knowledge, we are not aware of any previously proposed multi-resource scheduling algorithms for moldable workflows. Hence, we compare our algorithm MRSA against two baseline heuristics, which are described below.[7]

- *minTime*: allocates resources to minimize the execution time of each job;
- *minArea*: allocates resources to minimize the average area of each job.

Both minTime and minArea also use list scheduling to schedule the jobs (in Phase 2). Thus, they only differ from MRSA in how resources are allocated (in Phase 1). For all the evaluated algorithms, we use the LPT (Longest Processing Time) priority rule to order the jobs in the list schedule, which is known to work well for reducing the makespan. Thus, if the waiting queue contains more than one job, these jobs will be ordered by non-increasing order of execution time given their resource allocations.

While minTime and minArea could compute the resource allocations for a job relatively efficiently,[8] MRSA would take $O(\Pi_{i=1}^{d} P^{(i)})$ time by examining all possible resource allocations. Although this remains polynomial in the input size, when the total amount of available resources in the system is large, the complexity of MRSA can be quite high, making simulations feasible only for small problem instances. Thus, to speed up the simulations, we consider only the power-of-2 choices (i.e., 1, 2, 4, 8, ...) when computing MRSA's resource allocation for each resource type. This leads to a factor of 2 increase in the approximation ratio of the algorithm in the worst case,[9] but it drastically reduces the complexity of the algorithm to $O(\Pi_{i=1}^{d} \lg P^{(i)})$, thus allowing to simulate larger problem instances in a reasonable amount of time.

[7] In Section 7.7, we also compare MRSA with a few other approximation algorithms that are designed for scheduling a single type of resource.

[8] Given the monotonic job assumption, minTime could allocate all the available resources to a job for minimizing its execution time, while minArea would typically allocate a small amount of resource to a job for minimizing its area.

[9] For any job with resource allocation $p_j = (p_j^{(1)}, \cdots, p_j^{(d)})$, the allocation obtained by rounding all the $p_j^{(i)}$'s to the closest higher power-of-2 results in a smaller time and an area at most twice as high. Therefore, the best trade-off achievable using only power-of-2 choices is at most twice as high as $L(\mathbf{p}^*) = L_{\min}$, and the rest of the analysis still holds with this extra factor of 2.

The scatter plots in Fig. 3 show the makespans of MRSA for 50 workflows with $n = 30$ jobs, $d = 3$ resource types, and $P^{(i)} = 64$ for each resource type under the four speedup models. In the plots, each point represents a workflow, the x-axis represents the makespan when MRSA considers all resource allocations, and the y-axis represents the corresponding makespan when MRSA uses only power-of-2 allocations. When running the simulations on a laptop, it took more than 10 hours for each speedup model by considering all allocations, while it took only a few seconds by considering power-of-2 allocations. In terms of the makespan, from the figure, we can see a generally strong and positive correlation between the two allocation schemes for the Amdhal-Sum and Power-Sum models. In the case of Amdahl-Max and Power-Max models, the makespans obtained when using power-of-2 allocations are even better than those obtained when using all allocations for most workflows. This is possibly because power-of-2 allocations potentially allow the ready jobs to be better packed/scheduled in the second phase of the algorithm. Given these results and to enable faster simulations, we will use power-of-2 allocations for MRSA in all the experiments.

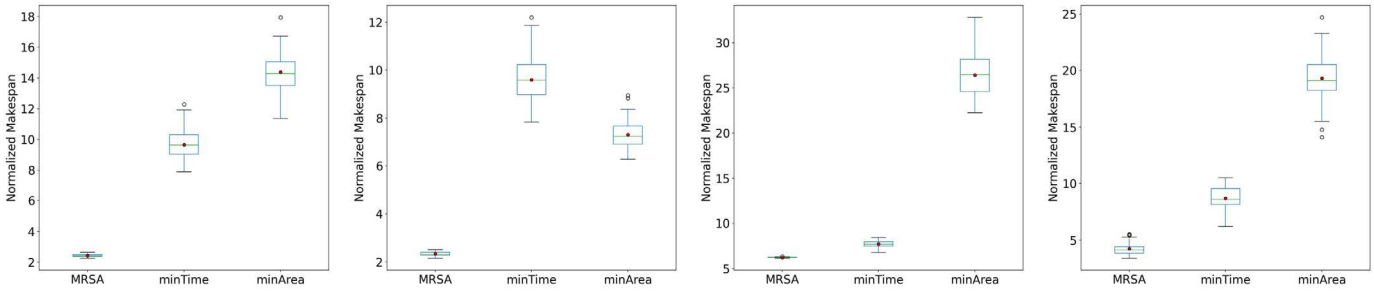
7.2. Performance comparison of algorithms

We first evaluate and compare the performance of the three algorithms (MRSA, minTime and minArea) for workflows that contain $n = 100$ jobs using $d = 3$ types of resources. Each resource type i has up to $P = 1024$ amount of available resources, and we consider the following two scenarios.

- *Uniform P*: the total amount of available resources across all resource types is the same. In this experiment, we set $P^{(i)} = 256$ for all $1 \le i \le d$.
- *Non-uniform P*: the total amount of available resources may differ among different resource types. Specifically, for each resource type i, the value of $P^{(i)}$ is randomly selected from $\{32, 64, 128, 256, 512, 1024\}$.

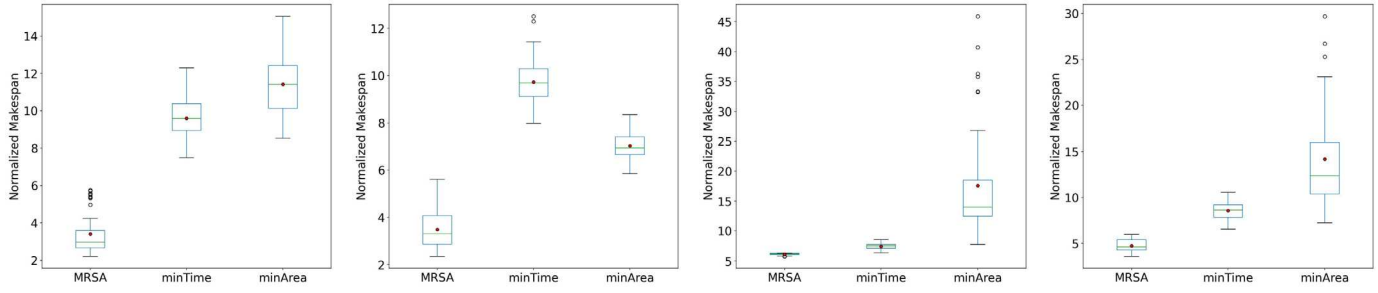
Other parameters (e.g., for generating graphs and job execution times) are set as their default values/ranges as described in Section 7.1. In the simulations, we randomly generate 50 workflows and obtain, for each workflow, the makespans of the three algorithms. We then normalize the obtained makespans by the lower bound (as shown in Lemma 1) for that workflow and report the statistics on the normalized makespans across the 50 workflows.

