Learning-Augmented Energy-Aware List Scheduling for Precedence-Constrained Tasks

YU SU, California Institute of Technology, Pasadena, United States
VIVEK ANAND, Georgia Institute of Technology, Atlanta, United States
JANNIE YU, California Institute of Technology, Pasadena, United States
JIAN TAN, Alibaba Inc., Sunnyvale, United States
ADAM WIERMAN, California Institute of Technology, Pasadena, United States

We study the problem of scheduling precedence-constrained tasks to balance between performance and energy consumption. We consider a system with multiple servers capable of speed scaling and seek to schedule precedence-constrained tasks to minimize a linear combination of performance and energy consumption. Inspired by the single-server setting, we propose the concept of *pseudo-size* for individual tasks, which is a measure of the externalities of a task in the precedence graph and is learned from historical workload data. We then propose a two-stage scheduling framework that uses a learned pseudo-size approximation and achieves a provable approximation bound on the linear combination of performance and energy consumption for both makespan and total weighted completion time, where the quality of the bound depends on the approximation quality of pseudo-sizes. We show experimentally that learning-based approaches consistently perform near optimally.

CCS Concepts: • **Theory of computation → Scheduling algorithms**; • **Computer systems organization → Cloud computing;**

Additional Key Words and Phrases: Energy-Aware Scheduling, Precedence-Constrained Tasks, Learning-Augmented Algorithms

ACM Reference Format:
Yu Su, Vivek Anand, Jannie Yu, Jian Tan, and Adam Wierman. 2024. Learning-Augmented Energy-Aware List Scheduling for Precedence-Constrained Tasks. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 9, 4, Article 13 (September 2024), 24 pages. https://doi.org/10.1145/3680278

1 Introduction

This article seeks to develop energy-aware scheduling policies for precedence-constrained tasks that arise in modern machine learning platforms. The problem of how to optimally schedule a job made up of tasks with precedence constraints has been studied for decades. The initial work on this scheduling problem arose in the context of scheduling jobs on multi-processor systems [17].

Funding for the publications listed above has come in large part from the NSF through CNS-2146814, CPS-2136197, CNS-2106403, NGSDI-2105648, AitF-1637598, CNS-1518941 and supported Jannie Yu, Yu Su and Adam Wierman.
Authors' Contact Information: Yu Su, California Institute of Technology, Pasadena, California, United States; e-mail: suyu@caltech.edu; Vivek Anand, Georgia Institute of Technology, Atlanta, Georgia, United States; e-mail: vivekanand@gatech.edu; Jannie Yu, California Institute of Technology, Pasadena, California, United States; e-mail: jannie@caltech.edu; Jian Tan, Alibaba Inc., Sunnyvale, California, United States; e-mail: j.tan@alibaba-inc.com; Adam Wierman, California Institute of Technology, Pasadena, California, United States; e-mail: adamw@caltech.edu.

ACM 2376-3639/2024/09-ART13
https://doi.org/10.1145/3680278

Today this problem attracts attention due to the prominence of large-scale, general-purpose machine learning platforms, e.g., Google's TensorFlow [1], Facebook's PyTorch [49], and Microsoft's **Azure Machine Learning (AzureML)** [13]. Machine learning jobs on the cloud are often expressed as **directed acyclic graphs (DAGs)** of precedence-constrained tasks, and how these jobs are scheduled to run on clusters of machines on the cloud is very crucial to the performance of the system [1]. Another timely example of scheduling precedence-constrained tasks is the parallelization of training and evaluation of large complex neural networks in heterogeneous clusters that consist of **Central processing Units (CPUs)**, **Graphical processing Units (GPUs)**, **Tensor Processing Units (TPUs)**, and so forth. This *device placement* problem has attracted considerable attention in recent years [24, 28, 46, 66].

Traditionally, computational efficiency has been the only focus of works studying how to schedule precedence-constrained tasks; e.g., the goal is to complete the tasks as soon as possible given a fixed set of heterogeneous machines. The most common metric in the literature is total weighted completion time, i.e., a weighted average of completion time of tasks. The mean response time is a special case of total weighted completion time (via assigning equal weights to all the tasks), as is the makespan (via adding a dummy node of weight 1 as the final task with all other tasks assigned to weight 0). For these performance measures, significant progress has been made in recent years. New results have emerged providing policies with poly-logarithmic approximation ratios in increasingly general settings, including settings focused on makespan and total weighted completion time, settings with heterogeneous related machines, and settings with uniform and machine-dependent communication times [20, 21, 40–42, 59].

However, the increasing scale of machine learning jobs has brought questions about the energy usage of such jobs to the forefront. Today, the emissions of training an **Artificial Intelligence (AI)** model can be as high as five times the lifetime emission of a car [58]. The computation required for deep learning has been doubling every 3.4 months, resulting in a 300,000x increase from 2012 to 2018 [4, 54]. Indeed, the energy cost for an individual data center is on the order of millions of dollars, and it has become a significant portion of operating costs for cloud platforms [30]. However, there is an inherent conflict between boosting performance and reducing energy consumption; i.e., a larger power budget, in general, allows a higher performance in practice. Thus, it is urgent to study how we can efficiently schedule machine learning jobs with both performance and energy consumption in mind. Balancing these performance measures and energy usage is crucial to the industry as well as societal goals of making cloud computing carbon neutral [26].

