Multi-Resource List Scheduling of Moldable Parallel Jobs under Precedence Constraints

Lucas Perotin
Laboratoire LIP, ENS Lyon
Lyon, France
lucas.perotin@ens-lyon.fr

Hongyang Sun
Vanderbilt University
Nashville, TN, USA
hongyang.sun@vanderbilt.edu

Padma Raghavan
Vanderbilt University
Nashville, TN, USA
padma.raghavan@vanderbilt.edu

ABSTRACT

The scheduling literature has traditionally focused on a single type of resource (e.g., computing nodes). However, scientific applications in modern High-Performance Computing (HPC) systems process large amounts of data, hence have diverse requirements on different types of resources (e.g., cores, cache, memory, I/O). All of these resources could potentially be exploited by the runtime scheduler to improve the application performance. In this paper, we study multi-resource scheduling to minimize the makespan of computational workflows comprised of parallel jobs subject to precedence constraints. The jobs are assumed to be moldable, allowing the scheduler to flexibly select a variable set of resources before execution. We propose a multi-resource, list-based scheduling algorithm, and prove that, on a system with d types of schedulable resources, our algorithm achieves an approximation ratio of $1.619d + 2.545\sqrt{d} + 1$ for any d, and a ratio of $d + O(\sqrt[3]{d^2})$ for large d. We also present improved results for independent jobs and for jobs with special precedence constraints (e.g., series-parallel graphs and trees). Finally, we prove a lower bound of d on the approximation ratio of any list scheduling scheme with local priority considerations. To the best of our knowledge, these are the first approximation results for moldable workflows with multiple resource requirements.

KEYWORDS

List scheduling, multiple resources, moldable jobs, precedence constraint, makespan, approximation ratio

ACM Reference Format:
Lucas Perotin, Hongyang Sun, and Padma Raghavan. 2021. Multi-Resource List Scheduling of Moldable Parallel Jobs under Precedence Constraints. In *50th International Conference on Parallel Processing (ICPP '21), August 9–12, 2021, Lemont, IL, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3472456.3472487

1 INTRODUCTION

Many complex scientific workflows that are running in today's High-Performance Computing (HPC) systems can be modeled as Directed Acyclic Graphs (DAGs), where the nodes represent the constituent jobs of the workflows and the edges represent the precedence constraints or dependencies among the jobs. While HPC systems often rely on dynamic runtime schedulers, such as KAAPI [14], StarPU [1] or PaRSEC [3], to ensure the efficient execution of these workflows, most existing schedulers focus only on the management of the computational resources (i.e., computing nodes or cores). However, many of today's scientific applications need to process large amounts of data, and thus require not only the computational resources but also strong data management supports. Indeed, modern HPC systems are equipped with more levels of memory/storage (e.g., NVRAMs, SSDs, burst buffers [22]), as well as more advanced architecture and software features (e.g., high-bandwidth memory [30], cache partitioning [34], bandwidth reservation [4]) to facilitate efficient data transfer. All of these different types of resources could potentially be partitioned among the concurrently running jobs and thus exploited by the runtime schedulers to improve the overall application performance and system utilization.

In this paper, we study multi-resource scheduling for a computational workflow that is comprised of a set of parallel jobs with DAG-based precedence 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 the workflow. We focus on parallel jobs that are *moldable* [11], which allows the scheduler to select a variable set of resources for a job, but once the job starts 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. Given these advantages, moldable jobs have been offered by many computational kernels in scientific libraries. Moreover, the moldable job model is also amenable to the resource allocation patterns currently supported by many different resource types (e.g., computing cores, memory blocks, cache lines).

As the considered multi-resource scheduling problem contains the single-resource problem as a special case, it is known to be strongly NP-complete [10]. Thus, we focus on designing good approximation algorithms. In contrast to the single-resource problem, however, the multi-resource problem needs to consider the combined effect of multiple types of resources on the execution time of the jobs, which poses additional challenges to the scheduling problem. By adopting a two-phase approach [32] widely used for scheduling moldable jobs, we design a multi-resource, list-based scheduling algorithm. In particular, our algorithm first computes an approximate resource allocation for all jobs on different resource

ICPP '21, August 9–12, 2021, Lemont, IL, USA

ACM ISBN 978-1-4503-9068-2/21/08…$15.00
https://doi.org/10.1145/3472456.3472487

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.