The boxplots in Fig. 4 show the simulation results for the three algorithms in the uniform P scenario under the four speedup models. We can see that MRSA outperforms the other two heuristics significantly in all cases. In terms of the makespan distribution, even MRSA's worst makespan for the 50 workflows is better than the best makespan obtained by the other two heuristics. When comparing the median/mean makespan across the 50 workflows, MRSA is almost six times faster than at least one of the two heuristics.



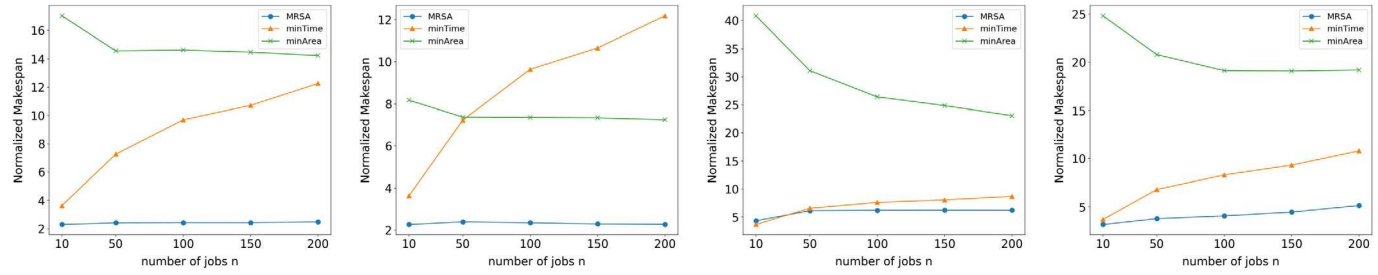
(a) Amdahl-Sum (b) Amdahl-Max (c) Power-Sum (d) Power-Max

Fig. 4. Boxplots showing the normalized makespans of the three algorithms for 50 workflows with $n = 100$ jobs, $d = 3$ resource types, and uniform P ($= 256$) under the four speedup models.



(a) Amdahl-Sum (b) Amdahl-Max (c) Power-Sum (d) Power-Max

Fig. 5. Boxplots showing the normalized makespans of the three algorithms for 50 workflows with $n = 100$ jobs, $d = 3$ resource types, and non-uniform P (up to 1024) under the four speedup models.



(a) Amdahl-Sum (b) Amdahl-Max (c) Power-Sum (d) Power-Max

Fig. 6. Impact of the number of jobs n on the performance of the three algorithms under the four speedup models.

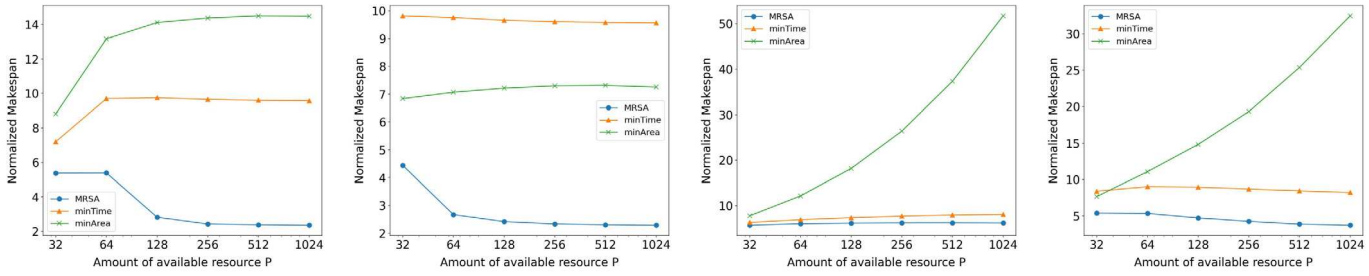
Fig. 5 shows the corresponding results in the non-uniform P scenario. The results are quite similar to those in the uniform scenario, and MRSA again significantly outperforms the other two heuristics in terms of the makespan distribution, as well as the mean and median values. In contrast to the uniform P scenario, MRSA's makespans now exhibit a slightly more skewed distribution and a larger range of variation, due to the non-uniformity in the amount of available resources of different types. But in both scenarios, we can observe that MRSA's performance is much better than the theoretical analysis predicts, which under this setting has an approximation ratio of 10.26 according to Theorem 1, while the normalized makespan of MRSA is at most 7 in our simulation.

Due to the similarity of results in both scenarios, we will use the uniform scenario for all subsequent experiments, which evaluate the impacts of different parameters from the default setting considered in this section.

7.3. Impact of system parameters

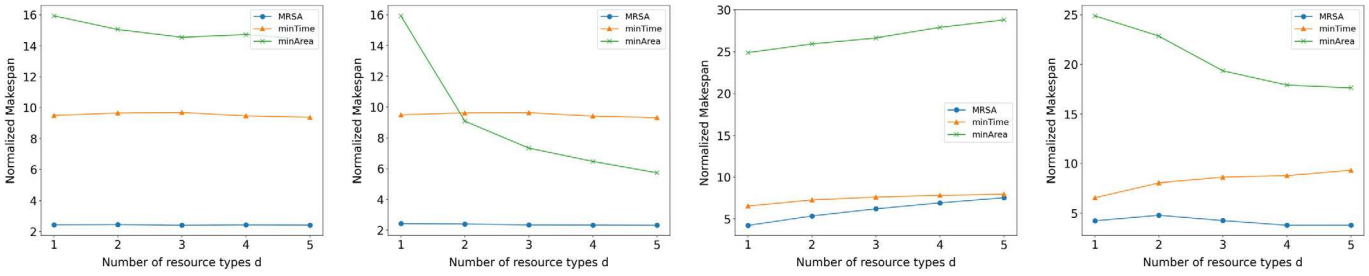
This section presents the simulation results that focus on evaluating the impacts of different system parameters on the performance of the algorithms. In particular, we consider three parameters: the number of jobs in a workflow (n), the amount of available resources (P), and the number of resource types (d). In the experiments, only the evaluated parameter is varied and all other parameters are set at their default values as in Section 7.2. We again generate 50 workflows and compute the normalized makespans of the three algorithms for each workflow. The results are reported by averaging the normalized makespans across the 50 workflows for each algorithm.