There has been considerable progress toward understanding how to schedule to balance performance and energy measures in single- and multi-server settings without dependencies between tasks [3, 14, 63, 64]. A focus within this line of work is on the question of co-designing the scheduling of tasks and speed scaling of servers. While there has been progress in studying speed scaling in simple scheduling problems, the question of how to balance energy usage with traditional performance metrics, such as total weighted completion time, in more complex settings where there are dependencies among tasks is a challenging open question. In fact, scheduling precedence-constrained tasks is NP-hard even when ignoring power consumption; i.e., the goals of both partitioning the jobs across machines and scheduling the jobs among a group of machines are NP-hard [37]. Further, speed scaling adds considerable difficulty to the question of how to schedule tasks optimally to balance energy and performance. When speed scaling is not considered, the relaxed version of the scheduling problem can be formulated as a **mixed integer linear program (MILP)**. Thus, there are many off-the-shelf solvers that work well for these problems on a relatively small scale in practice. In the presence of speed scaling, however, solving for the optimal schedule even for small-scale problems becomes even more complex and computationally expensive in practice because the optimization is no longer linear.

Given the hardness of the problem, a natural question is: *can we design a joint scheduler and speed scaling policy for precedence-constrained tasks that is provably near-optimal for a linear combination of performance and energy consumption?*

Contributions. In this article, we present the first learning-augmented algorithm for scheduling precedence-constrained tasks on multiple machines to balance performance and energy. Our results provide provable guarantees for both makespan and total weighted completion time.

Our algorithm, Learning-augmented Energy-aware List Scheduling (Algorithm 2), uses list scheduling in concert with a learned estimate of the so-called "pseudo-sizes" of tasks. Often the biggest challenge in designing a learning-augmented algorithm is to determine what quantity to learn. Identifying the concept of pseudo-size as what to learn is a key novel idea underlying our algorithm. We introduce the pseudo-size of a task as a measure of the externalities of the task on other tasks that have yet to complete (see Section 3.1 for details). Intuitively, if many other tasks have precedence constraints that depend on the current task, then the externalities of the task are high and it is worth investing more energy in order to increase the speed and finish the task quickly.

Our main results provide performance guarantees that depend on the quality of the learned approximation of the pseudo-size (Theorem 4.1 for total weighted completion time and Theorem 4.2 for makespan). If pseudo-size estimates are perfect, these results match the best results possible, and they further provide bounds on the degradation of performance as pseudo-size estimate quality drops.

Finally, we present simple and effective approaches for learning pseudo-size approximations and demonstrate the performance of these approaches in real-world workloads of jobs with precedence-constrained tasks. Our workloads cover a wide variety of compute-intensive and data-intensive workloads from bioinformatics, astronomy, seismology, and so forth. Our results show that gradient descent-based learning is near optimal across a wide variety of workloads, never performing more than 7% worse than our upper bound on the optimal cost.[1] We also highlight that a less computationally intensive approach based on linear regression can perform well in many scenarios and that, when precedence constraints have simple structures, even a deterministic non-learning-based estimation of pseudo-sizes can be effective.

Related Literature. In recent years, due to the successful deployment of machine learning algorithms under widely varying scenarios, the design and optimization of large-scale machine learning platforms attracts extensive attention from both academic and industry communities. One of the fundamental challenges is to balance system performance and energy usage while scheduling machine learning jobs with precedence constraints.

Significant progress has been recently made toward the goal of maximizing performance while scheduling precedence-constrained tasks if energy concerns are ignored. Under the related machines model, i.e., $Q|prec|\sum \omega_j C_j$, a Speed-based List Scheduling algorithm was proposed to obtain an $O(\log m)$-approximation [16]. Subsequently, an improvement to $O(\log m/\log\log m)$ was made in 2017 by [40] for both objectives: makespan and total weighted completion time. If communication delays are assumed to be fixed (uniform), two groups of researchers independently made progress toward a multiplicative approximation algorithm with logarithmic guarantees by adopting different approaches [20, 21, 42]. When it comes to incorporating non-uniform communication delays for the first time, i.e., $Q|prec, c_{i,j}|\sum \omega_j C_j$, **Generalized Earliest Time First (GETF)** was proposed in [59] to achieve a worst-case bound for both makespan and total weighted completion time, and it reduces to the state-of-the-art results in the case of zero communication delays.

[1] Computing the true optimal cost is intractable for the scale of real-world jobs. See Section 6.1.2.

There has been a flurry of work studying power management if precedence constraints are ignored, including settings with sleep states [19, 23, 50], settings that consider speed scaling, settings with single servers [6–8, 12, 64], and settings with multiple servers [3, 14, 27]. A comparison of different energy conservation schemes has also received considerable attention as well [29, 63]. Further, diverse performance measures have been considered in the literature, including deadline feasibility [33, 39, 64], flow time [2, 10, 57], and so forth. We refer to [32] for a comprehensive survey of related scheduling problems.

