An Empirical Evaluation of Multi-Resource Scheduling for Moldable Workflows

Sandhya Kandaswamy

Submitted to the graduate degree program in Department of Electrical Engineering and Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Dr. Hongyang Sun, Chairperson

Committee members

Dr. Suzanne M. Shontz

Dr. Heechul Yun

Date defended: December 09, 2022

The Thesis Committee for Sandhya Kandaswamy certifies
that this is the approved version of the following thesis:

An Empirical Evaluation of Multi-Resource Scheduling for Moldable Workflows

Dr. Hongyang Sun, Chairperson

Date approved: December 09, 2022

Abstract

Resource scheduling plays a vital role in High-Performance Computing (HPC) systems. However, most scheduling research in HPC has focused on only a single type of resource (e.g., computing cores or I/O resources). With the advancement in hardware architectures and the increase in data-intensive HPC applications, there is a need to simultaneously embrace 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. This thesis performs an empirical evaluation of a recently proposed multi-resource scheduling algorithm for minimizing the overall completion time (or makespan) of computational workflows comprised of moldable parallel jobs. Moldable parallel jobs allow the scheduler to select the resource allocations at launch time and 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). The algorithm was proven to have a worst-case approximation ratio that grows linearly with the number of resource types for moldable workflows. In this thesis, a comprehensive set of simulations is conducted to empirically evaluate the performance of the algorithm using synthetic workflows generated by DAGGEN and moldable jobs that exhibit different speedup profiles. The results show that the algorithm fares better than the theoretical bound predicts, and it consistently outperforms two baseline heuristics under a variety of parameter settings, illustrating its robust practical performance.

Contents

Abstract **iii**

1 Introduction **1**

2 Related Works **4**

2.1 Approximation Algorithms to Minimize Makespan 4

2.2 Multi-Resource Scheduling under Alternative Models 5

3 Problem Statement **7**

3.1 Scheduling Model 7

3.2 Assumptions 8

3.3 Scheduling Objective 9

4 Scheduling Algorithm **10**

4.1 Phase 1: Resource Allocation 10

4.2 Phase 2: List Scheduling 12

4.3 Approximation Result 13

5 Simulation Results **15**

5.1 Simulation Setup 15

5.2 Performance Comparison of Algorithms 19

5.3 Impact of System Parameters 21

5.4 Impact of Graph Structure 25

5.5 Impact of Job Speedup Functions 27

5.6 Impact of MRSA Parameters 32

6 Conclusion **35**

List of Figures

5.1 A graph consisting of 100 jobs generated by DAGGEN with fat = 0.5, density = 0.5, regular = 0.5, and jump = 1. 16

5.2 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. 19

5.3 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. 21

5.4 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. 22

5.5 Impact of the number of jobs n on the performance of the three algorithms under the four speedup models. 23

5.6 Impact of the amount of available resources P on the performance of the three algorithms under the four speedup models. 24

5.7 Impact of the number of resource types d on the performance of the three algorithms under the four speedup models. 25

5.8 Impact of the fat parameter in DAGGEN on the performance of the three algorithms under the four speedup models. 27

5.9 Impact of the *density* parameter in DAGGEN on the performance of the three algorithms under the four speedup models. 28

5.10 Impact of the *regular* parameter in DAGGEN on the performance of the three algorithms under the four speedup models. 29
5.11 Impact of the *jump* parameter in DAGGEN on the performance of the three algorithms under the four speedup models. 30
5.12 Impact of the sequential fraction s_0 (for the two Amdahl models) and the efficiency factor α_i (for the two power models) on the performance of the three algorithms. . 32
5.13 Impact of parameters μ and ρ on the performance (average normalized makespan) of MRSA under the four speedup models. 34

Chapter 1

Introduction

Complex scientific workflows running in today's High-Performance Computing (HPC) systems are typically modeled as Directed Acyclic Graphs (DAGs). The DAGs' nodes correspond to the jobs in the workflows, while their edges represent the dependencies (or precedence constraints) between the jobs. Improving the performance of scientific applications requires effective workflow scheduling. The majority of current dynamic runtime schedulers [16, 3, 6] which guarantee effective execution of workflows focus on only one type of resource (e.g., computing cores or I/O resources). However, 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 cores, cache, memory, I/O, and network resources) in the design of runtime schedulers. In fact, contemporary HPC systems come with more memory/storage levels (such as NVRAMs, SSDs, and burst buffers), all of which can be scheduled among the system's concurrently executing jobs. Advancements in architectural and software features (e.g., high-bandwidth memory [32], cache partitioning [35], bandwidth reservation [7]) can also be leveraged to facilitate efficient multi-resource scheduling for enhancing the overall performance of the application and/or system.

This thesis performs an empirical evaluation of multi-resource scheduling that schedules multiple types of resources to minimize the workflow's overall completion time, or *makespan*. The workflows are comprised of a set of parallel jobs with DAG-based precedence constraints. We concentrate on parallel jobs that are *moldable* [13], which allows the scheduler to select a variable set of resources for a job, but the resource allocations cannot be modified once the job begins its execution. This contrasts with *rigid* jobs, whose resource allocations are fixed and static, and with

malleable jobs, whose resource allocations can change dynamically while the job is running. As a result, moldable jobs can readily adapt to the different amounts of available resources while staying easy to design and implement.

We focus on evaluating a recently proposed approximation algorithm [28] for the multi-resource scheduling problem, which is strongly NP-complete by containing the single-resource scheduling problem as a special case [11]. The algorithm, called MRSA (Multi-Resource Scheduling Algorithm), adopts a two-phase approach that is widely used for scheduling moldable jobs [34]. Specifically, the first phase computes a resource allocation for all jobs on different resource types, and the second phase applies an extended list scheduling scheme to schedule the jobs. MRSA has been proven to have an approximation ratio that grows linearly with the number of resource types [28].

To empirically evaluate the performance of MRSA, we conducted a comprehensive set of simulations using synthetic workflows generated by DAGGEN [1], which is a task graph generator capable of generating DAGs of different structures. We also generated moldable parallel jobs that exhibit different runtime characteristics on multiple resource types by extending common speedup profiles that follow Amdahl's law [2] and power law [29] models. Our simulation results show, under a variety of parameter settings, that:

- MRSA has an average-case performance that is much better than the worst-case theoretical bound predicts;
- MRSA consistently outperforms two baseline heuristics, namely, minTime and minArea, which allocate resources to minimize the execution time and the area of each job, respectively.

The empirical study reported in this thesis nicely complements the theoretical result proven in [28], which together illustrate the robust practical performance of MRSA for the multi-resource scheduling of moldable workflows.

The rest of this thesis is organized as follows. Chapter 2 reviews some related work on multi-

resource scheduling. Chapter 3 formally introduces the scheduling model and problem statement. Chapter 4 briefly describes how MRSA works and its approximation result. Chapter 5 presents our simulation results evaluating the algorithm under a variety of parameter settings. Finally, Chapter 6 provides the concluding remarks.

Chapter 2

Related Works

This chapter reviews some related work on approximation algorithms designed for multi-resource scheduling to minimize the makespan, as well as algorithms designed for alternative multi-resource models and objectives.

2.1 Approximation Algorithms to Minimize Makespan

Scheduling parallel jobs (independent or with precedence constraints) to minimize makespan is a strong NP-complete problem [11]. Thus, many approximation algorithms have been proposed over the years (e.g., [34, 24, 14, 23, 21, 25, 22]). Most of these works, however, focused on only a single resource type.

There has also been some development for multi-resource scheduling under various parallel job models. Garey and Graham [15] considered scheduling sequential jobs on identical machines but with d additional types of resources. Further, they adopted a *rigid* job scheduling model, in that, each job has a fixed set of resource requirements from each resource type. They presented a list-scheduling algorithm and proved that it is a $(d+1)$-approximation when jobs are independent and the number of machines is not a constraining factor. When there is only one additional resource type, i.e., $d = 1$, Demirci et al. [10] presented an $O(\log n)$-approximation algorithm for jobs with precedence constraints, where n is the number of jobs, and Niemeier and Wiese [26] presented a $(2+\varepsilon)$-approximation algorithm for independent jobs.

He et al. [20, 19] considered scheduling 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. During

runtime, The amount of resource allocation can be dynamically changed, making it a *malleable* job scheduling model. For this model, they demonstrated that list scheduling achieves $(d+1)$-approximation. Shmoys et al. [30] considered a similar model but restricted the tasks of each DAG to be processed sequentially. They proved a polylog approximation result in the number of machines and job length.

Sun et al. [33] considered scheduling independent *moldable* jobs on d types of resources. They presented two scheduling algorithms: one based on list scheduling with a $2d$-approximation and the other based on shelf scheduling with a $(2d+1)$-approximation. Additionally, 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. Perotin et al. [28] applied a similar model while considering jobs with precedence constraints. They presented a list-based algorithm and proved that it has an approximation ratio of $1.619d + 2.545\sqrt{d} + 1$. This thesis conducts an empirical evaluation of the algorithm in [28] and reports its practical performance under a variety of parameter settings.

2.2 Multi-Resource Scheduling under Alternative Models

Some prior works have also proposed multi-resource scheduling algorithms that work under alternative models or objectives. Beaumont et al. [5] and Eyraud-Dubois and Kumar [12] 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 scheduling model can be found in [4].