Impact of number of jobs n. Fig. 6 shows the results when the number of jobs n is varied between 10 and 200. It is evident that MRSA consistently performs well in all cases. As n increases, its normalized makespan stays almost constant for the two Amdahl models and only increases slightly for the two power models. In contrast, we can observe a steady increase in the normalized makespan of minTime and a significant drop for minArea. This is because as the jobs do not have perfectly linear speedup, minTime becomes less efficient by allocating all the resources to each job. On the other hand, minArea allocates a small amount of resources to each job, which enables more jobs to be executed concurrently and more efficiently when there are more jobs in a workflow.



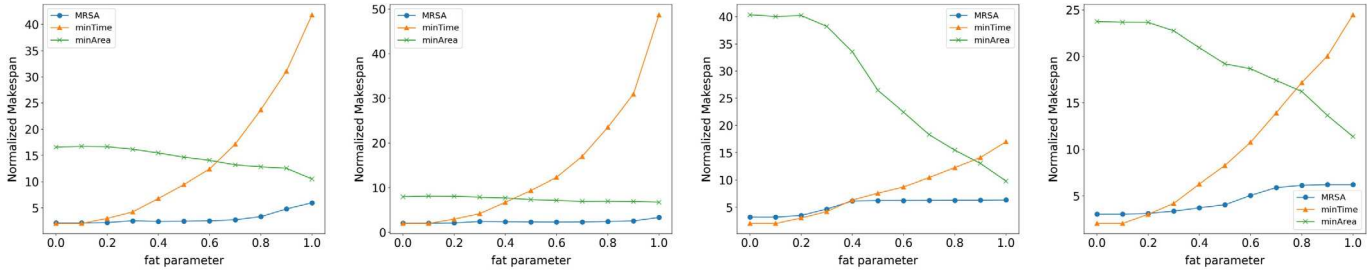
(a) Amdahl-Sum (b) Amdahl-Max (c) Power-Sum (d) Power-Max

Fig. 7. Impact of the amount of available resources P on the performance of the three algorithms under the four speedup models.



(a) Amdahl-Sum (b) Amdahl-Max (c) Power-Sum (d) Power-Max

Fig. 8. Impact of the number of resource types d on the performance of the three algorithms under the four speedup models.



(a) Amdahl-Sum (b) Amdahl-Max (c) Power-Sum (d) Power-Max

Fig. 9. Impact of the fat parameter in DAGGEN on the performance of the three algorithms under the four speedup models.

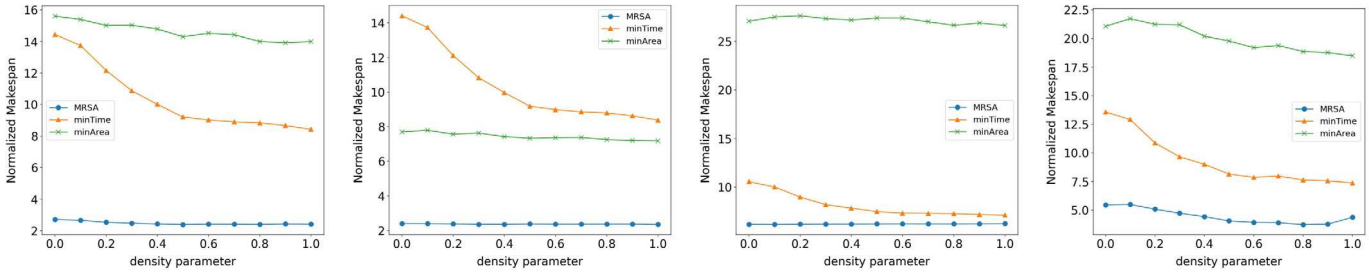
Impact of amount of available resources P. Fig. 7 shows the impact of the amount of available resources P (uniform across all resource types) when it is varied as a power-of-2 between 32 and 1024. We can see that the normalized makespan of MRSA decreases as the amount of resources increases, demonstrating its ability to leverage the availability of more system resources to improve performance. The only exception is for the Power-Sum model, where the normalized makespan of MRSA appears unaffected by P. For minTime, the performance trend is similar to that of MRSA but it remains worse than MRSA by a significant margin. For minArea, we see a general increase in normalized makespan as P increases. This is because not all resources will be utilized in this case due to minArea's conservative resource allocation strategy.

Impact of number of resource types d. Fig. 8 shows the performance of the algorithms when the number of resource types d is varied from 1 to 5. Although the approximation ratio of MRSA as suggested by Theorem 1 grows linearly with d, its practical performance as shown in the figure appears not much affected by the number of resource types, except for the Power-Sum model, where the normalized makespan gradually increases with d. The performance trend for minTime is similar to that for MRSA, and the impact on minArea varies for different speedup models, but MRSA remains the best performer in all cases.

7.4. Impact of graph structure

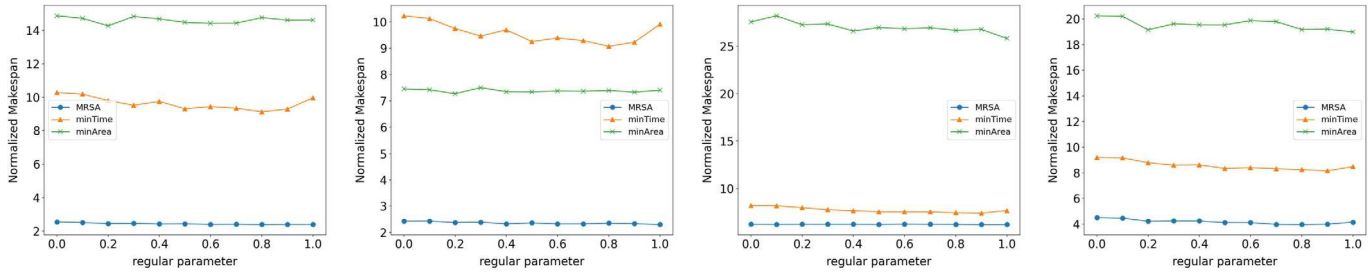
This section evaluates the impact of workflow/graph structure on the performance of the scheduling algorithms. As described in Section 7.1, four parameters (i.e., fat, density, regular and jump) affect the structure of the graphs generated by DAGGEN. To evaluate their impacts, we vary fat, density, and regular from 0 to 1 at an increment of 0.1, and vary jump from 1 to 5 at an increment of 1. Again, all other parameters are set at their default values, and the average normalized makespans across 50 workflows are reported for all algorithms.