We prove the following main results for a system consisting of d types of schedulable resources, under reasonable assumptions on the job execution times and speedups:

- An approximation ratio of $1.619d + 2.545\sqrt{d} + 1$ for any d, and a ratio of $d + O(\sqrt[3]{d^2})$ for large d;
- Improved approximations for some special graphs (e.g., series-parallel graphs, trees and independent jobs) with ratios of $1.619d + 1$ for any d and $d + O(\sqrt{d})$ for large d.
- A lower bound of d on the approximation ratio of any list scheduling scheme with local priority considerations.

To the best of our knowledge, these are the first approximation results for moldable workflows with multiple resource requirements. They also improve upon the $2d$-approximation previously shown in [31] for independent moldable jobs. The results demonstrate that our algorithm essentially achieves the optimal asymptotic approximation up to the dominating factor (i.e., d) among the generic class of local list scheduling schemes, thus matching the same asymptotic performance for rigid [13] and malleable [16] jobs. Altogether, these results lay the theoretical foundation for multi-resource scheduling of parallel workflows.

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 derives a lower bound on the optimal makespan. Section 4 presents our multi-resource scheduling algorithm and analyzes its approximation ratios for general job graphs. Section 5 proves improved results for some special graphs, including series-parallel graphs, trees and independent jobs. Section 6 shows a lower bound on the performance of local list scheduling schemes, and finally, Section 7 concludes the paper and briefly discusses open questions.

2 RELATED WORK

This section reviews some related work on scheduling moldable parallel jobs, as well as on multi-resource scheduling under different job models and objectives.

Moldable Job Scheduling. Scheduling moldable parallel jobs to minimize the makespan is strongly NP-hard on $P \geq 5$ processors [10], and the problem has been extensively studied in the literature from the perspective of approximation algorithms. Most prior work, however, has focused on a single type of resource while assuming different speedup models for the jobs.

For scheduling independent moldable jobs with arbitrary speedups, Turek et al. [32] presented a 2-approximation list-based algorithm and a 3-approximation algorithm based on building shelves. Ludwig and Tiwari [23] 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. [24] presented a $(1.5+\epsilon)$-approximation algorithm using dual approximation. Jansen and Land [17] showed a lower complexity 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 with precedence constraints, Lepère et al. [21] presented a 5.236-approximation algorithm for monotonic jobs. Jansen and Zhang [19] improved the approximation ratio to around 4.73 for the same model, and recently, Chen [5] 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 [20, 21]) or special speedup models (e.g., concave speedup [6, 18] and roofline speedup [12, 33]).

Multi-Resource Scheduling. Some approximation algorithms have been proposed on multi-resource scheduling to minimize makespan under different parallel job models.

Garey and Graham [13] 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. [9] presented an improved $O(\log n)$-approximation for jobs with precedence constraints, and Niemeier and Wiese [25] presented an improved $(2 + \epsilon)$-approximation for independent jobs.

He et al. [15, 16] considered parallel jobs that are represented as direct acyclic graphs (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 for this model. Shmoys et al. [28] 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. [31] considered scheduling 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 [32]. 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 [31] for a large number of resource types.

3 MODELS

This section presents the multi-resource scheduling model, gives a formal statement of the problem, and derives a lower bound on the optimal schedule.

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 usage cannot be changed once a job has started executing. For each job j, 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 $p_j^{(i)} \geq 0$ allocated to the job for each resource type $i = 1, 2, \ldots, d$. 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 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 time function of 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 are not concerned about how such a function is 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 [21, 24], 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 $(j_1 \rightarrow j_2) \in E$ 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.

[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* [27] when increasing both processor and cache allocations). We do not consider such superlinear speedup model in this paper.

3.2 Problem Statement

The objective is to find a schedule for the jobs 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 is *valid* if it respects 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., $j_1 \rightarrow j_2$, then the starting time of j_2 is no earlier than the completion time of j_1, i.e., $s_{j_2} \geq c_{j_1}$.

The above multi-resource scheduling problem is clearly NP-complete, as it contains the single-resource scheduling problem [19, 21] as a special case. Thus, we aim at designing approximation algorithms with bounded performance guarantees. An algorithm is said to be *r-approximation* if its makespan satisfies $\frac{T}{T_{\text{OPT}}} \leq r$ for any set of jobs, where T_{OPT} denotes the optimal makespan.

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 given a resource allocation decision $\mathbf{p} = (p_1, p_1, \ldots, p_n)$ for the jobs.

Definition 1. *For each job 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 the set of jobs:*

- $W^{(i)}(\mathbf{p}) = \sum_{j=1}^{n} w_j^{(i)}(p_j)$: *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.