Additionally, Ghodsi et al. [17] focused on the objective of resource allocation fairness in a multi-user setting. They proposed the Dominant Resource Fairness (DRF) algorithm that aims at maximizing the minimum dominant share across all users. Grandl et al. [18] considered schedul-

ing 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. [27] studied a similar problem but with two resources only (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.

Chapter 3

Problem Statement

This chapter formally introduces the multi-resource scheduling problem for moldable workflows. A scheduling model is first presented, followed by some assumptions made on the model. The chapter ends with a statement on the scheduling objective.

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, I/O, network). $P^{(i)}$ is a total amount of available resources for each resource type i. 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 job j for each resource type i, determines its execution time $t_j(p_j)$. We further define the following concepts for the job.

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

Furthermore, a set of *precedence constraints* is defined, which when combined with the jobs, creates a workflow or a Directed Acyclic Graph (DAG), denoted as $G = (V, E)$. In the workflow, each node $j \in V$ denotes a job and a directed edge $(j_1 \rightarrow j_2) \in E$ specifies that job j_2 cannot start executing until job j_1 is completed. Thus, j_1 is called an immediate *predecessor* of j_2, and j_2 is called an immediate *successor* of j_1.

3.2 Assumptions

The following reasonable assumptions on the resource allocation and execution time of the jobs are made.

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 assumption aligns with the common practice where resource types, such as memory or cache, are also typically allocated in discrete chunks (e.g., memory blocks, cache lines).

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.*

Techniques such as application modeling or profiling, performance prediction, or interpolation from historic data can be employed to determine the execution time of an application, thus validating this assumption.

Assumption 3 (Monotonic Jobs) *Given two resource allocations p_j and q_j for a job j, 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:*

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

The first inequality implies that a job cannot be executed longer by providing more resources (that is *non-increasing*) and the second inequality implies that the speedup of a job must be *non-superlinear* with respect to any resource type due to scheduling overheads. This also generalizes the monotonic job assumption under a single resource type [23, 25]. Further, both properties have been observed for many real-world applications.

3.3 Scheduling Objective

The objective is to determine a valid schedule for the jobs such that the maximum completion time or makespan of the workflow is minimized. The following two decisions define a schedule:

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

The completion time of a job j is defined as $c_j = s_j + t_j(p_j)$. The makespan of the workflow is given by $T = \max_j c_j$. A *valid* schedule 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;
- Let j_1 be an immediate predecessor of j_2, i.e., $j_1 \rightarrow j_2$, then the completion time of j_1 should not be later than the starting time of j_2, i.e., $c_{j_1} \leq s_{j_2}$.

The above multi-resource scheduling problem is an NP-complete problem, and it can be solved by designing a polynomial-time approximation algorithm. Here, an algorithm is said to be an *r-approximation* if its makespan satisfies $\frac{T}{T_{\text{OPT}}} \leq r$ for any workflow, where T_{OPT} denotes the optimal makespan for the same workflow. It is shown in [28] that the total area of all jobs in the workflow (as defined in Section 3) as well as the total execution times of the jobs along the critical path of the workflow can both be used as lower bounds on the optimal makespan. These lower bounds can then be conveniently explored to prove the approximation ratio of an algorithm.

Chapter 4

Scheduling Algorithm

This chapter describes MRSA (Multi-Resource Scheduling Algorithm) proposed in [28], which will be empirically evaluated in this thesis. MRSA adopts a *two-phase approach* that has been widely used for scheduling moldable jobs on a single type of resource [34, 23, 22]. The chapter describes both phases of the algorithm and concludes with its approximation result for moldable workflows.

4.1 Phase 1: Resource Allocation

The first phase concerns allocating resources to the jobs. For this phase, we consider a relevant discrete time-cost tradeoff problem [9], which has been well-studied in the literature on project management.

Definition 1 (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.,*

$$t_{j,i_1} \leq t_{j,i_2} \Rightarrow c_{j,i_1} \geq c_{j,i_2} \ . \tag{4.1}$$

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 bi-criteria, and a tradeoff exists between the total project duration and the total cost. The problem has been shown to be NP-complete when optimizing any one objective subject to a constraint on the other [8]. Skutella [31] presented a polynomial-time algorithm, which, given any feasible duration-cost pair (D,B) and any $\rho \in (0,1)$, finds a realization σ for the project that satisfies:

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

The multi-resource allocation problem can be transformed into the DTCT problem and solved using the above approximation result in [31]. A task j is created for each job j in the graph. The set of alternatives for the task corresponds to the set of resource allocations for the job. Similarly, 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 $\mathscr{S}$ be the set of all $Q = \prod_{i=1}^{d} P^{(i)}$ possible resource allocations for a job and in order to meet the condition (4.1) in Definition 1, for each job j, the subset $\mathscr{D}_j \subset \mathscr{S}$ of *dominated* allocations are ignored, which is defined as:

$$\mathscr{D}_j = \{p_j \mid \exists q_j, t_j(q_j) < t_j(p_j) \text{ and } a_j(q_j) < a_j(p_j)\} , \tag{4.2}$$

and only the remaining set of *non-dominated* allocations, denoted by $\mathscr{N}_j = \mathscr{S} \backslash \mathscr{D}_j$, is used to specify the alternatives of the task. Thus, a realization σ for the project, the total project duration $D(\sigma)$ and the total cost $B(\sigma)$ correspond to a resource allocation decision, the total execution time of the jobs along the critical path, and the total average area of all jobs in the workflow, respectively.

By adapting the algorithm in [31] for the DTCT problem, we can find an initial resource allo-

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, given values for parameters ρ and μ.
Output: Resource allocation $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ for all jobs.
begin
  (Step 1): For each job j, discard the subset $\mathscr{D}_j \subset \mathscr{S}$ of dominated resource allocations as defined in Equation (4.2);
  (Step 2): Transform the resource allocation problem into the DTCT problem and adapt the algorithm in [31] to obtain an initial resource allocation $\mathbf{p}'$;
  (Step 3): For each job j and each resource type i, adjust the initial resource allocation in $\mathbf{p}'$ based on Equation (4.3) to obtain a final resource allocation $\mathbf{p}$.
end

cation $\mathbf{p}'$ for the jobs, for any parameter $\rho \in (0,1)$. This initial allocation is subsequently adjusted to get the final resource allocation $\mathbf{p}$, with the aim of limiting the maximum resource usage of any job under any resource type, enabling more effective scheduling in the algorithm's second phase. Adopting the strategy for scheduling under a single type of resource [23, 22], the final resource allocation for each job j on each resource type i is obtained as follows:

$$p_j^{(i)} = \begin{cases} \lceil \mu P^{(i)} \rceil, & \text{if } p'^{(i)}_j > \lceil \mu P^{(i)} \rceil \\ p'^{(i)}_j, & \text{otherwise} \end{cases} \tag{4.3}$$

where $p'^{(i)}_j$ is the corresponding resource allocation in $\mathbf{p}'$ and $\mu \in (0, 0.5)$ is used to regulate the job's maximum resource usage.

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

4.2 Phase 2: List Scheduling

Given the resource allocation decision $\mathbf{p}$ from the first phase, the second phase schedules the jobs by making a starting time decision $\mathbf{s}$. The well-known list scheduling strategy (shown in Algorithm 2), which is extended to work with multiple types of resources is implemented to determine the starting times and finally schedule the jobs.

Algorithm 2: List Scheduling (Phase 2)

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

```
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
```

When all of a job's immediate predecessors in the workflow have finished or it has no immediate predecessor, the job is said to be *ready*. Initially, all the ready jobs are added to the queue $\mathscr{Q}$ (in any order). The algorithm inserts any new job k' that becomes ready as a result of the completion of job k at time 0 or anytime a running job k completes. Then, the $\mathscr{Q}$ is checked for ready jobs that can be executed at the current time and a ready job j is executed, provided the resource allocation p_j is at most the total amount of available resources across all resource types.

4.3 Approximation Result

MRSA combines the resource allocation phase (Algorithm 1) and the list scheduling phase (Algorithm 2). It was proven in [28] to have a worst-case approximation ratio that grows linearly with the number d of resource types for any moldable workflow. The following theorem shows the approximation result. The proof can be found in [28].

Theorem 1 *For any $d \geq 1$ and if $P^{\min} \triangleq \min_{i=1\ldots d} P^{(i)} \geq 7$, the makespan of MRSA for any moldable workflow satisfies:*

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

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

It is important to note that for the majority of resource types (e.g., processors, memory blocks, cache lines), $P^{\min} \geq 7$ reflects a reasonable condition on the amount of resources.

Chapter 5

Simulation Results

This chapter presents the experimental results conducted by using simulations for evaluating the performance of MRSA and comparing it against two heuristic algorithms. The chapter starts by describing the simulation setup and then shows the simulation results under various parameter settings.

5.1 Simulation Setup

Workflow Generation. A workflow consisting of moldable jobs with precedence constraints is modeled as a Directed Acyclic Graph (DAG), whose nodes and edges represent the jobs and their dependencies, respectively. We generate workflows using DAGGEN [1], a synthetic task graph generator capable of generating DAGs of different structures. The graphs generated by DAGGEN have their tasks organized in layers, and important parameters that influence the structure of the graphs are described below.