Impact of fat parameter. The fat parameter controls the width of a graph. Fig. 9 shows that, as the graph becomes wider, the normalized makespan of MRSA experiences only a mild increase, whereas minTime has a sharper increase in normalized makespan. While minTime allocates all the resources to each job and thus executes them sequentially, given a fixed number of jobs, its makespan is likely not affected but the makespan lower bound will decrease due to increased graph width (hence decreased graph depth and critical-path length). This results in an increase in minTime's normalized makespan. On the other hand, we see a decrease in the normalized makespan for minArea. This is because a larger graph width allows more jobs to be executed concurrently



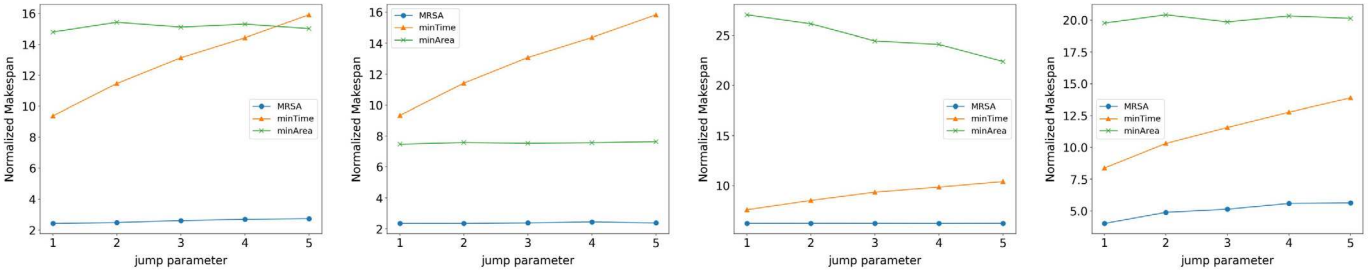
(a) Amdahl-Sum (b) Amdahl-Max (c) Power-Sum (d) Power-Max

Fig. 10. Impact of the density parameter in DAGGEN on the performance of the three algorithms under the four speedup models.



(a) Amdahl-Sum (b) Amdahl-Max (c) Power-Sum (d) Power-Max

Fig. 11. Impact of the regular parameter in DAGGEN on the performance of the three algorithms under the four speedup models.



(a) Amdahl-Sum (b) Amdahl-Max (c) Power-Sum (d) Power-Max

Fig. 12. Impact of the jump parameter in DAGGEN on the performance of the three algorithms under the four speedup models.

by minArea, which only allocates a small amount of resources to each job.

Impact of density parameter. The density parameter controls the number of dependencies between jobs of two consecutive layers of a graph. Fig. 10 shows that density barely affects the performance of MRSA, and as it increases, the normalized makespans of both minTime and minArea tend to decrease. This is probably due to the increase in the critical path and hence the makespan lower bound that has resulted from increased connectivity between layers of the graph.

Impact of regular parameter. The regular parameter controls the distribution of jobs between different layers of a graph. Fig. 11 shows that this parameter has little impact on the performance of all three algorithms.

Impact of jump parameter. The jump parameter controls the maximum number of layers that can be spanned by the edges of a graph. Fig. 12 shows that MRSA is again not affected by this parameter, except for the Power-Max model where its normalized makespan has a slight increase. The performance of minArea is also not affected much by jump, except for the Power-Sum model where its normalized makespan decreases. A larger jump potentially reduces the critical-path length of a graph and hence the makespan lower bound. This causes a uniform increase in the normalized makespan for minTime, as can be seen in the figure.

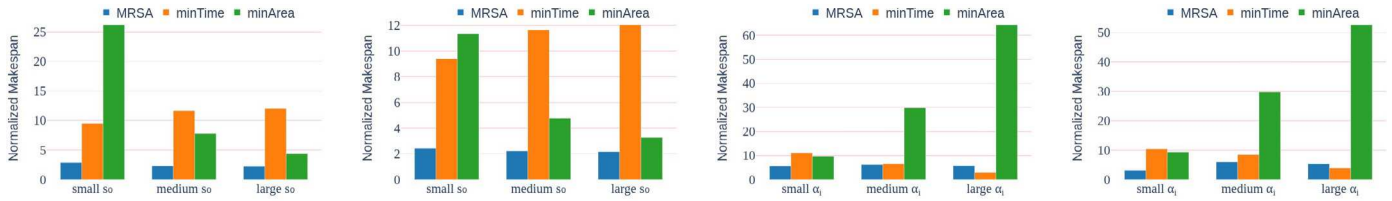
7.5. Impact of job speedup functions

In this section, we evaluate the impact of the jobs' speedup functions on the performance of the algorithms. In particular, we focus on two parameters, namely, the sequential fraction s_0 (in the Amdahl models) and the efficiency factor α_i (in the power models). Both parameters control the degree of parallelism for a job: while a larger s_0 makes the job less parallelizable, a larger α_i makes the job more parallelizable. In this set of simulations, we choose three different ranges (small, medium and large) for setting these two parameters.

For the sequential fraction s_0 in the two Amdahl models, its ranges are set as follows.

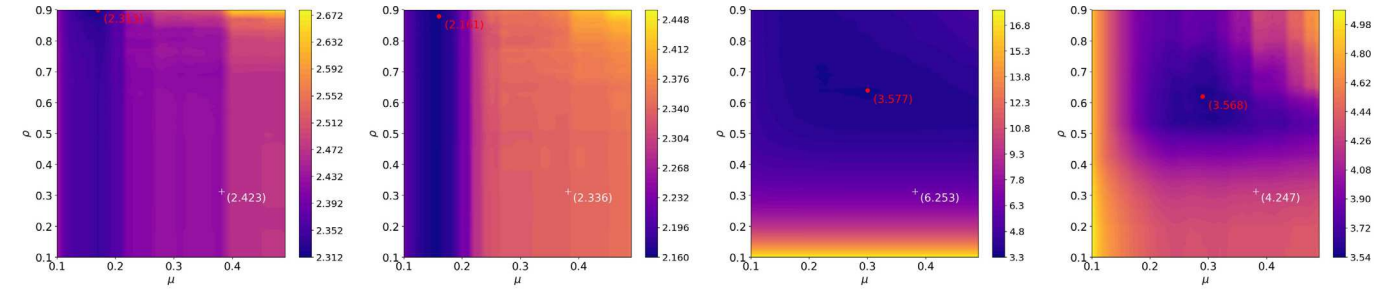
- *small*: s_0 is uniformly generated in $(0, 0.1]$;
- *medium*: s_0 is uniformly generated in $(0.2, 0.3]$;
- *large*: s_0 is uniformly generated in $(0.4, 0.5]$.