[3]A path is a sequence of jobs with linear precedence, i.e., $f = (j_{\pi(1)} \rightarrow j_{\pi(2)} \rightarrow \cdots \rightarrow j_{\pi(v)})$, 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.

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

Proof. We first show that, given any resource allocation $\mathbf{p}$, the makespan produced by any schedule must satisfy $T \geq \max(A(\mathbf{p}), C(\mathbf{p}))$. 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:

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

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 A MULTI-RESOURCE SCHEDULING ALGORITHM AND APPROXIMATION RESULTS

In this section, we present a multi-resource scheduling algorithm 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 [19, 21, 32].

4.1 Phase 1: Resource Allocation

4.1.1 Discrete Time-Cost Tradeoff (DTCT) Problem. To allocate resources for the jobs, we consider a relevant discrete time-cost tradeoff problem [8], 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 precedence-constrained tasks. 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 $D(\sigma^*)$ and 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 [7]:

- *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 [29] presented a polynomial-time algorithm, which, given any feasible budget-deadline pair (B, D), finds a realization σ for the project that satisfies: $D(\sigma) \leq \frac{D}{\rho}$ and $B(\sigma) \leq \frac{B}{1-\rho}$, for any $\rho \in (0, 1)$.[4]

4.1.2 Allocating Resources to Jobs. We transform our resource allocation problem to the DTCT problem and solve it using the approximation result in [29]. 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:

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

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 total execution time $C(\mathbf{p})$ of the jobs, and the total cost $B(\sigma)$ corresponds to the average total area $A(\mathbf{p})$.

A resource allocation decision $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ 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 $j = 1, \ldots, n$. The following lemma shows that the minimum makespan lower bound $L_{\min}$ can be achieved by a non-dominated resource allocation.

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

Proof. Consider any resource allocation $\mathbf{q}^* = (q_1^*, q_2^*, \ldots, q_n^*)$ that achieves $L(\mathbf{q}^*) = L_{\min}$, and suppose 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., $t_j(q_j'^*) < t_j(q_j^*)$ and $a_j(q_j'^*) < a_j(q_j^*)$, we get a new resource allocation $\mathbf{q}'^* = (q_1^*, \ldots, q_{j-1}^*, q_j'^*, q_{j+1}^*, \ldots, q_n^*)$, which satisfies $A(\mathbf{q}'^*) < A(\mathbf{q}^*)$ 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 $\mathbf{p}^*$ and proves the lemma. □

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.

[4] In essence, this bicriteria 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.

Lemma 3. *For any $\rho \in (0,1)$, a resource allocation $\mathbf{p}' = (p'_1, p'_2, \ldots, p'_n)$ can be found in polynomial time that satisfies:*

$$C(\mathbf{p}') \leq \frac{T_{OPT}}{\rho}\,, \tag{3}$$

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

Proof Sketch. The result can be obtained by adapting the algorithm in [29], 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 this constraint a priori, we can still achieve the same approximations by adopting the technique used in [19] for the problem with a single resource type. Specifically, the relaxed LP originally formulated in [29] 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. □

4.1.3 Adjusting Resource Allocation. Lastly, we adjust the resource allocation $\mathbf{p}'$ (obtained above with a value of ρ to be determined later) 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 (see Section 4.2). As with the case for a single type of resource [19, 21], we choose a parameter $\mu \in (0, 0.5)$, whose value will also be determined later, and define the 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 $\mathbf{p}'$. The $p_j^{(i)}$'s will then form the final resource allocation $\mathbf{p}$.

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

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

$$t_j(p_j) \leq \frac{t_j(p'_j)}{\mu}\,, \tag{6}$$

and its area on any resource type i is bounded by:

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

if the total amount of resource type i satisfies $P^{(i)} \geq \frac{1}{\mu^2}$.

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

Algorithm 1: Resource Allocation (Phase 1)

Input: For each job j, the execution time $t_j(p_j)$ and the average normalized work $a_j(p_j)$ under all possible resource allocations, given values for the parameters ρ and μ.
Output: Resource allocation decision $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ 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 [29] 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

Since the job's final resource allocation p_j is at most its initial allocation p'_j, i.e., $p_j \preceq p'_j$, and according to the adjustment procedure in Equation (5), we have $x_j^{(k)} \leq \frac{P^{(k)}}{\lceil \mu P^{(k)} \rceil} \leq \frac{1}{\mu}$. Thus, based on Assumption 3, we can get $t_j(p_j) \leq \left(\max_{i=1\ldots d} x_j^{(i)}\right) \cdot t_j(p'_j) = x_j^{(k)} \cdot t_j(p'_j) \leq \frac{t_j(p'_j)}{\mu}$.