- *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.

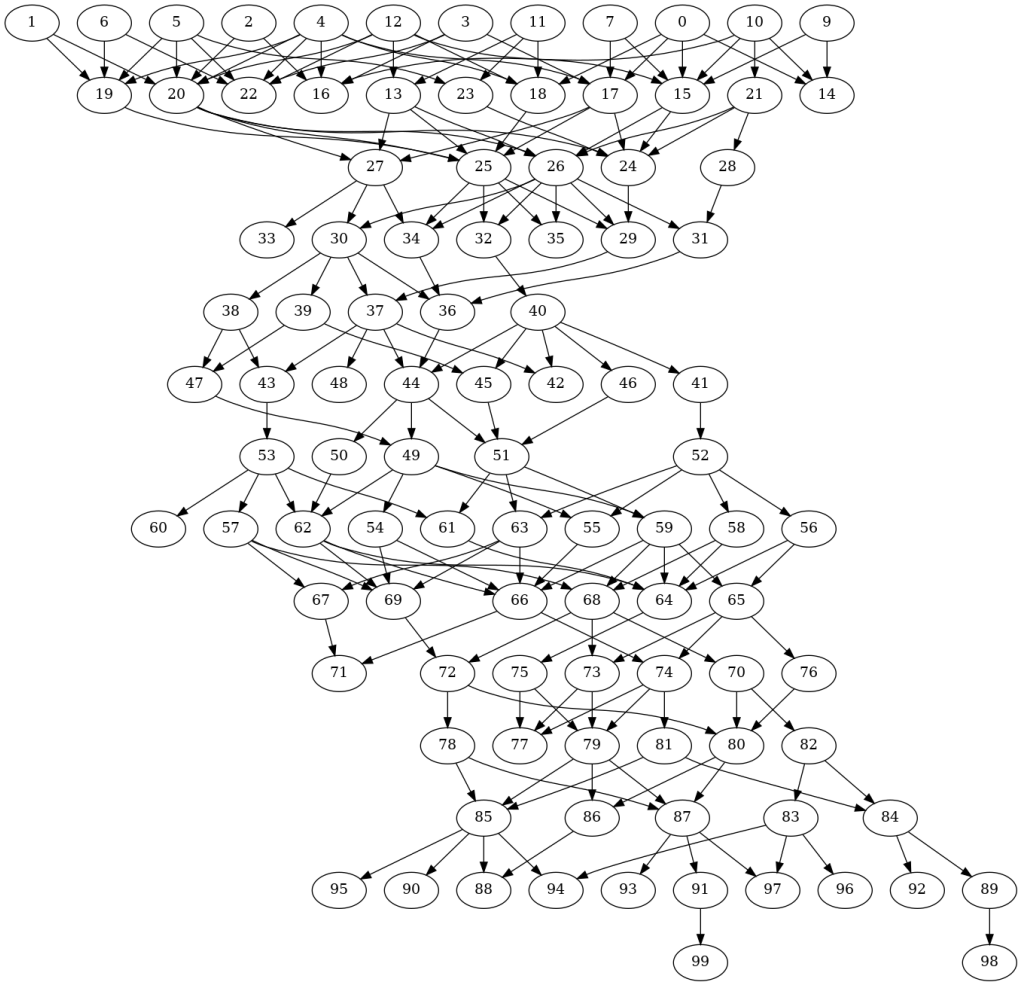


Figure 5.1: A graph consisting of 100 jobs generated by DAGGEN with fat = 0.5, density = 0.5, regular = 0.5, and jump = 1.

The range of possible values for fat, density, and regular is between 0 and 1, and jump can take any integer value of 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 5.4, we will also vary these values to evaluate their impacts on the performance of the algorithms. Figure 5.1 shows a graph consisting of 100 jobs generated by DAGGEN under the default 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 consider the following four execution time functions for the jobs that extend the classical Amdahl's

law [2] and power law [29] 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*: $t(p) = W\left(\sum_{i=1}^{d} \frac{s_i}{\left(p^{(i)}\right)^{\alpha_i}}\right)$;
- *Power-Max*: $t(p) = W\left(\max_{i=1..d} \frac{s_i}{\left(p^{(i)}\right)^{\alpha_i}}\right)$.

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.

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 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 $(0, 1]$. In Section 5.5, 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.

We note that all the speedup models considered above satisfy the monotonic job assumption stated in Chapter 3.

Comparing Algorithms. The multi-resource scheduling algorithm MRSA is evaluated and compared against the following two baseline heuristics.

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

Both heuristics 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 algorithms, we use the LPT

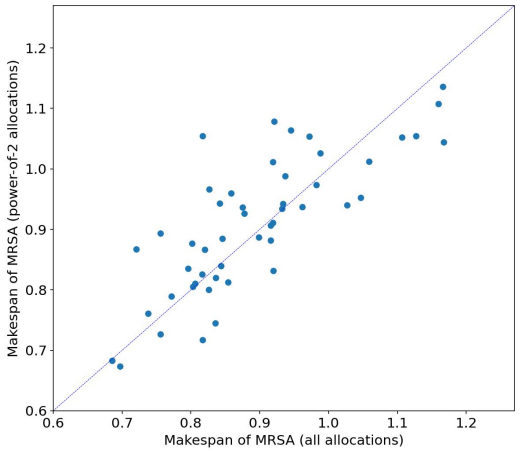
(Longest Processing Time) priority rule to order the jobs in the list schedule, which is known to work well for reducing the makespan. Thus, if the waiting queue contains more than one job, these jobs will be ordered by non-increasing order of execution time given their resource allocations.

While minTime and minArea could efficiently compute the resource allocations for a job in $O(1)$ time[1], MRSA would take $O(\Pi_{i=1}^{d} P^{(i)})$ time by examining all possible resource allocations. When the total amount of available resources in the system is large, the complexity of MRSA is 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. Although this leads to a factor of 2 increase in the approximation ratio of the algorithm in the worst case, it drastically reduces the complexity of the algorithm to $O(\Pi_{i=1}^{d} \lg P^{(i)})$, thus allowing us to simulate larger problem instances in a reasonable amount of time.

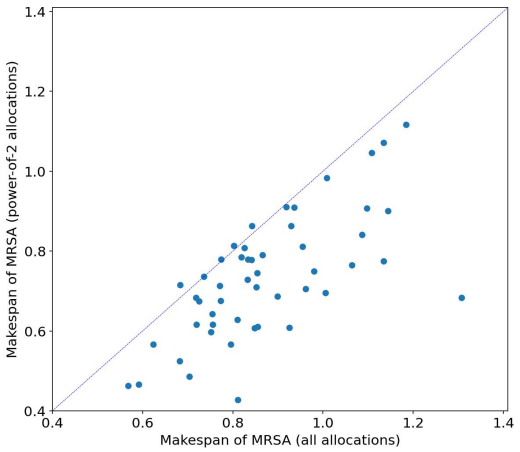
The scatter plots in Figure 5.2 show the makespans of MRSA for 50 workflows with $n = 30$ jobs, $d = 3$ resource types, and $P^{(i)} = 64$ for each resource type under the four speedup models. In the plots, each point represents a workflow, the x-axis represents the makespan when MRSA considers all resource allocations, and the y-axis represents the corresponding makespan when MRSA uses only power-of-2 allocations. In terms of the running time, it took more than 10 hours to complete the simulation for each speedup model when considering all allocations, while the simulation took only a few seconds when 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 smaller than those obtained when using all allocations for most workflows[2]. Given these results and to enable faster simulations, we will use power-of-2 allocations for MRSA in all the experiments.

[1]Given the monotonic job assumption, minTime could allocate all the available resources to a job, i.e., $p = (P^{(1)}, P^{(2)}, \ldots, P^{(d)})$, for minimizing its execution time, while minArea typically allocates only one unit of resource in each resource type, i.e., $p = (1, 1, \ldots, 1)$, for minimizing the job's area.

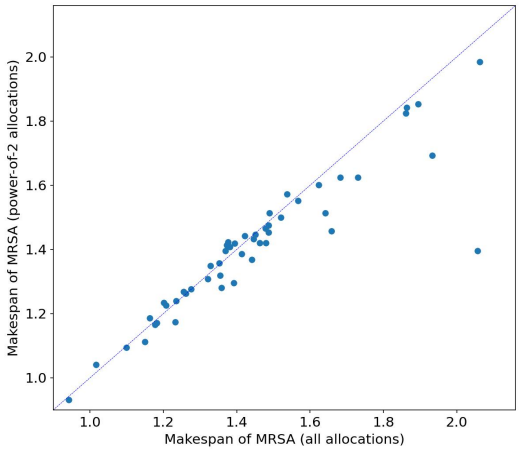
[2]This is possibly because power-of-2 allocations potentially allow the ready jobs to be better packed/scheduled in the second phase of the algorithm.