For the efficiency factor α_i in the two power models, its ranges are set as follows.



(a) Amdahl-Sum (b) Amdahl-Max (c) Power-Sum (d) Power-Max

Fig. 13. Impact of the sequential fraction s_0 (in the two Amdahl models) and the efficiency factor α_i (in the two power models) on the performance of the three algorithms.



(a) Amdahl-Sum (b) Amdahl-Max (c) Power-Sum (d) Power-Max

Fig. 14. Impact of parameters μ and ρ on the performance of MRSA under the four speedup models. The red dot (•) indicates the combination of μ and ρ that gives the best simulated average normalized makespan (in parentheses), while the white plus sign (+) indicates the combination as suggested by Theorem 1 (with simulated average normalized makespan in parentheses).

- *small*: α_i is uniformly generated in $(0.2, 0.4]$;
- *medium*: α_i is uniformly generated in $(0.5, 0.7]$;
- *large*: α_i is uniformly generated in $(0.8, 1]$.

Fig. 13 shows that MRSA is not much affected by different ranges of these two parameters in all speedup models and it consistently performs well, illustrating its ability to adapt to variations in the speedup functions of the jobs. For the two Amdahl models (as shown in Fig. 13(a,b)), we can see an increasing trend in the normalized makespan of minTime as s_0 increases and a decreasing trend in the normalized makespan of minArea. This is because when the jobs become less parallelizable (with an increased sequential fraction s_0), the minTime algorithm that allocates all resources to a job becomes less efficient, and it calls for a more conservative resource allocation strategy, which is what minArea does. The same can be observed and explained for the two power models (as shown in Fig. 13(c,d)). In particular, as α_i increases, which makes the jobs more parallelizable, minTime becomes more efficient in resource allocation, thus its normalized makespan decreases. On the other hand, the normalized makespan of minArea increases due to its inability to adapt to changes in job characteristics and to utilize all the available resources in the system. Note that when α_i is large (i.e., close to 1), the jobs become almost fully parallelizable, thus allocating all resources to a job (as is done by minTime) is close to being optimal, which is why minTime fares even better than MRSA in this case.

7.6. Impact of MRSA parameters

Two parameters μ and ρ are used in MRSA and they are involved in the derivation of the algorithm's approximation ratio. According to Theorem 1, their values are optimized at $\mu^* \approx 0.382$ and $\rho^* \approx 0.312$ when there are three types of resources (i.e., $d = 3$). In this experiment, we aim to evaluate the impact of these two parameters on the practical performance of the algorithm.

Fig. 14 shows the normalized makespan of MRSA (averaged over 50 workflows) under the four speedup models when μ is varied from 0.1 to 0.5 and ρ is varied from 0.1 to 0.9, both with an increment of 0.01 each time. The red dot (•) in each plot indicates the combination of μ and ρ that gives the best average normalized makespan (in parentheses) under the respective speedup model in our simulation, while the white plus sign (+) indicates the combination as suggested by Theorem 1 with average normalized makespan obtained in simulation in parentheses. We can see that the simulation results are not exactly in line with the theoretical analysis. For the two Amdahl models, the best μ is between 0.15 and 0.2, and the best ρ is close to 0.9. However, the difference in average normalized makespan is relatively small (i.e., within 0.2 in both cases). For the two power models, the best μ is around 0.3, and the best ρ is between 0.6 and 0.7. In these two cases, there is a larger difference in average normalized makespan (i.e., 2.67 in Power-Sum and 0.68 in Power-Max). Such a discrepancy is possibly due to the following reasons: (1) the approximation ratio derived in Theorem 1 is not tight, hence the suggested μ^* and ρ^* are not optimal; (2) the theoretical analysis assumes the worst-case scenario, thus may not reflect the practical performance as reported by the simulation[10]; and (3) the analysis is based on a generic job execution model and does not consider specific speedup functions. The insights gained in this experiment can be potentially explored in future work to improve the approximation results of the algorithm (e.g., by further tuning the choices of parameters μ and ρ, and by taking into account specific speedup models of the jobs).

7.7. Results for a single resource type

In this section, we evaluate the performance of MRSA when there is only a single type of resource, i.e., $d = 1$. For this special case, the two parameters of MRSA are set as $\mu^* \approx 0.382$ and $\rho^* \approx 0.44$ according to Theorem 1, and it has an approximation ratio of 5.164.

We note that MRSA is not specifically designed and optimized for a single resource type, and there exist better approximation algorithms

[10] Indeed, the worst-case scenario considers that, at any time, only one of the d resource types is used, resulting in a very low efficiency in term of the area. This leads to a conservative decision with a small ρ that trades low areas with high critical-path length. In the practical simulations, however, several resources are used most of the time, which is why a larger ρ gives better results.

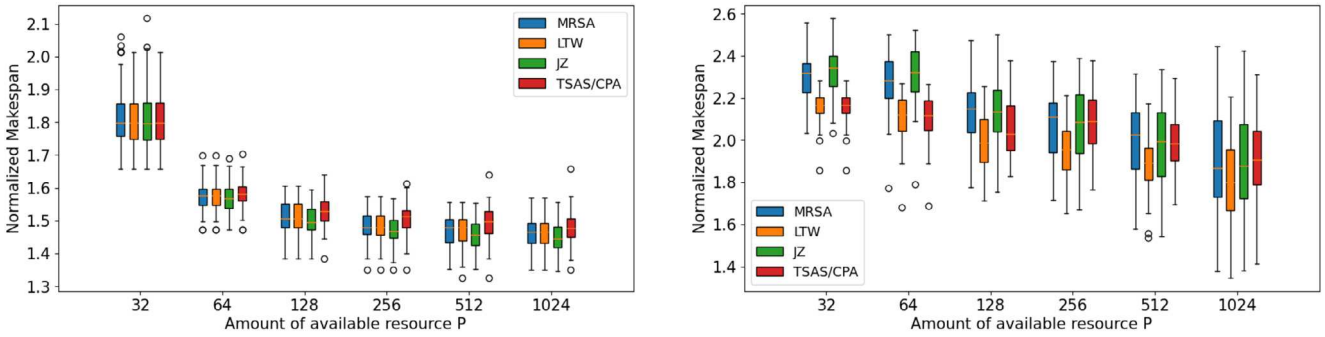


Fig. 15. Performance comparison of the four approximation algorithms for the single resource case ($d = 1$) under the two speedup models.

for this case. This section provides a performance comparison of MRSA with a few other approximation algorithms that similarly use the two-phase scheduling framework for this scheduling problem. In particular, the compared algorithms all use list scheduling in the second phase to schedule the jobs but they deploy different strategies to allocate resources in the first phase by strategically balancing the area and critical-path length of the graph. The following describes the compared algorithms and their resource allocation strategies.