To prove the area bound, we distinguish three cases.

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

Case (2): For any resource type $i \neq k$ with $p_j^{(i)} \leq \lfloor \mu P^{(i)} \rfloor \leq \mu P^{(i)}$, and since $p_j^{(k)} = \lceil \mu P^{(k)} \rceil \geq \mu P^{(k)}$, we have $a_j^{(i)}(p_j) = \frac{w_j^{(i)}(p_j)}{P^{(i)}} = \frac{p_j^{(i)} \cdot t_j(p_j)}{P^{(i)}} \leq \frac{\mu P^{(i)} \cdot t_j(p_j)}{P^{(i)}} \leq \mu \cdot x_j^{(k)} \cdot t_j(p'_j) = \mu \cdot \frac{p'^{(k)}_j \cdot t_j(p'_j)}{p_j^{(k)}} \leq \mu \cdot \frac{w_j^{(k)}(p'_j)}{\mu P^{(k)}} = \frac{w_j^{(k)}(p'_j)}{P^{(k)}} \leq \sum_{\ell=1}^{d} \frac{w_j^{(\ell)}(p'_j)}{P^{(\ell)}} = d \cdot a_j(p'_j)$.

Case (3): For any resource type $i \neq k$ with $p_j^{(i)} = \lceil \mu P^{(i)} \rceil \leq \mu P^{(i)} + 1$, by following the derivation steps in Case (2), we can get $a_j^{(i)}(p_j) \leq \left(1 + \frac{1}{\mu P^{(i)}}\right) \frac{w_j^{(k)}(p'_j)}{P^{(k)}} \leq \sum_{\ell=1}^{d} \frac{w_j^{(\ell)}(p'_j)}{P^{(\ell)}} + \frac{w_j^{(k)}(p'_j)}{\mu P^{(i)} P^{(k)}} - \frac{w_j^{(i)}(p'_j)}{P^{(i)}} = \sum_{\ell=1}^{d} \frac{w_j^{(\ell)}(p'_j)}{P^{(\ell)}} + \frac{t_j(p'_j)}{P^{(i)}} \left(\frac{p'^{(k)}_j}{\mu P^{(k)}} - p'^{(i)}_j \right)$. Since $p'^{(k)}_j \leq P^{(k)}$ and $p'^{(i)}_j \geq \lceil \mu P^{(i)} \rceil \geq \mu P^{(i)}$, we have $\frac{p'^{(k)}_j}{\mu P^{(k)}} - p'^{(i)}_j \leq \frac{1}{\mu} - \mu P^{(i)}$, which is at most 0 when $P^{(i)} \geq \frac{1}{\mu^2}$. In this case, we get $a_j^{(i)}(p_j) \leq \sum_{\ell=1}^{d} \frac{w_j^{(\ell)}(p'_j)}{P^{(\ell)}} = d \cdot a_j(p'_j)$. □

Algorithm 1 summarizes all three steps involved in this first phase of the multi-resource scheduling algorithm.

4.2 Phase 2: List Scheduling

4.2.1 Algorithm Description. The second phase schedules the jobs by making a starting time decision $\mathbf{s}$, given the resource allocation decision $\mathbf{p}$ determined by the first phase. This is done through a modified list scheduling strategy, as shown in Algorithm 2, that extends to multiple types of resources.

Algorithm 2: List Scheduling (Phase 2)

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

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

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 Q. Then, at time 0 or whenever a running job k completes and hence releases resources, the algorithm inserts, into the queue 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 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.

We point out that the ready jobs can be inserted into the queue in any order without affecting the approximation ratio of the algorithm. In practice, giving priority to certain jobs (e.g., with longer execution time or on the critical path) may yield better performance.

4.2.2 Properties of List Scheduling. We now derive some properties of the list scheduling algorithm, which will be used later in the analysis of the overall multi-resource scheduling algorithm.