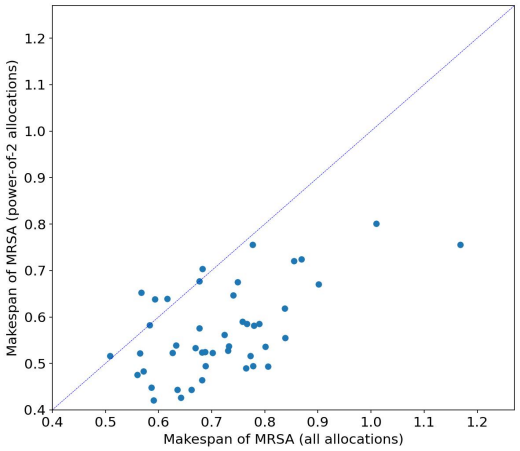
(a) Amdahl-Sum



(b) Amdahl-Max



(c) Power-Sum



(d) Power-Max

Figure 5.2: 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.

5.2 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 $1 \leq i \leq d$.

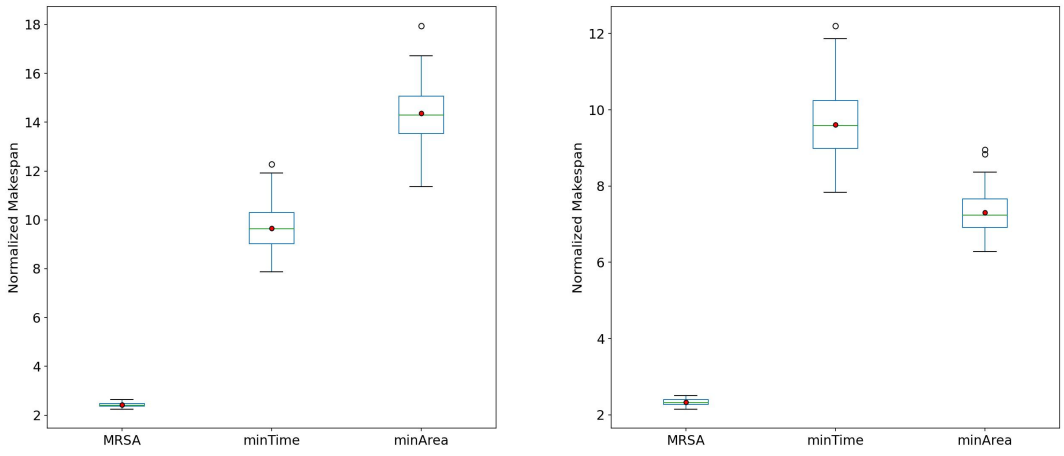
- *Non-uniform P*: the total amount of available resources may differ among different resource types. Specifically, for each resource type i, the value of $P^{(i)}$ is randomly selected from $\{32, 64, 128, 256, 512, 1024\}$.

Other parameters (e.g., for generating graphs and job execution times) are set as their default values/ranges as described in Section 5.1. In the experiments, we randomly generated 50 workflows and obtained, for each workflow, the makespans of the three algorithms by simulation. We then normalize the obtained makespans by the lower bound [28] for that workflow and finally report the statistics on the normalized makespans across the 50 workflows.

The boxplots in Figure 5.3 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.

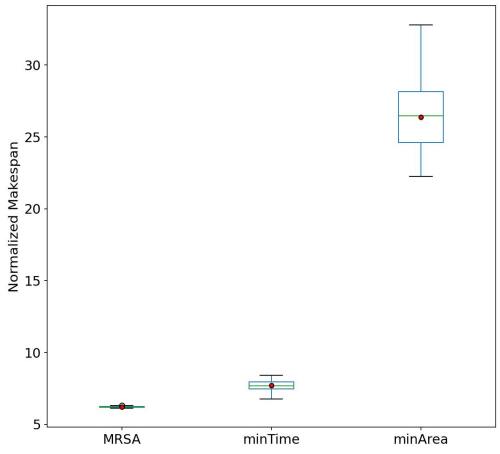
Figure 5.4 shows the corresponding results in the non-uniform P scenario. The results are quite similar to those in the uniform scenario, and MRSA again significantly outperforms the other two heuristics in terms of the makespan distribution, as well as the mean and median values. In contrast to the 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.

Thanks to the similarity of the results in the uniform P and non-uniform P scenarios, we will use the uniform case for all subsequent experiments, which evaluate the impacts of different parameters from the default setting considered in this section.

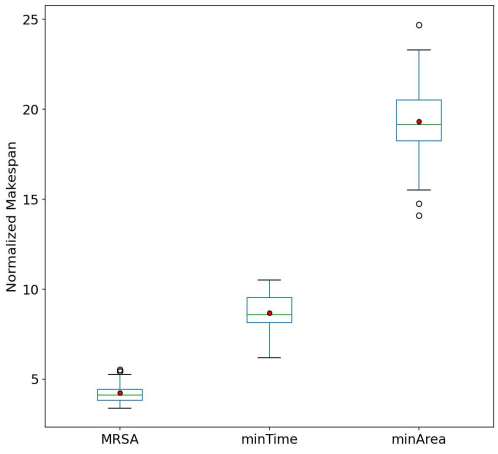


(a) Amdahl-Sum

(b) Amdahl-Max



(c) Power-Sum

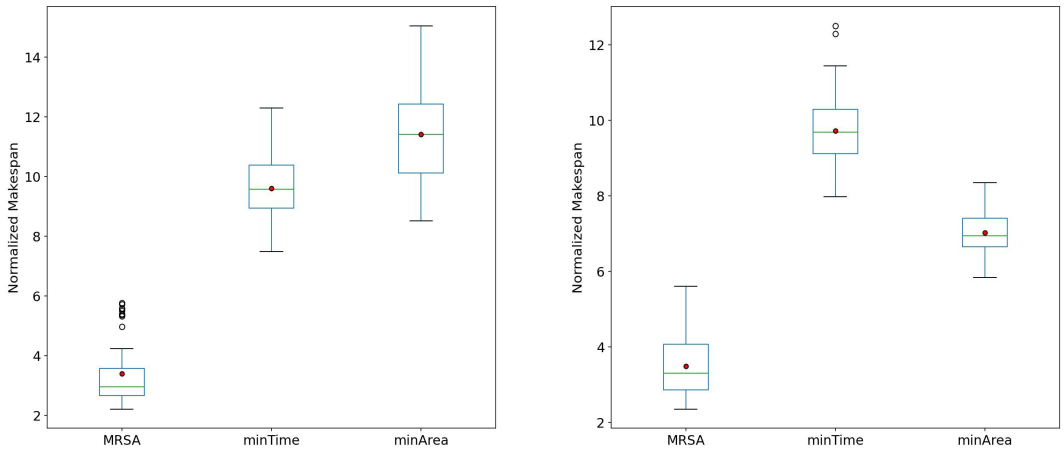


(d) Power-Max

Figure 5.3: 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.

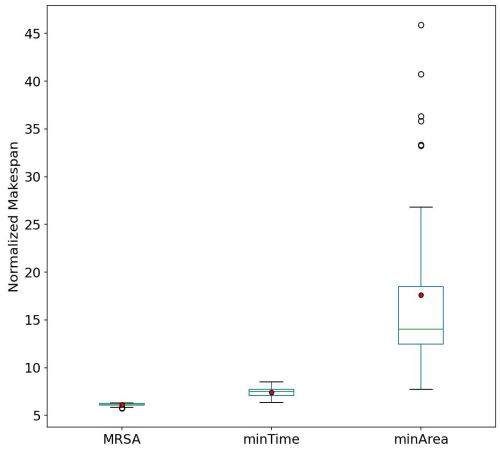
5.3 Impact of System Parameters

This section presents the results of simulations 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 5.2. We again generate 50 workflows and compute the normalized makespans of the three algorithms for each workflow. The results are then reported by averaging the normalized makespans across the 50 workflows for each algorithm.