- *LTW*: This algorithm was proposed by Lepère, Trystram, and Woeginger [42]. It works similarly as MRSA by setting $\mu \approx 0.382$ but $\rho = 0.5$. It was shown to have an approximation ratio of 5.236.
- *JZ*: This algorithm was proposed by Jansen and Zhang [36]. It also follows the same general strategy as MRSA and LTW, but sets $\mu \approx 0.2709$ and $\rho \approx 0.431$, achieving a better approximation ratio of 4.73.
- *TSAS/CPA*: TSAS was proposed in [55], and it finds a resource allocation using convex programming that balances the area and critical-path length (i.e., minimizes the maximum of the two), while setting $\mu \approx 0.58579$. It was shown to have an approximation ratio of 11.66. CPA [54] works similarly as TSAS but finds an allocation using a greedy heuristic that has much lower computational complexity.

The simulation of this section follows the same setup as in the previous sections (but with $d = 1$), while varying the total amount P of the only resource type between 32 and 1024. Thanks to the smaller search space with $d = 1$, we consider all possible allocations (instead of only power-of-2 allocations as done in previous sections), thus offering a more precise evaluation of these algorithms.

Fig. 15 shows the comparison results for the four algorithms under the Amdahl and power models. Note that, when $d = 1$, the "Sum" and "Max" variations collapse into the same model. We can see that the four algorithms have very similar performance under the Amdahl model for all values of P. For the power model, LTW and TSAS/CPA perform slightly better than the other two algorithms for small P, but all of them again have similar performance when P become larger. Under both models, the normalized makespans of all algorithms never exceed 2.6 and the average values decrease as P increases, showing these algorithms' ability to effectively utilize the available resources to achieve good performance. Overall, the results suggest that MRSA along with the compared approximation algorithms perform comparably for scheduling moldable workflows under a single resource type.

8. Conclusion and future work

In this paper, we have studied the problem of scheduling moldable workflows with multiple types of resources with the objective of minimizing the makespan. We have proposed a multi-resource scheduling algorithm, MRSA, that adopts a two-phase approach by combining an approximate resource allocation and an extended list scheduling scheme. Theoretically, we have proven approximation ratios of the algorithm for general workflows, as well as improved ratios for some special graphs including SP graphs or trees and independent jobs. Table 1 summarizes all the approximation results. We have also proven a lower bound on the approximation ratio of any local list-based scheduling scheme. Empirically, we have evaluated the performance of MRSA by conducting simulations using synthetic workflows generated by DAGGEN and moldable jobs following different speedup models. The results show that the practical performance of MRSA is better than the approximation ratio predicts, and that it outperforms two other heuristic algorithms under a variety of parameter settings, including different system parameters, graph structures, job speedup models, etc. It also performs comparably with a few other approximation algorithms designed for the special case of a single resource type.

Table 1
Summary of approximation results.

Precedence	Approximation Ratio
General Graphs	• $1.619d + 2.545\sqrt{d} + 1$ for $d \geq 1$ • $d + 3\sqrt[3]{d^2} + O(\sqrt[3]{d})$ for $d \geq 22$
SP Graphs or Trees	• $(1+\epsilon)(1.619d + 1)$ for $d \geq 1$ • $(1+\epsilon)\left(d + 2\sqrt{d-1}\right)$ for $d \geq 4$
Independent Jobs	• $2d$ for $d \geq 1$ [61] • $1.619d + 1$ for $d = 3$ • $d + 2\sqrt{d-1}$ for $d \geq 4$

The future work for this research could include both theoretical and empirical extensions. On the theoretical front, we will seek to find better scheduling algorithms by proving improved approximation ratios, while possibly taking specific job speedup functions into account. We point out that the lower bound shown in Theorem 6 does not rule out the possibility of a *global* list scheduling algorithm that considers the structure of the precedence graph when determining the priorities for the jobs (e.g., by prioritizing the jobs on the critical path). On the practical side, while our evaluations in this paper are based on simulations using synthetic jobs, it will be insightful to validate the performance of our algorithm by evaluating it using real-world moldable workflows on real systems that have multi-resource scheduling capabilities.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We have shared in the paper the link to code which is publicly available

Acknowledgment

This research is supported in part by the US National Science Foundation (NSF) grant 2135310.