We first define some notations. Let T denote the makespan of a list schedule. We note that the algorithm only allocates and deallocates resources upon job completions. Hence, the entire schedule's duration $[0, T]$ can be partitioned into a set $\mathcal{I} = \{I_1, I_2, \ldots\}$ 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 type does not change during an interval. For any resource type i, let $P^{(i)}_{util}(I)$ 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., $\mathcal{I}_1 = \{I \mid \forall i, P^{(i)}_{util}(I) \leq \lceil \mu P^{(i)} \rceil - 1\}$.
- $\mathcal{I}_2$: 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., $\mathcal{I}_2 = \{I \mid \exists k, P^{(k)}_{util}(I) \geq \lceil \mu P^{(k)} \rceil$ and $\forall i$, $P^{(i)}_{util}(I) \leq \lceil (1-\mu) P^{(i)} \rceil - 1\}$.
- $\mathcal{I}_3$: 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., $\mathcal{I}_3 = \{I \mid \exists k, P^{(k)}_{util}(I) \geq \lceil (1-\mu) P^{(k)} \rceil\}$.

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 $T = T_1 + T_2 + T_3$.

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 is 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 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 [12, 19, 21]. 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:

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

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 $P^{\min} = \min_{i=1\ldots d} P^{(i)} \geq \frac{1}{\mu^2}$, we can derive:

$$\begin{aligned} \mu T_2 &= \mu \sum_{I \in \mathcal{I}_2} |I| \\ &\leq d \sum_{I \in \mathcal{I}_2} \sum_{j=1}^{n} \beta_{j,I} \cdot a_j(p_j') \\ &= d \sum_{j=1}^{n} \left(a_j(p_j') \cdot \sum_{I \in \mathcal{I}_2} \beta_{j,I} \right) . \end{aligned} \tag{8}$$

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:

$$(1-\mu) T_3 \leq d \sum_{j=1}^{n} \left(a_j(p_j') \cdot \sum_{I \in \mathcal{I}_3} \beta_{j,I} \right) . \tag{9}$$

Thus, combining Inequalities (8) and (9), we can get:

$$\begin{aligned} \mu T_2 + (1-\mu) T_3 &\leq d \sum_{j=1}^{n} \left(a_j(p_j') \cdot \sum_{I \in \mathcal{I}_2 \cup \mathcal{I}_3} \beta_{j,I} \right) \\ &\leq d \sum_{j=1}^{n} a_j(p_j') = d \cdot A(\mathbf{p}') . \end{aligned}$$ □

4.3 Approximation Results

We now derive the main approximation results of the multi-resource scheduling algorithm, which combines the resource allocation phase (Algorithm 1) and the list scheduling phase (Algorithm 2). The following theorem shows its approximation ratio for any number d of resource types.

Theorem 1. *For any $d \geq 1$ and if $P^{\min} \geq 7$, the performance of the multi-resource scheduling algorithm satisfies:*

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

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 $P^{\min} \geq 7$ 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 $P^{\min} \geq \frac{1}{\mu^2}$, we get:

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

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

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

Clearly, $f_d(\mu, \rho)$ is an increasing function of μ for all d. Thus, to minimize the function, we can set $\mu^* = 1 - \frac{1}{\phi}$. In this case, we require $P^{\min} \geq \frac{1}{(\mu^*)^2} \approx 6.854$ and we define $f_d(\rho) \triangleq f_d(\mu^*, \rho) = \frac{1}{\rho} + \frac{\phi d}{1-\rho}$. Now, 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 $f_d(\mu^*, \rho^*) = \phi d + 2\sqrt{\phi d} + 1$. □

We point out that, 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. [21]. Jansen and Zhang [19] showed that the algorithm actually achieves an even better ratio of 4.73 by proving a tighter critical-path bound than the one shown in Lemma 5. Unfortunately, their analysis cannot be generalized to the case with more than one type of resources.

While Theorem 1 proves the approximation ratio of the multi-resource scheduling algorithm for any d, the following theorem shows an improved result for large d. The proof is omitted here due to space constraint and can be found in the complete version of the paper [26].

Theorem 2. *For $d \geq 22$ and if $P^{\min} \geq d^{2/3}$, the performance of the multi-resource scheduling algorithm satisfies:*

$$\frac{T}{T_{OPT}} \leq d + 3\sqrt[3]{d^2} + O(\sqrt[3]{d}) .$$

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

Although Theorem 2 holds for a large number of resource types (i.e., $d \geq 22$) and is unlikely to be practical in today's resource management systems, the result does have significant theoretical importance. In particular, it gives the first approximation for general list-based algorithm that is asymptotically tight up to the dominating factor d in the context of multi-resource moldable job scheduling (see Theorem 6).