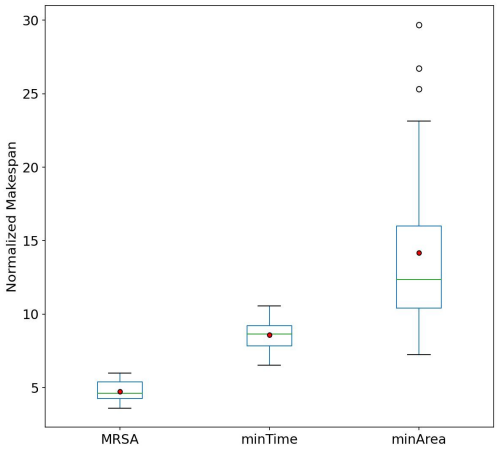


(a) Amdahl-Sum

(b) Amdahl-Max



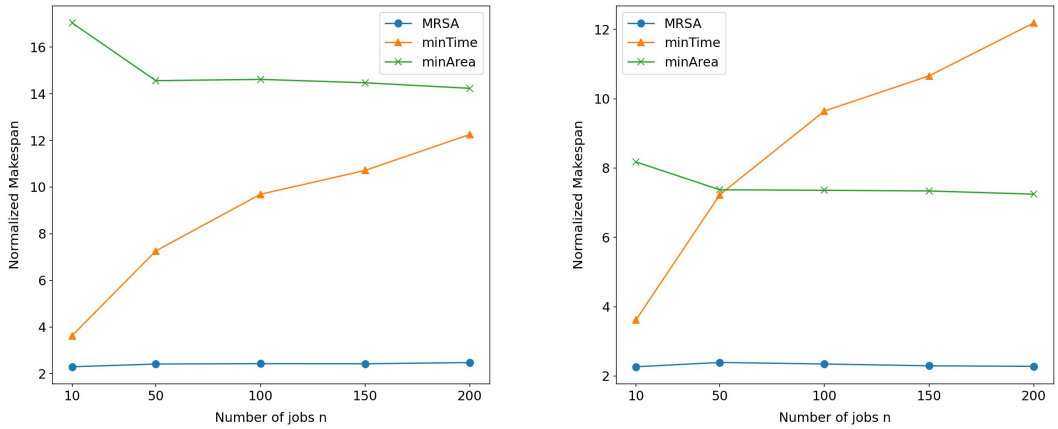
(c) Power-Sum



(d) Power-Max

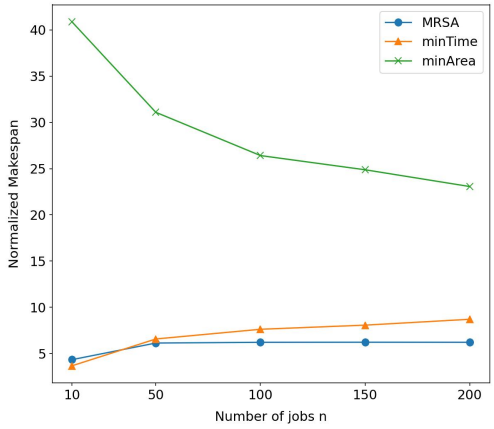
Figure 5.4: Boxplots showing the normalized makespans of the three algorithms for 50 workflows with $n = 100$ jobs, $d = 3$ resource types, and non-uniform P (up to 1024) under the four speedup models.

Impact of number of jobs n: Figure 5.5 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

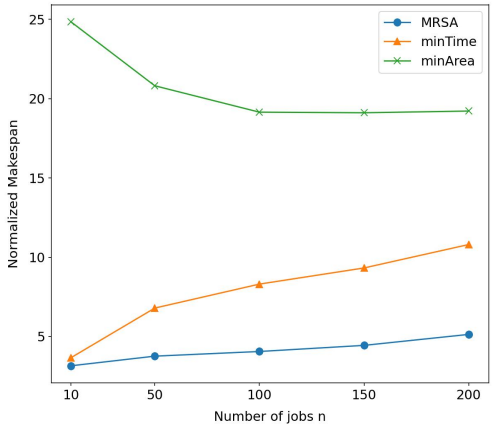


(a) Amdahl-Sum

(b) Amdahl-Max



(c) Power-Sum

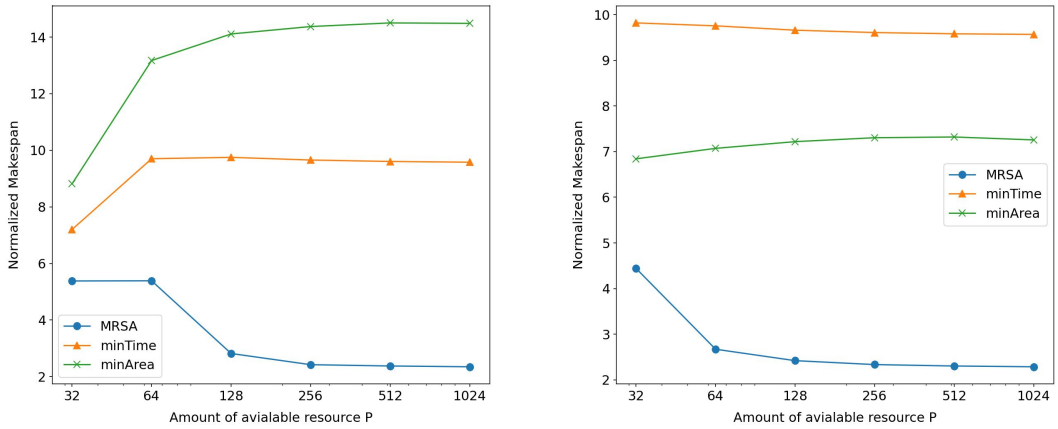


(d) Power-Max

Figure 5.5: Impact of the number of jobs n on the performance of the three algorithms under the four speedup models.

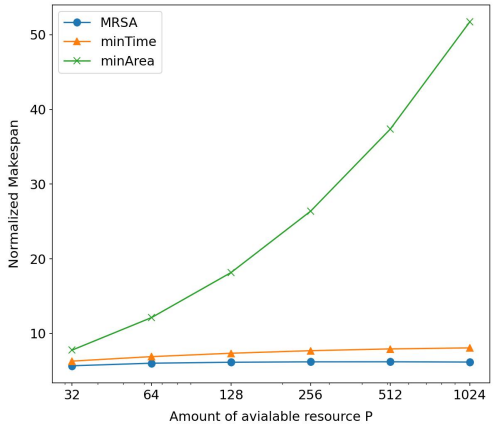
workflow.

Impact of amount of available resources P: Figure 5.6 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

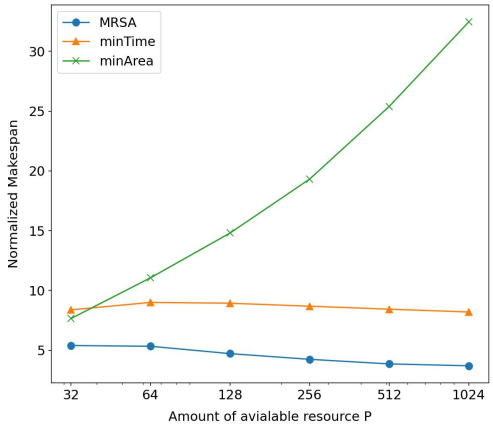


(a) Amdahl-Sum

(b) Amdahl-Max



(c) Power-Sum

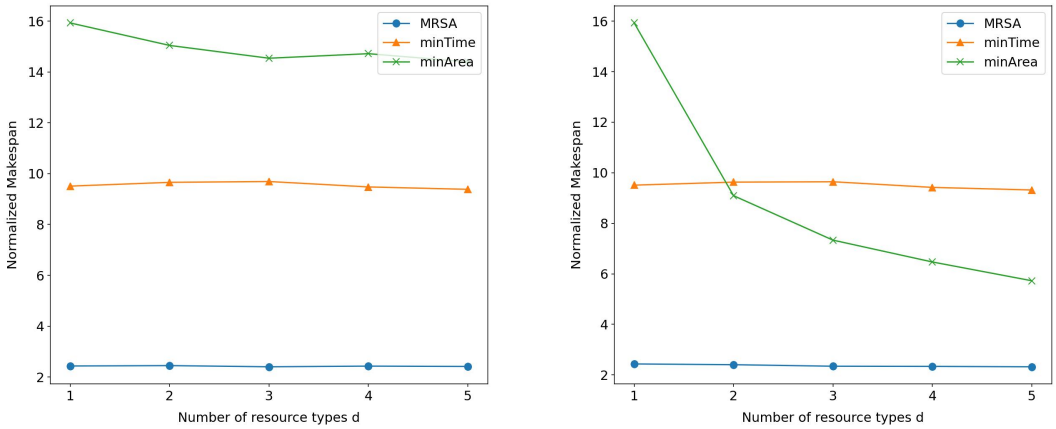


(d) Power-Max

Figure 5.6: Impact of the amount of available resources P on the performance of the three algorithms under the four speedup models.

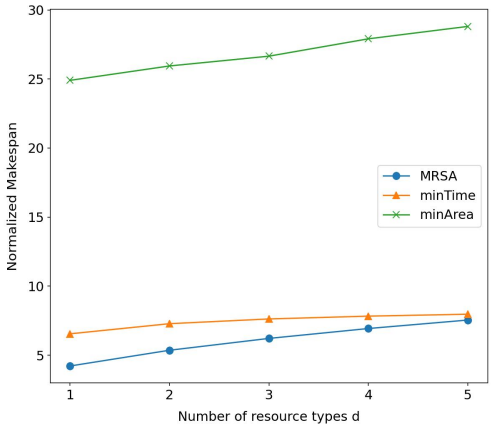
general increase in normalized makespan as P increases. This is because not all resources will be utilized in this case due to minArea's conservative resource allocation strategy.

Impact of number of resource types d: Figure 5.7 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

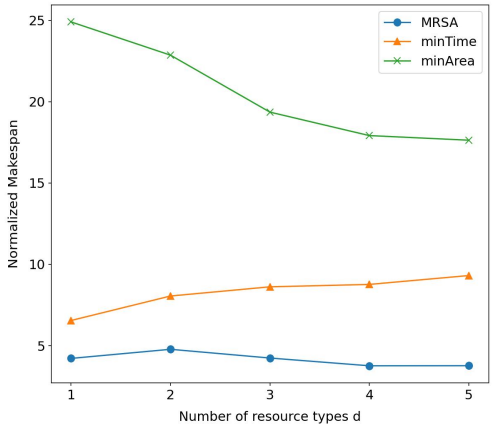


(a) Amdahl-Sum

(b) Amdahl-Max



(c) Power-Sum



(d) Power-Max

Figure 5.7: Impact of the number of resource types d on the performance of the three algorithms under the four speedup models.

models, but MRSA remains the best performer in all cases.