References

[1] G.M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, in: AFIPS'67, 1967, pp. 483–485.
[2] C. Augonnet, S. Thibault, R. Namyst, P.-A. Wacrenier StarPU, A unified platform for task scheduling on heterogeneous multicore architectures, Concurr. Comput., Pract. Exper. 23 (2) (2011) 187–198.
[3] S. Bansal, P. Kumar, K. Singh, An improved two-step algorithm for task and data parallel scheduling in distributed memory machines, Parallel Comput. 32 (10) (2006) 759–774.
[4] O. Beaumont, L.-C. Canon, L. Eyraud-Dubois, G. Lucarelli, L. Marchal, C. Mommessin, B. Simon, D. Trystram, Scheduling on two types of resources: a survey, ACM Comput. Surv. 53 (3) (2020).
[5] O. Beaumont, L. Eyraud-Dubois, S. Kumar, Fast approximation algorithms for task-based runtime systems, Concurr. Comput., Pract. Exper. 30 (17) (2018) e4502.
[6] H.L. Bodlaender, B. de Fluiter, Parallel algorithms for series parallel graphs, in: ESA, 1996, pp. 277–289.
[7] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, J.J. Dongarra PaRSEC, Exploiting heterogeneity to enhance scalability, Comput. Sci. Eng. 15 (6) (2013) 36–45.
[8] M. Caccamo, R. Pellizzoni, L. Sha, G. Yao, H. Yun Memguard, Memory bandwidth reservation system for efficient performance isolation in multi-core platforms, in: RTAS, 2013, pp. 55–64.
[9] C. Chen, An improved approximation for scheduling malleable tasks with precedence constraints via iterative method, IEEE Trans. Parallel Distrib. Syst. 29 (9) (2018) 1937–1946.
[10] C.-Y. Chen, C.-P. Chu, A 3.42-approximation algorithm for scheduling malleable tasks under precedence constraints, IEEE Trans. Parallel Distrib. Syst. 24 (8) (2013) 1479–1488.
[11] W. Chen, Y. Xu, X. Wu, Deep reinforcement learning for multi-resource multi-machine job scheduling, CoRR, arXiv:1711.07440 [abs], 2017.
[12] P. De, E.J. Dunne, J.B. Ghosh, C.E. Wells, Complexity of the discrete time-cost tradeoff problem for project networks, Oper. Res. 45 (2) (1997) 302–306.
[13] P. De, E. James Dunne, J.B. Ghosh, C.E. Wells, The discrete time-cost tradeoff problem revisited, Eur. J. Oper. Res. 81 (2) (1995) 225–238.
[14] G. Demirci, H. Hoffmann, D.H.K. Kim, Approximation algorithms for scheduling with resource and precedence constraints, in: STACS, 2018.
[15] G. Demirci, I. Marincic, H. Hoffmann, A divide and conquer algorithm for dag scheduling under power constraints, in: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, 2018, pp. 466–477.
[16] J. Du, J.Y.-T. Leung, Complexity of scheduling parallel task systems, SIAM J. Discrete Math. 2 (4) (1989) 473–487.
[17] L. Eyraud-Dubois, S. Kumar, Analysis of a list scheduling algorithm for task graphs on two types of resources, in: IPDPS, 2020.
[18] Y. Fan, Z. Lan, Exploiting multi-resource scheduling for HPC, in: SC Poster, 2019.
[19] Y. Fan, Z. Lan, P. Rich, W.E. Allcock, M.E. Papka, B. Austin, D. Paul, Scheduling beyond CPUs for HPC, in: HPDC, 2019.
[20] Y. Fan, P. Rich, W. Allcock, M. Papka, Z. Lan. Rome, A multiresource job scheduling framework for exascale HPC system, in: IPDPS Poster, 2018.
[21] D.G. Feitelson, Job scheduling in multiprogrammed parallel systems (extended version), IBM Res. Rep. RC19790 (87657) (1997).
[22] A. Feldmann, M.-Y. Kao, J. Sgall, S.-H. Teng, Optimal on-line scheduling of parallel jobs with dependencies, J. Comb. Optim. 1 (4) (1998) 393–411.
[23] M.R. Garey, R.L. Graham, Bounds for multiprocessor scheduling with resource constraints, SIAM J. Comput. 4 (2) (1975) 187–200.
[24] T. Gautier, X. Besseron, L. Pigeon KAAPI, A thread scheduling runtime system for data flow computations on cluster of multi-processors, in: PASCO, 2007, pp. 15–23.
[25] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica, Dominant resource fairness: fair allocation of multiple resource types, in: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, 2011, pp. 323–336.
[26] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, A. Akella, Multi-resource packing for cluster schedulers, SIGCOMM Comput. Commun. Rev. 44 (4) (Aug. 2014) 455–466.
[27] H. Hamzeh, S. Meacham, K. Khan, K. Phalp, A. Stefanidis Mrfs, A multi-resource fair scheduling algorithm in heterogeneous cloud computing, in: COMPSAC, 2020.
[28] Y. He, J. Liu, H. Sun, Scheduling functionally heterogeneous systems with utilization balancing, in: IPDPS, 2011, pp. 1187–1198.
[29] Y. He, H. Sun, W.-J. Hsu, Adaptive scheduling of parallel jobs on functionally heterogeneous resources, in: ICPP, 2007, p. 43.
[30] Y. Hu, C. de Laat, Z. Zhao, Learning workflow scheduling on multi-resource clusters, in: 2019 IEEE International Conference on Networking, Architecture and Storage (NAS), 2019.
[31] K.-C. Huang, W.-Y. Wu, F.-J. Wang, H.-C. Liu, C.-H. Hung, An iterative expanding and shrinking process for processor allocation in mixed-parallel workflow scheduling, SpringerPlus 5 (1138) (2016).
[32] S. Hunold, Low-cost tuning of two-step algorithms for scheduling mixed-parallel applications onto homogeneous clusters, in: CCGrid, 2010.
[33] S. Hunold, Scheduling moldable tasks with precedence constraints and arbitrary speedup functions on multiprocessors, in: R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Waśniewski (Eds.), Parallel Processing and Applied Mathematics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 13–25.
[34] K. Jansen, F. Land, Scheduling monotone moldable jobs in linear time, in: IPDPS, 2018, pp. 172–181.
[35] K. Jansen, H. Zhang, Scheduling malleable tasks with precedence constraints, in: SPAA, 2005, pp. 86–95.
[36] K. Jansen, H. Zhang, An approximation algorithm for scheduling malleable tasks under general precedence constraints, ACM Trans. Algorithms 2 (3) (2006) 416–434.
[37] S. Jiang, J. Wu, Multi-resource allocation in cloud data centers: a trade-off on fairness and efficiency, Concurr. Comput., Pract. Exper. 33 (6) (2021) e6061.
[38] C. Joe-Wong, S. Sen, T. Lan, M. Chiang, Multi-resource allocation: fairness-efficiency tradeoffs in a unifying framework, in: IEEE INFOCOM, 2012.
[39] J. Khamse-Ashari, I. Lambadaris, G. Kesidis, B. Urgaonkar, Y. Zhao, Per-server dominant-share fairness (PS-DSF): a multi-resource fair allocation mechanism for heterogeneous servers, in: ICC, 2017.
[40] D. Klusáček, H. Rudová, Multi-resource aware fairsharing for heterogeneous systems, in: W. Cirne, N. Desai (Eds.), JSSPP, 2015.
[41] R. Lepère, G. Mounié, D. Trystram, An approximation algorithm for scheduling trees of malleable tasks, Eur. J. Oper. Res. 142 (2) (2002) 242–249.
[42] R. Lepère, D. Trystram, G.J. Woeginger, Approximation algorithms for scheduling malleable tasks under precedence constraints, Int. J. Found. Comput. Sci. 13 (4) (2002) 613–627.
[43] B. Li, Y. Fan, M. Dearing, Z. Lan, P. Rich, W. Allcock, M. Papka Mrsch, Multi-resource scheduling for HPC, in: IEEE CLUSTER, 2022, pp. 47–57.
[44] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, C. Maltzahn, On the role of burst buffers in leadership-class storage systems, in: MSST, 2012, pp. 1–11.
[45] W. Ludwig, P. Tiwari, Scheduling malleable and nonmalleable parallel tasks, in: SODA, 1994, pp. 167–176.
[46] J. Mohan, A. Phanishayee, J. Kulkarni, V. Chidambaram, Looking beyond GPUs for DNN scheduling on multi-tenant clusters, in: USENIX OSDI, 2022.
[47] G. Mounié, C. Rapine, D. Trystram, A 3/2-approximation algorithm for scheduling independent monotonic malleable tasks, SIAM J. Comput. 37 (2) (2007) 401–412.
[48] M. Niemeier, A. Wiese, Scheduling with an orthogonal resource constraint, in: WAOA, 2012, pp. 242–256.
[49] M. NoroozOliaee, B. Hamdaoui, M. Guizani, M.B. Ghorbel, Online multi-resource scheduling for minimum task completion time in cloud servers, in: INFOCOM Workshops, 2014.
[50] L. Perotin, H. Sun, P. Raghavan, Multi-resource list scheduling of moldable parallel jobs under precedence constraints, in: ICPP, 2021.
[51] G.N.S. Prasanna, B.R. Musicus, Generalized multiprocessor scheduling and applications to matrix computations, IEEE Trans. Parallel Distrib. Syst. 7 (6) (1996) 650–664.
[52] K. Psychas, J. Ghaderi, Randomized algorithms for scheduling multi-resource jobs in the cloud, IEEE/ACM Trans. Netw. 26 (5) (2018) 2202–2215.
[53] A. Radulescu, C. Nicolescu, A. van Gemund, P. Jonker, Cpr: mixed task and data parallel scheduling for distributed systems, in: IPDPS, 2001.
[54] A. Radulescu, A. van Gemund, A low-cost approach towards mixed task and data parallel scheduling, in: ICPP, 2001.
[55] S. Ramaswamy, S. Sapatnekar, P. Banerjee, A framework for exploiting task and data parallelism on distributed memory multicomputers, IEEE Trans. Parallel Distrib. Syst. 8 (11) (1997) 1098–1116.
[56] S. Ristov, R. Prodan, M. Gusev, K. Skala, Superlinear speedup in HPC systems: why and when?, in: Federated Conference on Computer Science and Information Systems (FedCSIS), 2016, pp. 889–898.
[57] M. Sheikhalishahi, R.M. Wallace, L. Grandinetti, J.L. Vazquez-Poletti, F. Guerriero, A multi-dimensional job scheduling, Future Gener. Comput. Syst. 54 (2016) 123–131.
[58] D.B. Shmoys, C. Stein, J. Wein, Improved approximation algorithms for shop scheduling problems 23 (3) (1994) 617–632.
[59] M. Skutella, Approximation algorithms for the discrete time-cost tradeoff problem, Math. Oper. Res. 23 (4) (1998) 909–929.
[60] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, Y.-C. Liu, Knights Landing: second-generation Intel Xeon Phi product, IEEE MICRO 36 (2) (2016) 34–46.
[61] H. Sun, R. Elghazi, A. Gainaru, G. Aupy, P. Raghavan, Scheduling parallel tasks under multiple resources: list scheduling vs. pack scheduling, in: IPDPS, 2018, pp. 194–203.
[62] F. Suter DAGGEN, A synthetic task graph generator, https://github.com/frs69wq/daggen.
[63] J. Turek, J.L. Wolf, P.S. Yu, Approximate algorithms scheduling parallelizable tasks, in: SPAA, 1992.
[64] H. Wang, Z. Liu, H. Shen, Job scheduling for large-scale machine learning clusters, in: CoNEXT, 2020.