5 IMPROVED APPROXIMATION RESULTS FOR SOME SPECIAL GRAPHS

In the preceding section, we have derived the approximation ratios of the multi-resource scheduling algorithm 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 (DAG) is a *series-parallel (SP) graph* [2] 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 [21] to find a near-optimal resource allocation. The algorithm was proposed in the context of a single resource type, but can be readily adapted to work for multiple types of resources (by first discarding the subset of dominated resource allocations as shown in Step 1 of Algorithm 1). 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 $L(\mathbf{p}') \leq X$ can be found for a positive integer X, and finally performs a binary search on X. The following lemma shows the result. More details about the algorithm can be found in [21].

Lemma 7. *For a set of jobs whose precedence constraints form a series-parallel graph or a tree, and for any $\epsilon \geq 0$, an FPTAS (i.e., polynomial in $1/\epsilon$) exists, which can compute a resource allocation $\mathbf{p}' = (p'_1, p'_2, \ldots, p'_n)$ that satisfies:*

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

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 $d \geq 1$ and if $P^{\min} \geq 7$, the performance of the multi-resource scheduling algorithm for SP graphs or trees satisfies the following:*

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

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 $P^{\min} \geq \frac{1}{\mu^2}$, we get:

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

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

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

Clearly, $f_d(\mu)$ is an increasing function of μ for all d. 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 $P^{\min} \geq \frac{1}{(\mu^*)^2} \approx 6.854$. □

[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.

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

Theorem 4. *For any $d \geq 4$ and if $P^{\min} \geq d + 2\sqrt{d-1}$, the performance of the multi-resource scheduling algorithm for SP graphs or trees satisfies the following:*

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

The result 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 $P^{\min} \geq \frac{1}{\mu^2}$, we get:

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

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

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

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

$$\begin{aligned} g_d(\mu^*) &= (1+\epsilon) \cdot \left(\sqrt{d-1} + 1 + \frac{d-1}{1 - \frac{1}{\sqrt{d-1}+1}}\right) \\ &= (1+\epsilon) \cdot \left(d + 2\sqrt{d-1}\right) \ . \end{aligned}$$

□

5.2 Results for Independent Jobs

We finally consider independent jobs without any precedence constraints. For this case, Sun et al. [31] 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 [31] 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 [31].

Lemma 8. *For a set of independent jobs, a resource allocation $\mathbf{p}' = (p'_1, p'_2, \ldots, p'_n)$ can be found in polynomial time, such that:*

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

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 independent jobs, while the area bound (Lemma 6) remains unchanged, we show 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 $\mathcal{I}_2$ as well as all intervals in $\mathcal{I}_1$. Thus, job j must be unadjusted (since it is executed during $\mathcal{I}_1$). Thus, we have $T_1 + T_2 \leq t_j(p_j) = t_j(p'_j) \leq \max_{j=1\ldots n} t_j(p'_j) = C(\mathbf{p}')$. □

Theorem 5. *The performance of multi-resource scheduling for independent jobs satisfies $T/T_{OPT} \leq r$, 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 [31] 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:

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

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:

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

The overall approximation ratio is 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 $g_d(\mu^*) = d + 2\sqrt{d-1}$. □

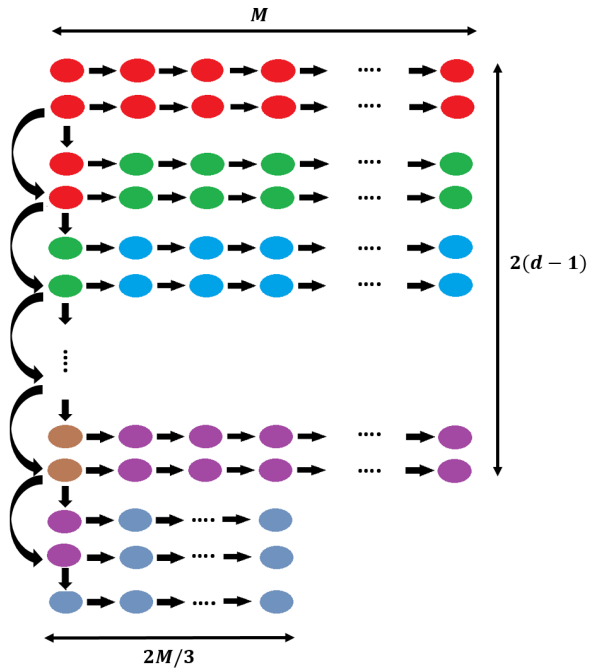

Figure 1: Lower bound instance with an approximation ratio of d for any deterministic list scheduling algorithm with local job priority considerations.