5.4 Impact of Graph Structure

This section evaluates the impact of workflow/graph structure on the performance of the scheduling algorithms. As described in Section 5.1, 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

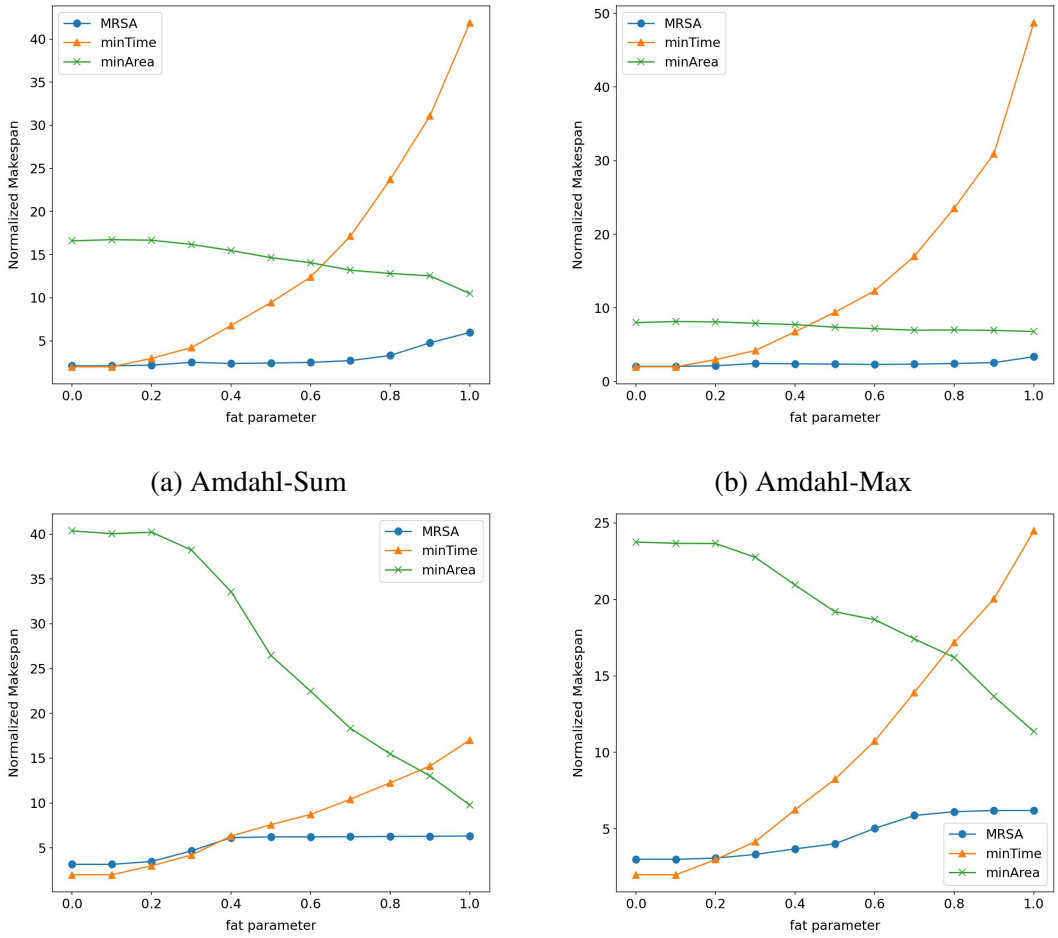
across 50 workflows are reported for all algorithms.

Impact of fat parameter: The fat parameter controls the width of a graph. Figure 5.8 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 a drastic increase in minTime's normalized makespan. On the other hand, we see a decrease in the normalized makespan for minArea. This is due to the fact that a larger graph width allows more jobs to be executed concurrently 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. Figure 5.9 shows that density barely affects the performance of MRSA, and as it increases, the normalized makespans of 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. Figure 5.10 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. Figure 5.11 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



(a) Amdahl-Sum

(b) Amdahl-Max

(c) Power-Sum

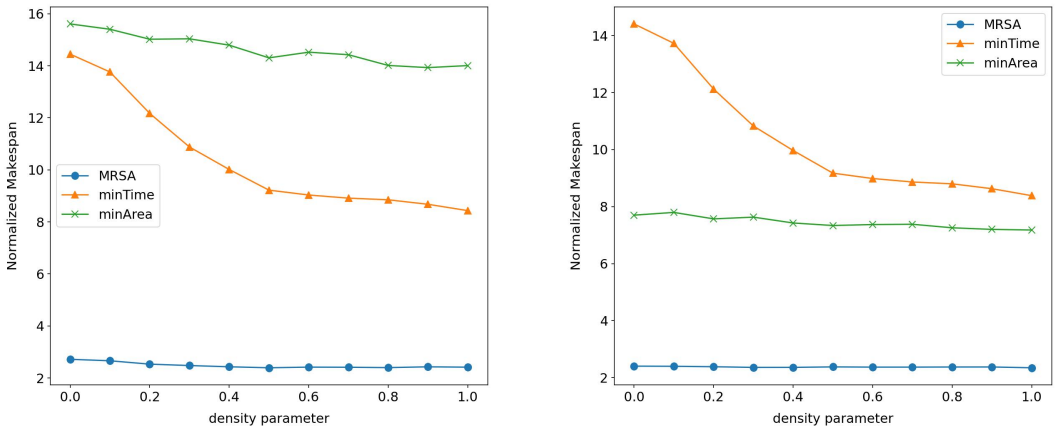
(d) Power-Max

Figure 5.8: Impact of the fat parameter in DAGGEN on the performance of the three algorithms under the four speedup models.

normalized makespan for minTime, as can be seen in the figure.

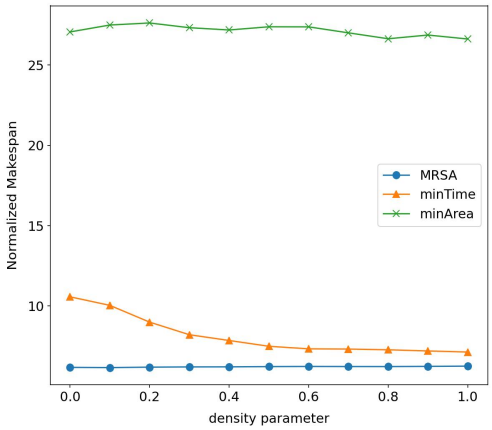
5.5 Impact of Job Speedup Functions

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 (for the Amdahl models) and the efficiency factor α_i (for 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 the experiment, we choose three different ranges (small, medium

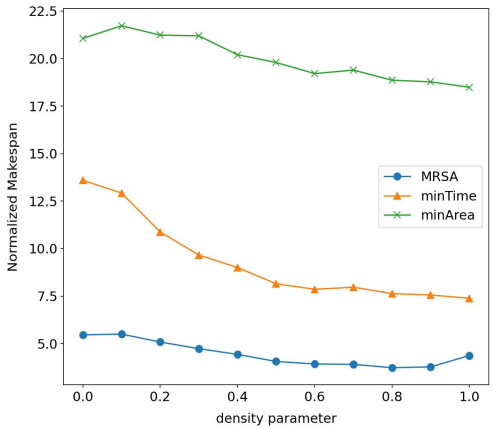


(a) Amdahl-Sum

(b) Amdahl-Max

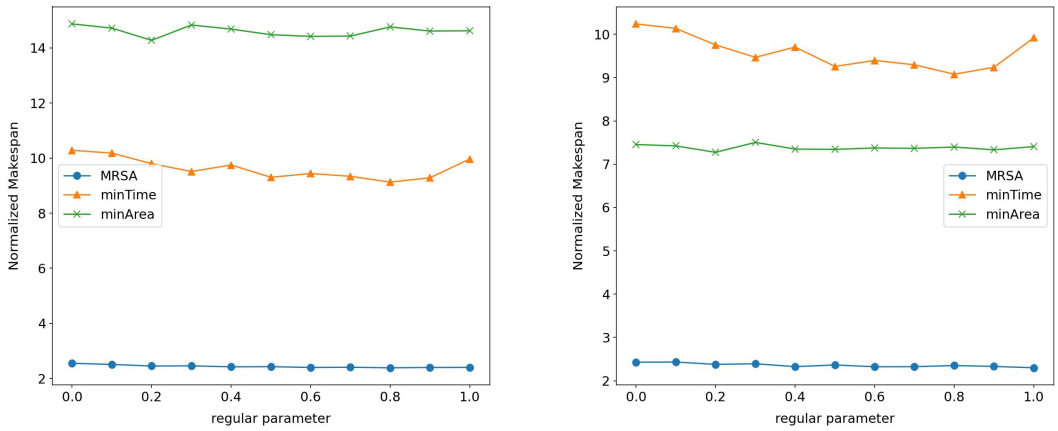


(c) Power-Sum



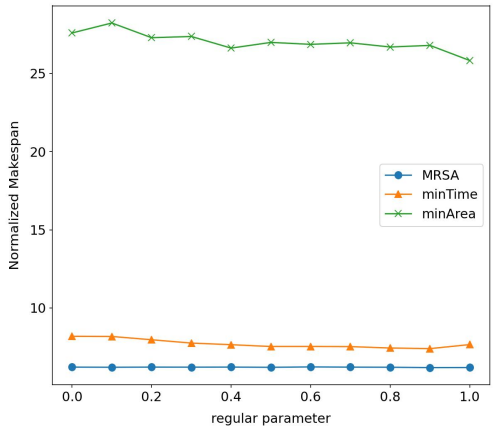
(d) Power-Max

Figure 5.9: Impact of the *density* parameter in DAGGEN on the performance of the three algorithms under the four speedup models.