When both performance and energy goals are considered for tasks with precedence constraints, much less is known. The closest work to ours is [51]. They consider the problem of scheduling precedence-constrained tasks to minimize the makespan subject to an energy constraint and obtain a poly-logarithmic approximation algorithm by reducing the problem to the problem $Q|prec|C_{max}$. However, their technique does not apply to our setting for two reasons. First, we consider a general objective, which is a linear combination of performance and energy consumption, instead of focusing on the constrained budget problem. Second, total weighted completion time is a more general choice for the performance measure, of which makespan is a special case. There are a variety of papers that build on [51]; e.g., [5] solves the same problem as [51] but improves the competitive ratio. Other papers include [35] and [55], which give empirical heuristics for the objective of makespan plus energy, without providing theoretical guarantees. There are a variety of other such works that design heuristics to balance both performance and energy, e.g., [11, 35, 36, 43–45, 56, 60–62, 65]. However, as we focus on algorithms with provable guarantees, we do not dive into the details of these heuristics. Thus, at this point, it remains unknown if it is possible to design a scheduler for precedence-constrained tasks that provably minimizes the combination of performance and energy measures in general settings.

Our work takes a learning-augmented approach, using machine-learned predictions to overcome the challenges associated with scheduling precedence-constrained jobs to minimize performance and energy. The area of learning-augmented algorithms has received considerable attention in recent years, as machine-learned advice is an increasingly powerful tool for algorithms to exploit. The field began with a focus on algorithms that can use predictions to improve running times through techniques such as warm starts or predicting the internal state of algorithms. See, e.g., [22, 25, 31] and the references therein. Our work falls into this category and is the first learning-augmented algorithm designed to balance energy and performance for precedence-constrained tasks. Another independent line of work considers future predictions in online decision-making tasks. Examples of this approach include [15, 47, 52, 53] and the references therein. Within this line of work, a related recent paper [34] studies the value of future predictions in the online problem of scheduling precedence-constrained jobs to minimize completion time (without consideration of energy). This line of work differs from how we use machine-learned advice in this article. We focus on predictions of the internal state of the algorithm to achieve a good approximation efficiently. In this context, our goal is to (1) understand the right parameters to predict in order to design a robust algorithm that performs well when predictions are accurate but does not suffer badly if predictions are poor; (2) provide bounds on the performance, depending on on the error of the predictions; and (3) show that the selected parameters can be learned from historical data.

2 Model

We study the problem of scheduling a job made of a set $\mathcal{V}$ of n tasks on a system consisting of a set $\mathcal{M}$ of m machines. The tasks form a DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in which each node j represents a task and an edge (j', j) between task j and task j' represents a precedence constraint. We interchangeably use node or task, as convenient. Precedence constraints are denoted by a partial order $\prec$ between two nodes of any edge, where $j' \prec j$ means that task j can only be scheduled after task j' completes.

Let p_j represent the processing size of task j. If task j is assigned to run at a speed s_j on machine i, then it would take $\frac{p_j}{s_j}$ time units to run on machine i.

For simplicity, we assume that the DAG is connected. This is without loss of generality because, otherwise, the DAG can be viewed as multiple DAGs and the same results can be applied to each individual connected component. Consequently, our results trivially apply to the case of multiple jobs. Additionally, our model assumes that each machine can process at most one task at a time; i.e., there is no *time-sharing*, and the machines are assumed to be *non-preemptive*—i.e., once a task starts on a machine, the scheduler must wait for the task to complete before scheduling any new task to this machine. This is a natural assumption in many settings, as interrupting a task and transferring it to another machine can cause significant processing overhead and communication delays due to data locality. These assumptions are standard in the related literature, e.g., [20, 51, 59].

Performance and Energy Metrics. The objective function we consider is $T + \lambda E$, a linear combination of a performance measure T and an energy/power usage E. The system operator can emphasize either performance or energy as desired via a choice of weight λ. For this work, we adopt total weighted completion time as the performance measure, though our results apply to both makespan and mean response time as well. For the energy metric E, our focus is on total energy usage, which is the sum of energy usage of all tasks in the DAG, i.e., $E = \sum_{j \in \mathcal{V}} e(j)$.

Speed Scaling. The scheduler we consider has the ability to scale speed servers in order to trade off performance and energy. In our model, a server chooses speed s_j for task j. For a task running at speed s_j, its energy consumption $e(j)$ is modeled as the product of the instantaneous power $f(s_j)$ and running time t_j of that task, i.e., $e(j) = f(s_j) \cdot t_j$. A common form of the power function in the literature is a polynomial, i.e., $f(s) = s^\alpha$, where $\alpha > 1$ [7, 9, 10]. A quadratic form is most common, so we focus on that in this work, though our main results apply more generally. Note that for any convex choice of instantaneous power function, given any optimal schedule, the server always runs at a constant speed during the execution of a single task; otherwise it would be possible to adopt the average speed for running the task without any sacrifice on performance measure but potentially conserve more energy.

The Optimization Problem. We can now formally define the joint scheduling and speed scaling problem via an optimization formulation. Let $x_{i,j}$ be a binary decision variable that indicates whether task j is assigned to machine i, i.e.,

$$x_{i,j} = \begin{cases} 1, & \text{if task } j \text{ runs on machine } i, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Let C_j denote the completion time of task j and s_j denote the assigned speed of task j. The scheduling problem (with total weighted completion time as the performance measure) can be formulated as follows:

$$\min_{x_{i,j}, C_j, s_j, T, E} \quad T + \lambda E$$

$$x_{i,j} \in \{0, 1\} \qquad \forall i, j \tag{2a}$$

$$\sum_i x_{i,j} = 1 \qquad \forall j \tag{2b}$$

$$C_{j'} + \frac{p_j}{s_j} \le C_j \qquad j' \prec j \tag{2c}$$

$$\sum_j \omega_j C_j \le T \qquad \forall j \tag{2d}$$

$$\sum_j f(s_j)\frac{p_j}{s_j} = E \tag{2e}$$

$$\sum_i x_{i,j}x_{i,j'} = q_{j,j'} \qquad \forall j, j' \text{ if } j! = j' \tag{2f}$$

$$C_{j'} - C_j \geq \frac{p_{j'}}{s_{j'}} \text{ or } C_j - C_{j'} \geq \frac{p_j}{s_j} \qquad \text{if } q_{j,j'} = 1\ . \tag{2g}$$