6 LOWER BOUND FOR LIST SCHEDULING

Lastly, we prove a lower bound of d on the approximation ratio of any deterministic algorithm that, for the second phase, uses list scheduling with only *local* priority considerations (i.e., without taking into account the precedence graphs when assigning priorities to the jobs). This lower bound holds regardless of the resource allocation scheme for the first phase. The result shows that our multi-resource scheduling algorithms essentially achieve tight approximation ratios up to the dominating factor for large d among the generic class of local list scheduling schemes.

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. Figure 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 optimal schedule can be obtained by prioritizing the job dependencies going downward, resulting 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

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 $d \geq 1$ [31] • $1.619d + 1$ for $d = 3$ • $d + 2\sqrt{d-1}$ for $d \geq 4$

utilize one type of resource at any time, resulting 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:

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

This completes the proof of the theorem. □

7 CONCLUSION

In this paper, we have studied the problem of scheduling parallel jobs with precedence constraints under multiple types of schedulable resources. We focused on moldable jobs, which allow the scheduler to flexibly select a variable set of resources before the execution of the jobs, and the goal is to minimize the overall completion time, or the makespan. We have proposed a multi-resource scheduling algorithm that adopts the two-phase approach by combining an approximate resource allocation and an extended list scheduling scheme. We have proven approximation ratios of the algorithm for the general precedence graph, as well as for some special graphs including SP-DAGs or trees and independent jobs. The results are summarized in Table 1. We have also proven a lower bound on the approximation ratio of any local list scheduling scheme, which shows that our algorithm achieves the optimal asymptotic performance up to the dominating factor.

We point out that the lower bound proven 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., giving priority to the jobs on the critical path). It remains an open question to find such an algorithm by showing a better approximation ratio than d, or to prove a matching lower bound for *any* list-based scheduling scheme.

REFERENCES

[1] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. 2011. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurr. Comput. : Pract. Exper.* 23, 2 (2011), 187–198.

[2] Hans L. Bodlaender and Babette de Fluiter. 1996. Parallel algorithms for series parallel graphs. In *ESA*. 277–289.

[3] George Bosilca, Aurelien Bouteiller, Anthony Danalis, Mathieu Faverge, Thomas Herault, and Jack J. Dongarra. 2013. PaRSEC: Exploiting Heterogeneity to Enhance Scalability. *Computing in Science and Engg.* 15, 6 (2013), 36–45.

[4] Marco Caccamo, Rodolfo Pellizzoni, Lui Sha, Gang Yao, and Heechul Yun. 2013. MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-Core Platforms. In *RTAS*. 55–64.

[5] C. Chen. 2018. An Improved Approximation for Scheduling Malleable Tasks with Precedence Constraints via Iterative Method. *IEEE Transactions on Parallel and Distributed Systems* 29, 9 (2018), 1937–1946.

[6] Chi-Yeh Chen and Chih-Ping Chu. 2013. A 3.42-Approximation Algorithm for Scheduling Malleable Tasks under Precedence Constraints. *IEEE Trans. Parallel Distrib. Syst.* 24, 8 (2013), 1479–1488.

[7] Prabuddha De, E. James Dunne, Jay B. Ghosh, and Charles E. Wells. 1997. Complexity of the Discrete Time-Cost Tradeoff Problem for Project Networks. *Operations Research* 45, 2 (1997), 302–306.

[8] Prabuddha De, E. James Dunne, Jay B. Ghosh, and Charles E. Wells. 1995. The discrete time-cost tradeoff problem revisited. *European Journal of Operational Research* 81, 2 (1995), 225–238.

[9] Gökalp Demirci, Henry Hoffmann, and David H. K. Kim. 2018. Approximation Algorithms for Scheduling with Resource and Precedence Constraints. In *STACS*.

[10] J. Du and J. Y.-T. Leung. 1989. Complexity of Scheduling Parallel Task Systems. *SIAM J. Discret. Math.* 2, 4 (1989), 473–487.

[11] Dror G. Feitelson. 1997. Job Scheduling in Multiprogrammed Parallel Systems (Extended Version). *IBM Research Report RC19790(87657)* (1997).

[12] Anja Feldmann, Ming-Yang Kao, Jiří Sgall, and Shang-Hua Teng. 1998. Optimal On-Line Scheduling of Parallel Jobs with Dependencies. *Journal of Combinatorial Optimization* 1, 4 (1998), 393–411.