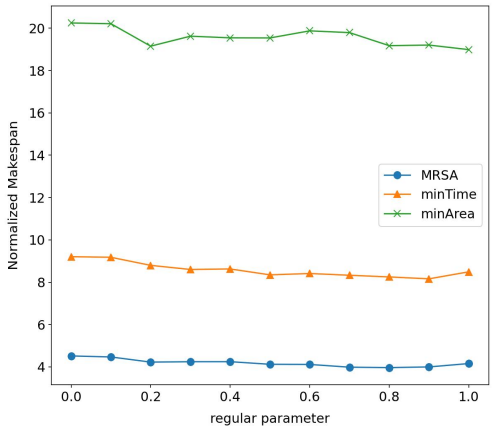


(a) Amdahl-Sum

(b) Amdahl-Max

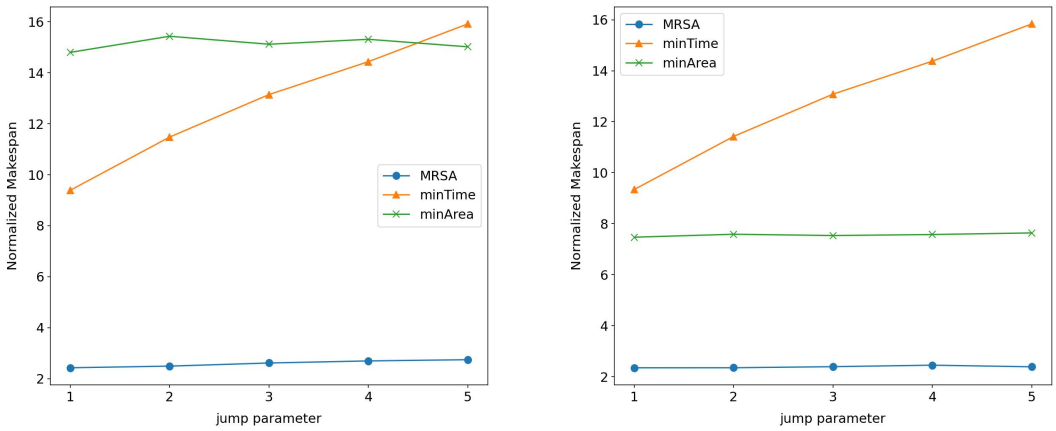


(c) Power-Sum



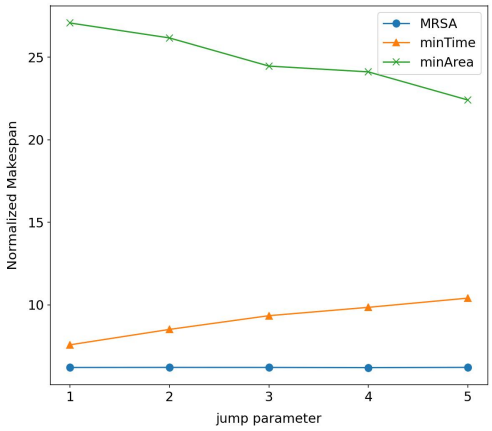
(d) Power-Max

Figure 5.10: Impact of the *regular* parameter in DAGGEN on the performance of the three algorithms under the four speedup models.

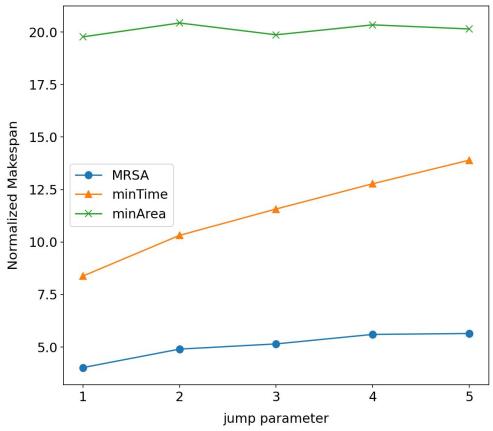


(a) Amdahl-Sum

(b) Amdahl-Max



(c) Power-Sum



(d) Power-Max

Figure 5.11: Impact of the *jump* parameter in DAGGEN on the performance of the three algorithms under the four speedup models.

and large) for setting these two parameters.

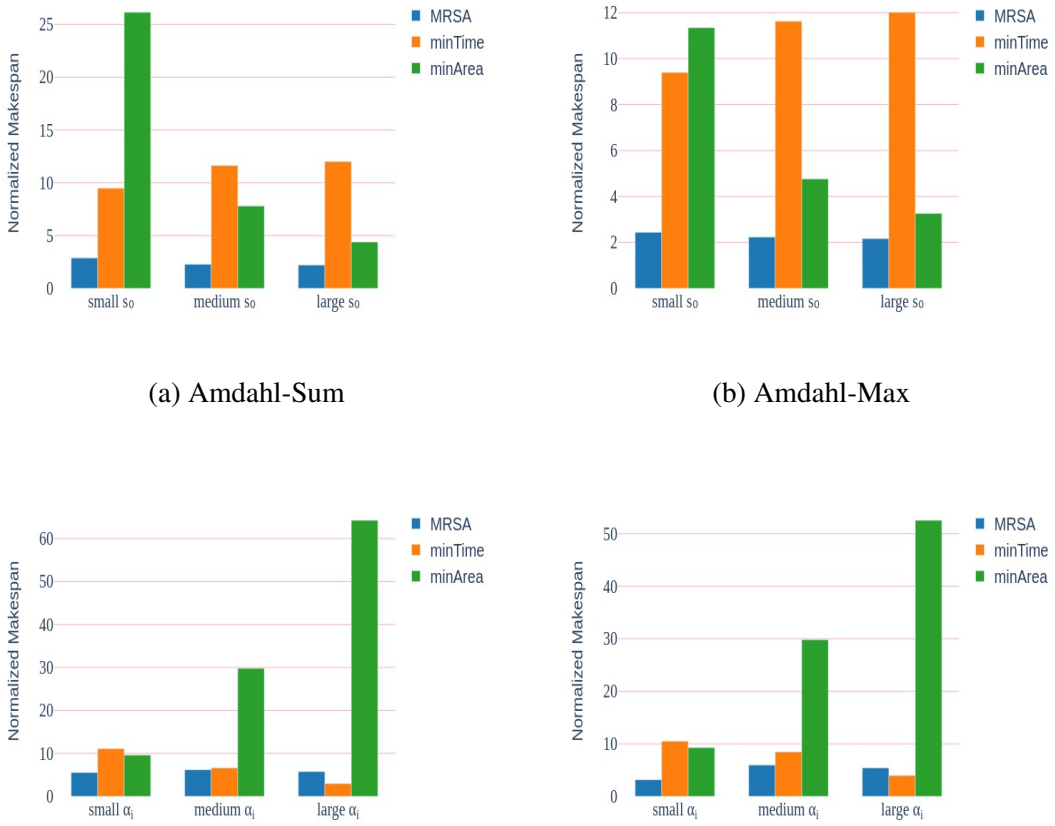
For the sequential fraction s_0, the ranges are set as follows.

- *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, the ranges are set as follows.

- *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 $(0.8, 1]$.

Figure 5.12 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 Figure 5.12(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 Figure 5.12(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.



(a) Amdahl-Sum

(b) Amdahl-Max

(c) Power-Sum

(d) Power-Max

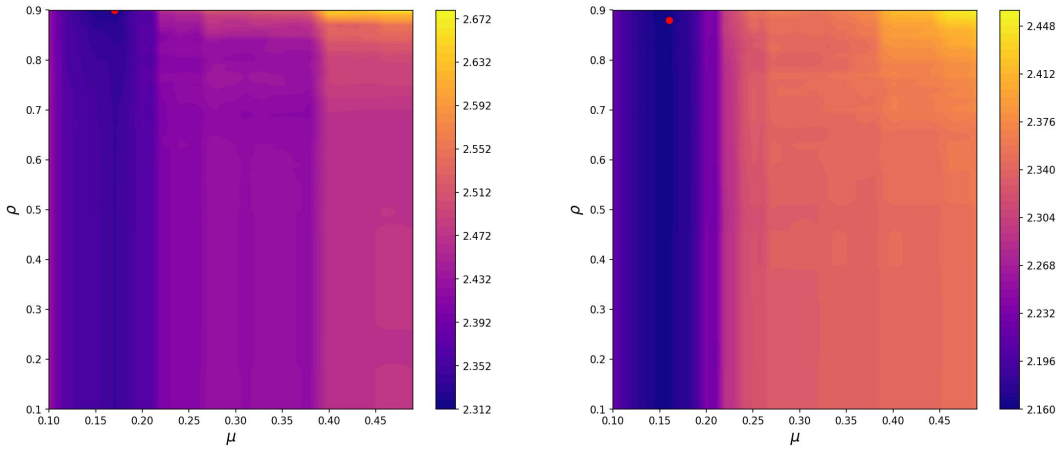
Figure 5.12: Impact of the sequential fraction s_0 (for the two Amdahl models) and the efficiency factor α_i (for the two power models) on the performance of the three algorithms.