The optimal value of this problem is denoted as OPT. Constraint (2b) requires every task to be scheduled on some machine. Constraint (2c) guarantees that for any successor predecessor pair, the successor task will not start until the predecessor completes. In Constraint (2d), T represents the total weighted completion time, and it can be reduced to other performance measures by assigning appropriate weights, such as makespan and mean response time. In Constraint (2e), E is the sum of multiplication of power function per unit time and running time. Constraint (2f) guarantees that $q_{j,j'} = 1$ if task j and task j' are assigned to the same machine. Any machine should not process more than one task at a time, as in Constraint (2g). By addition of an auxiliary binary variable $b_{j,j'}$, we can rewrite Constraint (2g) as follows:

$$q_{j,j'}\left(C_j - C_{j'} + \frac{p_{j'}}{s_{j'}}\right) \leq b_{j,j'}\left(T - C_{j'} + \frac{p_{j'}}{s_{j'}}\right) \forall j, j' \tag{3a}$$

$$b_{j,j'}C_{j'} \leq \left(C_j - \frac{p_j}{s_j}\right) q_{j,j'} \qquad \forall j, j' \tag{3b}$$

$$b_{j,j'} \leq q_{j,j'} \qquad \forall j, j' \tag{3c}$$

$$b_{j,j'} \in \{0, 1\} \qquad \forall j, j'. \tag{3d}$$

When task j and task j' are assigned to run on the same machine, i.e., $q_{j,j'} = 1$, the auxiliary variable represents the ordering of these two tasks, i.e., $b_{j,j'} = 0$ if $j \prec j'$.

We emphasize that there are other possible formulations of the optimization problem. For example, we could construct a different formulation with a time-indexed program, though it would remain nonlinear due to speed constraints. Thus, we adopt the above formulation for simplicity. Because solving for precedence-constrained tasks on multiple servers is NP-hard, we focus on approximation algorithms in this article.

3 Algorithm Design

Inspired by the single-server case, we introduce the notion of *pseudo-size* to quantify the importance of tasks in the DAG. Intuitively, a task that many other tasks depend on via precedence constraints should be prioritized to run at a fast speed, while a task that few other tasks depend on should be set to run at a slow speed. This intuition is simple, but understanding how many other tasks depend on a given task is complex since it depends on both the task's children in the DAG and the "width" of the DAG at the task's position and the sizes of the various tasks in the DAG. Our approximation algorithm uses a pseudo-size approximation to capture these factors in combination with an approximation algorithm for the case of identical machines without energy concerns, e.g., [48], to produce a schedule that balances between performance and energy goals. The pseudo-size of tasks that we use depends on various features of the precedence graph, such as degree of nodes, number of children, and so forth, as well as the given number of machines. In practice, experts can extract these features and feed them to an off-the-shelf learning algorithm to obtain an approximation, which is then used by the approximation algorithm.

The introduction of pseudo-size is critical in two ways. First, conditional on an approximation of pseudo-size, we are able to reduce the general problem, for which it is hard to find an optimal

solution, to a simpler, tractable scheduling problem on the case of identical machines. This not only mitigates the required computation but also makes it possible to compute a theoretical bound performance of the final schedule. Second, since the concept of pseudo-size stems from the single-server scenario, it comes with an intuitive interpretation in the physical world, which is not a given in the world of learning algorithms. In practice, the pseudo-size of a task quantifies magnitude of externalities it has on other tasks in the graph.

In the remainder of the article, we first formally define the notion of *pseudo-size* and propose a family of approximation scheduling algorithms based on the pseudo-size approximation. We then prove bounds on the algorithms in terms of the quality of the pseudo-size estimates in Section 4 before discussing the task of estimating the pseudo-size in Sections 6.2.4 and 6.2.5.

3.1 Key Idea: Pseudo-size

Understanding the Single-server Case. We start by diving into the single-server problem. As shown in [63], the optimal speed for running a task without precedence constraints is proportional to the square root of the number of other tasks that are waiting for the task to complete when the power cost function is quadratic. In general, if the power cost function $P(s)$ is proportional to s^{α}, then the optimal speed is proportional to $n^{1/\alpha}$, where n is the number of waiting tasks [63]. Intuitively, the same effect should hold true for one server even if there are precedence constraints among the tasks. The characterization of optimal speeds in the one-server case is summarized as follows.

Example 1. Consider n tasks with dependency to be scheduled to run on a single machine, i.e., optimization problem (2) with $m = 1$. For any given feasible ordering in which tasks are indexed with respect to the given ordering, the optimal speed for running task j is

$$\sqrt{(n-j+1)\cdot\frac{\omega_j}{\lambda}},$$

where $(n - j + 1)$ is the number of tasks waiting for its completion.