[65] Q. Wang, K.H. Cheng, A heuristic of scheduling parallel tasks and its analysis, SIAM J. Comput. 21 (2) (1992) 281–294.
[66] W. Wang, B. Li, B. Liang, J. Li, Towards multi-resource fair allocation with placement constraints, in: ACM SIGMETRICS, 2016, pp. 415–416.
[67] W. Wang, B. Liang, B. Li, Multi-resource fair allocation in heterogeneous cloud computing systems, IEEE Trans. Parallel Distrib. Syst. 26 (10) (2015) 2822–2835.
[68] H. Xu, Y. Liu, W.C. Lau, Multi resource scheduling with task cloning in heterogeneous clusters, in: ICPP, 2022.
[69] M. Xu, L.T.X. Phan, X. Phan, H. Choi, I. Lee vCAT, Dynamic cache management using CAT virtualization, in: RTAS, 2017.
[70] Y. Zhao, Y. Liu, Y. Peng, Y. Zhu, X. Liu, X. Jin, Multi-resource interleaving for deep learning training, in: ACM SIGCOMM, 2022, pp. 428–440.
[71] H. Zhou, Q. Li, W. Tong, S. Kausar, H. Zhu, P-aware: a proportional multi-resource scheduling strategy in cloud data center, Clust. Comput. 19 (2016) 1089–1103.

Lucas Perotin is a PhD student in the Parallel Computing Laboratory (LIP) at Ecole Normale Supérieure de Lyon (ENS Lyon), France. He graduated with a master's degree in Computer Science also from ENS Lyon in 2020. He is mainly interested in scheduling and resilience algorithms for high-performance computing.

Sandhya Kandaswamy is a master's student in the Department of Electrical Engineering and Computer Science (EECS) at the University of Kansas, USA. Her research interests are in data science, machine learning, and high-performance computing.

Hongyang Sun is an Assistant Professor in the Department of Electrical Engineering and Computer Science of the University of Kansas, USA. He obtained his Ph.D. in Computer Science from Nanyang Technological University in Singapore, has worked as a Postdoctoral Researcher at ENS Lyon, INRIA (Rhône-Alpes), and IRIT (Toulouse) in France, and held a research faculty position at Vanderbilt University, USA. His research interests include scheduling and resilience algorithm as well as machine learning techniques to solve important problems in high-performance computing and cloud/edge computing. See https://www.ittc.ku.edu/~sun/ for further information.

Padma Raghavan is Vanderbilt's inaugural Vice Provost for Research and a Professor of Computer Science and Computer Engineering. She joined Vanderbilt in February 2016 from Penn State, where she was the founding Director of the university's Institute for CyberScience. She also served as the Associate Vice President for Research and Strategic Initiatives and as a Distinguished Professor of Computer Science and Engineering at Penn State. She specializes in computational data science and high-performance computing. Her research has been recognized by the NSF CAREER Award (1995), the Maria Goeppert-Mayer Distinguished Scholar Award (2002, University of Chicago and the Argonne National Laboratory), and selection as a Fellow of the Institute of Electrical and Electronic Engineers (IEEE, 2013). See https://engineering.vanderbilt.edu/bio/padma-raghavan/ for further information.