5.6 Impact of MRSA Parameters

Two parameters μ and ρ are used in MRSA and 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 MRSA.

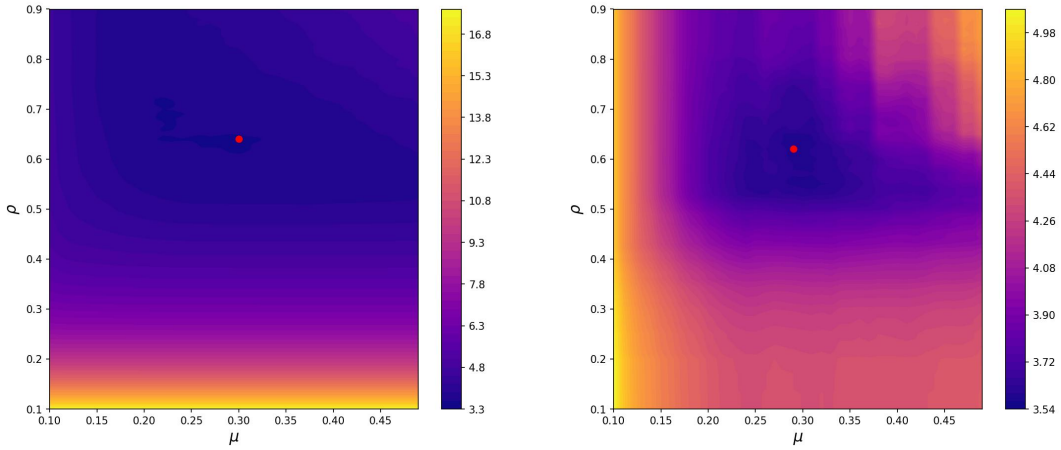
Figure 5.13 shows the normalized makespan of MRSA (averaged over 50 workflows) under the four speedup models when μ is varied from 0.1 to 0.5 and ρ is varied from 0.1 to 0.9, both with an increment of 0.01 each time. The red dot in each plot indicates the combination of μ and ρ that

gives the best average performance under the respective speedup model in our simulation. We can see that the results are not 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; and for the two power models, the best μ is around 0.3 and the best ρ is between 0.6 and 0.7. Such discrepancy is possibly due to the following reasons: (1) the derived approximation ratio is not tight; (2) the theoretical analysis assumes the worst-case scenario and thus may not reflect the average-case performance; and (3) the analysis is based on a generic job execution model and does not consider specific speedup functions. Hence, the insights gained in this experiment can be potentially explored in future work to improve the approximation results of the algorithm, by further tuning the choices of parameters μ and ρ, and by taking into account specific speedup models of the jobs.



(a) Amdahl-Sum

(b) Amdahl-Max



(c) Power-Sum

(d) Power-Max

Figure 5.13: Impact of parameters μ and ρ on the performance (average normalized makespan) of MRSA under the four speedup models.

Chapter 6

Conclusion

This thesis has performed an empirical evaluation of MRSA, a multi-resource scheduling algorithm for moldable workflows, with the objective of minimizing the overall completion time, or the makespan. Due to the NP-completeness of the scheduling problem, MRSA was originally designed as an approximation algorithm in [28] with an approximation ratio proven to be linear in the number of resource types. Our simulations conducted using workflows generated by DAGGEN and moldable jobs following different speedup models have shown that the practical performance of MRSA is actually much better than the worst-case approximation ratio predicts. Furthermore, MRSA outperforms two other heuristic algorithms under a variety of parameter settings, including different system parameters, graph structures, job speedup models, etc.

The results of this thesis have provided a nice empirical complement to the theoretical results proven in [28]. The writing of an extended paper is currently underway to combine both sets of results for journal submission in the near future.

References

[1] DAGGEN: A synthetic task graph generator. `https://github.com/frs69wq/daggen`.

[2] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS'67*, pages 483–485, 1967.

[3] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. StarPU: A unified platform for task scheduling on heterogeneous multicore architectures. *Concurr. Comput. : Pract. Exper.*, 23(2):187–198, 2011.

[4] O. Beaumont, L.-C. Canon, L. Eyraud-Dubois, G. Lucarelli, L. Marchal, C. Mommessin, B. Simon, and D. Trystram. Scheduling on two types of resources: A survey. *ACM Comput. Surv.*, 53(3), 2020.

[5] O. Beaumont, L. Eyraud-Dubois, and S. Kumar. Fast approximation algorithms for task-based runtime systems. *Concurrency and Computation: Practice and Experience*, 30(17):e4502, 2018.

[6] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, and J. J. Dongarra. PaRSEC: Exploiting heterogeneity to enhance scalability. *Computing in Science and Engg.*, 15(6):36–45, 2013.

[7] M. Caccamo, R. Pellizzoni, L. Sha, G. Yao, and H. Yun. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *RTAS*, page 55–64, 2013.

[8] P. De, E. J. Dunne, J. B. Ghosh, and C. E. Wells. Complexity of the discrete time-cost tradeoff problem for project networks. *Operations Research*, 45(2):302–306, 1997.

[9] P. De, E. James Dunne, J. B. Ghosh, and C. E. Wells. The discrete time-cost tradeoff problem revisited. *European Journal of Operational Research*, 81(2):225–238, 1995.

[10] G. Demirci, H. Hoffmann, and D. H. K. Kim. Approximation algorithms for scheduling with resource and precedence constraints. In *STACS*, 2018.

[11] J. Du and J. Y.-T. Leung. Complexity of scheduling parallel task systems. *SIAM J. Discret. Math.*, 2(4):473–487, 1989.

[12] L. Eyraud-Dubois and S. Kumar. Analysis of a list scheduling algorithm for task graphs on two types of resources. In *IPDPS*, 2020.

[13] D. G. Feitelson. Job scheduling in multiprogrammed parallel systems (extended version). *IBM Research Report RC19790(87657)*, 1997.

[14] A. Feldmann, M.-Y. Kao, J. Sgall, and S.-H. Teng. Optimal on-line scheduling of parallel jobs with dependencies. *Journal of Combinatorial Optimization*, 1(4):393–411, 1998.

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

[16] T. Gautier, X. Besseron, and L. Pigeon. KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In *PASCO*, page 15–23, 2007.

[17] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, pages 323–336, 2011.

[18] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. *SIGCOMM Comput. Commun. Rev.*, 44(4):455–466, Aug. 2014.

[19] Y. He, J. Liu, and H. Sun. Scheduling functionally heterogeneous systems with utilization balancing. In *IPDPS*, pages 1187–1198, 2011.

[20] Y. He, H. Sun, and W.-J. Hsu. Adaptive scheduling of parallel jobs on functionally heterogeneous resources. In *ICPP*, page 43, 2007.

[21] K. Jansen and H. Zhang. Scheduling malleable tasks with precedence constraints. In *SPAA*, page 86–95, 2005.

[22] K. Jansen and H. Zhang. An approximation algorithm for scheduling malleable tasks under general precedence constraints. *ACM Trans. Algorithms*, 2(3):416–434, 2006.

[23] R. Lepère, D. Trystram, and G. J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. *Int. J. Found. Comput. Sci.*, 13(4):613–627, 2002.

[24] W. Ludwig and P. Tiwari. Scheduling malleable and nonmalleable parallel tasks. In *SODA*, pages 167–176, 1994.

[25] G. Mounié, C. Rapine, and D. Trystram. A 3/2-approximation algorithm for scheduling independent monotonic malleable tasks. *SIAM J. Comput.*, 37(2):401–412, 2007.

[26] M. Niemeier and A. Wiese. Scheduling with an orthogonal resource constraint. In *WAOA*, pages 242–256, 2012.

[27] M. NoroozOliaee, B. Hamdaoui, M. Guizani, and M. B. Ghorbel. Online multi-resource scheduling for minimum task completion time in cloud servers. In *INFOCOM Workshops*, 2014.

[28] L. Perotin, H. Sun, and P. Raghavan. Multi-resource list scheduling of moldable parallel jobs under precedence constraints. In *50th International Conference on Parallel Processing*, 2021.

[29] G. N. S. Prasanna and B. R. Musicus. Generalized multiprocessor scheduling and applications to matrix computations. *IEEE Trans. Parallel Distrib. Syst.*, 7(6):650—664, 1996.

[30] D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. 23(3):617–632, 1994.

[31] M. Skutella. Approximation algorithms for the discrete time-cost tradeoff problem. *Math. Oper. Res.*, 23(4):909–929, 1998.

[32] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y.-C. Liu. Knights Landing: Second-generation Intel Xeon Phi product. *IEEE Micro*, 36(2):34–46, 2016.

[33] H. Sun, R. Elghazi, A. Gainaru, G. Aupy, and P. Raghavan. Scheduling parallel tasks under multiple resources: List scheduling vs. pack scheduling. In *IPDPS*, pages 194–203, 2018.

[34] J. Turek, J. L. Wolf, and P. S. Yu. Approximate algorithms scheduling parallelizable tasks. In *SPAA*, 1992.

[35] M. Xu, L. T. X. Phan, X. Phan, H. Choi, and I. Lee. vCAT: Dynamic cache management using CAT virtualization. In *RTAS*, 2017.