Proof. For any feasible ordering of tasks on the machines, we assume that tasks are indexed with respect to the given ordering, i.e., $1 \prec 2 \prec \cdots \prec n$. Thus, the objective function can be simplified to

$$\left(\frac{n\cdot p_1}{s_1}+\lambda\cdot p_1\cdot s_1\right)+\left(\frac{(n-1)\cdot p_2}{s_2}+\lambda\cdot p_2\cdot s_2\right)+\cdots+\left(\frac{p_n}{s_n}+\lambda\cdot p_n\cdot s_n\right).$$

Clearly, the objective is minimized only when speed of task j is set to be equal to $\sqrt{(n-j+1)/\lambda}$. For task j, $(n - j + 1)$ is the number of tasks (including the task itself) that are depending on its completion. □

For a large λ the above highlights an emphasis on the total energy consumption, in which case the optimal speeds tend to have a smaller magnitude, and vice versa. When λ is chosen to be 1, then total weighted completion time of tasks is equal to the sum of energy consumption in any given optimal schedule; i.e., equal budgets are allocated to performance measure and energy consumption for any optimal schedule. In this case, the optimal speed of a task is exactly the square root of the number of dependent tasks provided that the weight ω_j of task j is 1. Via a choice of λ, the system designer can adjust the budget for performance and energy consumption as desired.

Defining the Pseudo-size. The pseudo-size of a task is a measure of externalities of the task with respect to its running speed. If a task has many other tasks waiting for its completion, then the pseudo-size is large and the scheduler should prioritize the task by assigning a fast running speed. This parallels the term $(n - j + 1)$ in the single-server case.

Definition 3.1. For a given DAG $\mathcal{G}$ to be scheduled on a set of m machines, the pseudo-size β_j of task j is defined as the scaled square of the optimal speed s_j^* for that task multiplied by the scaling parameter γ_j, i.e.,

$$\beta_j = \gamma_j \cdot (s_j^*)^2, \tag{4}$$

where $\gamma_j \triangleq \frac{\lambda}{\omega_j}$ is a constant scalar introduced by system parameters.

Note that in the single-server case, the optimal speed (s^*) and the pseudo-size (β^*) are indeed the same. The concept of pseudo-size is inspired by the single-server setting but extends the idea in an intuitive way to multi-server environments, where it is no longer the optimal solution. By introducing pseudo-size, we aim to facilitate the analysis and offer an intuitive way to characterize the magnitude of externalities that a task has on other tasks waiting for its completion, which can be beneficial for designing new scheduling algorithms and understanding their performance tradeoffs.

The scaling parameter γ depends on the choice of performance measure as well as the weight λ. Note that the optimal speed also depends on the number of machines. For any relatively large-scale problem, the optimal speed cannot be determined due to the computational complexity. As a result, an approximation of pseudo-size becomes significant for reducing the runtime. Our approach in this article is to use machine-learned advice to approximate this pseudo-size and then use the pseudo-size in the context of a scheduling optimization problem to jointly determine the speeds and scheduling order for the tasks, thus using machine-learned advice to significantly reduce the computational complexity of finding a near-optimal job speed and order schedule.

3.2 Learning-augmented Energy-aware List Scheduling

Using the concept of pseudo-size, we propose a novel algorithm, *Learning-augmented Energy-aware List Scheduling*, for both makespan and total weighted completion time. The concept of task pseudo-size offers a new and intuitive perspective to learn the optimal speeds via an approximation of pseudo-size. The data-driven approach enables us to adapt to different scenarios quickly, e.g., different structures of DAGs, heterogeneous task sizes and number of machines, and so forth. Additionally, as a two-stage algorithm, Learning-augmented Energy-aware List Scheduling enables us to separate the evaluation of these two stages and combine them together to form a worst-case bound under certain conditions.

Our algorithm is summarized in Algorithm 1. The algorithm has two stages. During the first stage, we take advantage of an off-the-shelf data-driven algorithm to learn an approximation of task pseudo-size from workload data. The learned model will vary depending on choice of different performance measures, DAG structure in the application of interest, and so forth.

During the second stage, we exploit the learned pseudo-size and derive the associated speed for every single task in the DAG. Given the running speeds, we are able to transform the problem into the case of identical machines and adopt any appropriate listing scheduling to compute a final schedule. Note that the priority list adopted for makespan and total weighted completion time must be different in order to achieve good theoretical guarantees. In the case of makespan, any greedy list scheduling algorithm can be employed; e.g., Largest Task First, Higher Level First, Precedence Tasks First, and more are examined in detail in [38]. In contrast, in the case of total

ALGORITHM 1: Learning-augmented Energy-aware List Scheduling

INPUT: DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and number of machines m.
OUTPUT: schedule $\mathcal{S}$ and speed scaling policy $\{s_j\}$.

First Stage
1: Approximate task pseudo-size $\{\beta_j\}$ via learning.

Second Stage
2: Derive the associated running speed $\{s_j\}$.
3: Transform the initial problem into identical machine case by assigning a new task size $p'_j \leftarrow p_j/s_j$ for every task.
4: Apply list scheduling. ▷ LS varies depending on choice of C_{max} and $\sum \omega_j C_j$.

weighted completion time, list scheduling has to be further refined. We complete the algorithm description by describing how to refine list scheduling for total weighted completion time in the following.