[13] M. R. Garey and R. L. Graham. 1975. Bounds for multiprocessor scheduling with resource constraints. *SIAM J. Comput.* 4, 2 (1975), 187–200.

[14] Thierry Gautier, Xavier Besseron, and Laurent Pigeon. 2007. KAAPI: A Thread Scheduling Runtime System for Data Flow Computations on Cluster of Multiprocessors. In *PASCO*. 15–23.

[15] Yuxiong He, Jie Liu, and Hongyang Sun. 2011. Scheduling Functionally Heterogeneous Systems with Utilization Balancing. In *IPDPS*. 1187–1198.

[16] Yuxiong He, Hongyang Sun, and Wen-Jing Hsu. 2007. Adaptive Scheduling of Parallel Jobs on Functionally Heterogeneous Resources. In *ICPP*. 43.

[17] K. Jansen and F. Land. 2018. Scheduling Monotone Moldable Jobs in Linear Time. In *IPDPS*. 172–181.

[18] Klaus Jansen and Hu Zhang. 2005. Scheduling Malleable Tasks with Precedence Constraints. In *SPAA*. 86–95.

[19] Klaus Jansen and Hu Zhang. 2006. An Approximation Algorithm for Scheduling Malleable Tasks Under General Precedence Constraints. *ACM Trans. Algorithms* 2, 3 (2006), 416–434.

[20] Renaud Lepère, Gregory Mounié, and Denis Trystram. 2002. An approximation algorithm for scheduling trees of malleable tasks. *European Journal of Operational Research* 142, 2 (2002), 242–249.

[21] Renaud Lepère, Denis Trystram, and Gerhard J. Woeginger. 2002. Approximation Algorithms for Scheduling Malleable Tasks Under Precedence Constraints. *Int. J. Found. Comput. Sci.* 13, 4 (2002), 613–627.

[22] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. 2012. On the role of burst buffers in leadership-class storage systems. In *MSST*. 1–11.

[23] Walter Ludwig and Prasoon Tiwari. 1994. Scheduling Malleable and Nonmalleable Parallel Tasks. In *SODA*. 167–176.

[24] Gregory Mounié, Christophe Rapine, and Denis Trystram. 2007. A 3/2-Approximation Algorithm for Scheduling Independent Monotonic Malleable Tasks. *SIAM J. Comput.* 37, 2 (2007), 401–412.

[25] Martin Niemeier and Andreas Wiese. 2012. Scheduling with an Orthogonal Resource Constraint. In *WAOA*. 242–256.

[26] Lucas Perotin, Hongyang Sun, and Padma Raghavan. 2021. Multi-Resource List Scheduling of Moldable Parallel Jobs under Precedence Constraints. (2021). arXiv:cs.DC/2106.07059

[27] S. Ristov, R. Prodan, M. Gusev, and K. Skala. 2016. Superlinear speedup in HPC systems: Why and when?. In *Federated Conference on Computer Science and Information Systems (FedCSIS)*. 889–898.

[28] David B. Shmoys, Clifford Stein, and Joel Wein. 1994. Improved Approximation Algorithms for Shop Scheduling Problems. 23, 3 (1994), 617–632.

[29] Martin Skutella. 1998. Approximation Algorithms for the Discrete Time-Cost Tradeoff Problem. *Math. Oper. Res.* 23, 4 (1998), 909–929.

[30] Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu. 2016. Knights Landing: Second-Generation Intel Xeon Phi Product. *IEEE Micro* 36, 2 (2016), 34–46.

[31] H. Sun, R. Elghazi, A. Gainaru, G. Aupy, and P. Raghavan. 2018. Scheduling Parallel Tasks under Multiple Resources: List Scheduling vs. Pack Scheduling. In *IPDPS*. 194–203.

[32] John Turek, Joel L. Wolf, and Philip S. Yu. 1992. Approximate Algorithms Scheduling Parallelizable Tasks. In *SPAA*.

[33] Qingzhou Wang and Kam Hoi Cheng. 1992. A Heuristic of Scheduling Parallel Tasks and Its Analysis. *SIAM J. Comput.* 21, 2 (1992), 281–294.

[34] Meng Xu, Linh Thi Xuan Phan, Xuan Phan, Hyon-Young Choi, and Insup Lee. 2017. vCAT: Dynamic Cache Management Using CAT Virtualization. In *RTAS*.