Total Weighted Completion Time. The full details of list scheduling in the case of total weighted completion time are summarized in Algorithm 2. Given a pseudo-size approximation $\{\beta_j\}$, we transform the problem into a problem over identical machines problem by assigning a new task size for task j:

$$p'_j \leftarrow p_j \cdot \sqrt{\gamma_j/\beta_j}. \tag{5}$$

The pseudo-size of a task quantifies the magnitude of externalities that it has on other tasks waiting for its completion. Equation (5) scales task sizes to account for externalities. After the transformation using the pseudo-size approximation, the list scheduling adopted for total weighted completion time is built upon a **linear program (LP)** [48] and is essentially the identical machines problem.

$$\min_{C_j} \sum_{j\in\mathcal{V}} \omega_j C_j$$

$$C_j \geq C_{j'} + p'_j \qquad \forall j' \prec j \tag{6a}$$

$$\sum_{j\in\mathcal{F}} p'_j C_j \geq \frac{1}{2m}\left(\sum_{j\in\mathcal{F}} p'_j\right)^2 + \frac{1}{2}\sum_{j\in\mathcal{F}} p'^2_j \qquad \forall \mathcal{F} \subset \mathcal{V} \tag{6b}$$

The objective of the linear program is to minimize total weighted completion time. Constraint (6a) enforces precedence constraints among any predecessor–successor pair. Constraint (6b) is a weaker version of the no-time-sharing requirement; i.e., one machine can only process at most one task at a time.

This linear program does not find an optimal solution for the identical machine problem. Instead, we further build upon its solutions to construct a priority list. Let $\{C_j^{LP}\}$ denote the optimal solutions for the LP in Equation (6). The priority list is constructed with respect to α-points of tasks based on the LP solution, which are defined as below for $0 \leq \alpha \leq 1$. The α-point M_j^{LP} of task j based on the LP solution is defined as follows:

$$M_j^{LP} \triangleq C_j^{LP} - \alpha \cdot p'_j.$$

Once we compute the α-points of tasks, we obtain the priority list $\mathcal{L}$ by indexing these tasks with respect to the magnitude of α-points in a non-decreasing order; i.e., a task with a large

ALGORITHM 2: Learning-augmented Energy-aware List Scheduling: Total Weighted Completion Time

INPUT: DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and number of machines m.
OUTPUT: schedule $\mathcal{S}$ and speed scaling policy $\{s_j\}$.

First Stage
1: Approximate task pseudo-size $\{\beta_j\}$ via learning.

Second Stage
2: Transform the initial problem into the identical machine problem by assigning a new task size $p'_j \leftarrow p_j/s_j$, where $s_j = \sqrt{\beta_j/\gamma_j}$ for each task.
3: Construct a priority list $\mathcal{L}$ via LP (6). ▷ List scheduling starts **for** *task j in the priority list* $\mathcal{L}$ **do**
4: Start task j on any available machine at the earliest possible time.
end

α-point value has a high priority. In practice, we can search for a near-optimal α to further optimize over the overall objective, but for the purpose of this work, we assume that $\alpha = \frac{1}{2}$ to achieve the bounds in the theorems below.

Remark 1. While our analysis focuses on offline scheduling of a single job, our algorithm can be deployed in more general settings. For example, it can be implemented beyond the case of a single job to the online scheduling of multiple jobs via a straightforward extension. Simply connect a dummy source task node with outgoing edges pointed to all nodes with zero in-degree and a corresponding dummy sink with incoming edges from all nodes with zero out-degree. If a new job arrives into the system, while tasks are being scheduled, we can still connect the new job DAG to the dummy source and sink. Then, the entire system becomes one connected DAG, and Algorithms 1 and 2 can be applied. This reduction allows the algorithm to be deployed; however, an interesting open problem is to derive performance bounds for this extension.

4 Performance Bounds

Learning-augmented algorithms leverage machine learning tools to make a prediction in a data-driven fashion and then feed this prediction as the actual input to an algorithm. The goal is to design such an algorithm that incorporates these predictions and provides theoretical guarantees based on the quality of learning-based predictions. These learning-augmented algorithms have been applied in various settings, e.g., predicting task size for a scheduler [47], improving online algorithms [52, 53], etc. Similarly in this work, we take the inspiration of pseudo-size from the single server case and construct a scheduler and speed scaling policy via building upon the pseudo-size approximation. This allows our algorithm to adapt to workload data. Assuming a good approximation of task pseudo-size, we are able to provide worst-case bounds on a linear combination of performance and energy consumption when either makespan or total weighted completion time is considered. Even when a pseudo-size approximation is bad, our results characterize how the overall worst-case performance degrades based on the quality of the task pseudo-size approximation.

Total Weighted Completion Time. The scheduling algorithm makes use of the pseudo-size in two stages. First, we use learning-based algorithms to learn an approximation of the pseudo-size $\{\beta_j\}$. Second, we exploit the pseudo-size approximation to transform the problem into a scheduling problem with identical machines via assigning a new task size of $p_j \cdot \sqrt{\gamma_j/\beta_j}$ for task j. Then we can take advantage of the approximation algorithm [48] in the case of identical machines. We denote the schedule produced by the above algorithm as schedule S. Let $T(S)$ and $E(S)$ denote the

performance and energy consumption of the schedule S, respectively. We use $f(S)$ to denote the linear combination of performance and energy consumption for schedule S, i.e., $f(S) = T(S) + \lambda \cdot E(S)$.

Our main result is the following learning-augmented approximation ratio.[2]

Theorem 4.1 (Total Weighted Completion Time). *Given a pseudo-size approximation for tasks in a DAG satisfying* $(1-\epsilon_j^-)\cdot\beta_j^* \le \beta_j \le (1+\epsilon_j^+)\cdot\beta_j^*$, *a schedule S created by Algorithm 2 satisfies*

$$f(S) \le \max\left\{\max_j \frac{4}{\sqrt{1-\epsilon_j^-}}, \max_j \sqrt{1+\epsilon_j^+}\right\}\cdot OPT,$$

where $\{\beta_j^*\}$ is the optimal pseudo-size of the problem.

The approximation ratio in the above theorem is small as long as the pseudo-size is approximated well. To the best of our knowledge, this provides the first approximation bound for precedence-constrained tasks when minimizing a linear combination of performance and energy consumption, which is more general compared to settings focused on minimizing the makespan for a given energy budget as in [51]. Moreover, the techniques in [51] do not apply to the case of total weighted completion time. Our result provides a new perspective to minimize a linear combination of total weighted completion time and energy budget via an approximation of pseudo-size of tasks.

However, note that these bounds may not be tight. With regard to the case of minimizing total weighted completion time but not energy for identical machines, the best ratio is $2\log 2 + 2$ by [40]. This highlights that it may be possible to further improve the bound using time-indexed linear programming as in [40]; however, such a formulation would be significantly more complex, and it is difficult to bound the interaction with the learned advice in our setting.

Makespan. If makespan is adopted as the performance measure rather than total weighted completion time, during the second stage of the scheduling algorithm, we transform the problem into the identical machine problem and then apply a list scheduling algorithm. For a produced schedule S, we provide the following learning-augmented approximation ratio.

Theorem 4.2 (Makespan). *Given a pseudo-size approximation for tasks in a DAG satisfying* $(1-\epsilon_j^-)\cdot\beta_j^* \le \beta_j \le (1+\epsilon_j^+)\cdot\beta_j^*$, *a schedule S created by Algorithm 1 satisfies*

$$f(S) \le \max\left\{\max_j \frac{2-1/m}{\sqrt{1-\epsilon_j^-}}, \max_j \sqrt{1+\epsilon_j^+}\right\}\cdot OPT.$$

In contrast to Theorems 4.1 reduces the factor of 4 to a factor of $2 - 1/m$, highlighting an improved bound in the case of makespan (as expected). Again, the key point of the result is that the approximation ratio is small as long as the pseudo-size is approximated well. This also matches the best result in the case of minimizing makespan alone if predictions of task pseudo-size are perfect.

5 Proofs

We now prove our two main results, Theorems 4.1 and 4.2. Though the proofs here assume a quadratic scaling of pseudo-size from Definition 3.1 for simplicity of exposition, the case of general

[2]Note that our bounds hold for any arbitrary speed scaling exponent α. All one needs to do is to replace $\sqrt{1-\epsilon_j^-}$ and $\sqrt{1+\epsilon_j^+}$ with $(1-\epsilon_j^-)^{1/\alpha}$ and $(1-\epsilon_j^+)^{1/\alpha}$, respectively, in Theorems 4.1 and 4.2.

polynomial scaling can be derived using the same methodology with pseudo-size scaling as $n^{1/\alpha}$ instead of $\sqrt{n}$, where n is the number of waiting tasks.

Proof of Theorem 4.1

Our proof proceeds in three steps. We first bound the total energy consumption, E. Then, we bound the total weighted completion time, T. Finally, we combine these bounds to bound the overall objective.

(i) *Bound total energy consumption E.* Assuming an approximation of task pseudo-size, i.e., $(1-\epsilon_j^-)\cdot\beta_j^* \le \beta_j \le (1+\epsilon_j^+)\cdot\beta_j^*$, we can translate this approximation into a bound on speeds adopted in the algorithm:

$$\sqrt{1-\epsilon_j^-}\cdot s_j^* \le s_j \le \sqrt{1+\epsilon_j^+}\cdot s_j^*.$$

Given that energy consumption of task j is

$$e(j) = f(s_j)\cdot t_j = p_j \cdot s_j \le \sqrt{1+\epsilon_j^+}\cdot E_j^*,$$

we get an upper bound on the total energy consumption:

$$E(S) \le \max_j(\sqrt{1+\epsilon_j^+})\cdot E^*. \tag{7}$$

(ii) *Bound total weighted completion time T.* According to the main result (Theorem 4.1) in [48], for any schedule generated by the LP in Equation (6), we obtain

$$C_j - p'_j \le 2M_j^{LP} + 2\left(C_j^{LP} - p'_j\right),$$

which can be further relaxed as below:

$$C_j \le 4C_j^{LP} - 2p'_j. \tag{8}$$

Recall that C_j is the completion time of task j and p_j is its size. Let $T(p_1,\ldots,p_n)$ denote the optimal total weighted completion time for the identical machines problem with task sizes $(p_1,\ldots,p_n)$. Thus,

$$\sum_j \omega_j C_j \le 4\sum_j \omega_j C_j^{LP} \tag{9a}$$

$$\le 4\cdot T\left(p'_1,\ldots,p'_n\right) \tag{9b}$$

$$\le 4\cdot T\left(\frac{p_1}{s_1^*}\cdot\frac{1}{\sqrt{1-\epsilon_1^-}},\ldots,\frac{p_n}{s_n^*}\cdot\frac{1}{\sqrt{1-\epsilon_n^-}}\right) \tag{9c}$$

$$\le 4\cdot T\left(\frac{p_1}{s_1^*}\cdot\max_j\frac{1}{\sqrt{1-\epsilon_j^-}},\ldots,\frac{p_n}{s_n^*}\cdot\max_j\frac{1}{\sqrt{1-\epsilon_j^-}}\right) \tag{9d}$$

$$= \max_j\frac{4}{\sqrt{1-\epsilon_j^-}}\cdot T\left(\frac{p_1}{s_1^*},\ldots,\frac{p_n}{s_n^*}\right) \tag{9e}$$

$$= \max_j\frac{4}{\sqrt{1-\epsilon_j^-}}\cdot\sum_j\omega_j C_j^*. \tag{9f}$$

Inequality (9a) is a further relaxation of Inequality (8). As the LP in Equation (6) only covers partial constraints for the scheduling problem of identical machines, its optimal objective gives a lower bound as in Inequality (9b). Consider the scheduling problem in the case of identical machines: if we increase task sizes for each task, then the total weighted completion time will increase as

well. This explains why Inequality (9c) and Inequality (9d) hold true. If we scale the task size by a constant factor, then the total weighted completion time will scale correspondingly as in Equality (9e). This can be verified if we go over the constraints in the optimization formulation for the scheduling problem in the case of identical machines. Equality (9f) says that the general scheduling problem, when either performance or energy consumption is considered, can be reduced to the problem of minimizing total weighted completion time alone when task sizes are rescaled.

(*iii*) *Combine (i) and (ii) to bound the overall objective.* By combining Inequality (7) and Inequality (9), we achieve

$$T(S) + \lambda E(S) \leq \max\left\{\max_j \frac{4}{\sqrt{1-\epsilon_j^-}}, \max_j \sqrt{1+\epsilon_j^+}\right\} \cdot OPT.$$

Proof of Theorem 4.2

Our proof proceeds in three steps. First we bound the total energy consumption, E. Then we bound the total weighted completion time, T. Finally, we combine these bounds to bound the overall objective.

(*i*) *Bound total energy consumption E.* This step parallels the case of total weighted completion time. Assuming an approximation of task pseudo-size, i.e., $(1-\epsilon_j^-)\cdot\beta_j^* \leq \beta_j \leq (1+\epsilon_j^+)\cdot\beta_j^*$, we can translate this approximation into a bound on speeds adopted in the algorithm:

$$\sqrt{1-\epsilon_j^-}\cdot s_j^* \leq s_j \leq \sqrt{1+\epsilon_j^+}\cdot s_j^*. \tag{10}$$

Given that energy consumption of task j is

$$e(j) = f(s_j)\cdot t_j = p_j \cdot s_j \leq \sqrt{1+\epsilon_j^+}\cdot E_j^*, \tag{11}$$

we get an upper bound on the total energy consumption:

$$E(S) \leq \max_j(\sqrt{1+\epsilon_j^+})\cdot E^*. \tag{12}$$

(*ii*) *Bound makespan T.* We start by constructing a "chain" as follows. In the given schedule S, we start with the task that ends last. Among its predecessors, we choose one of the tasks with the latest completion time. There can be multiple candidate tasks that end at the same time. Though the overall performance might vary depending on different tie-breaking rules in practice, random tie-breaking rules are good enough for the purpose of this proof. We note that there is no idle gaps in between the execution of any pair of two predecessor–successor tasks in such a chain; otherwise, the successor task could have started earlier. We continue doing so until we reach the root of the DAG. In such a way, we construct a chain $\mathcal{J}$ such that every machine is busy during the complement time of the execution process of this chain of tasks. Thus,

$$\begin{aligned} T(S) &\leq \sum_{j\in\mathcal{J}} p'_j + \frac{\sum_{j\in\mathcal{V}} p'_j - \sum_{j\in\mathcal{J}} p'_j}{m} \\ &= \left(1-\frac{1}{m}\right)\sum_{j\in\mathcal{J}} p'_j + \frac{1}{m}\sum_{j\in\mathcal{V}} p'_j. \end{aligned}$$

Let $C_{max}\left(p_1,\ldots,p_n\right)$ denote the optimal makespan for the identical machines problem with task sizes $(p_1,\ldots,p_n)$. Then

$$T(S) \le \left(1-\frac{1}{m}\right)\cdot C_{max}\left(p_1',\ldots,p_n'\right) + C_{max}\left(p_1',\ldots,p_n'\right) \tag{14a}$$

$$\le \left(2-\frac{1}{m}\right)\cdot C_{max}\left(\frac{p_1}{s_1^*}\cdot\frac{1}{\sqrt{1-\epsilon_1^-}},\ldots,\frac{p_n}{s_n^*}\cdot\frac{1}{\sqrt{1-\epsilon_n^-}}\right) \tag{14b}$$

$$\le \left(2-\frac{1}{m}\right)\cdot C_{max}\left(\frac{p_1}{s_1^*}\cdot\max_j\frac{1}{\sqrt{1-\epsilon_j^-}},\ldots,\frac{p_n}{s_n^*}\cdot\max_j\frac{1}{\sqrt{1-\epsilon_j^-}}\right) \tag{14c}$$

$$= \max_j\frac{2-1/m}{\sqrt{1-\epsilon_j^-}}\cdot C_{max}\left(\frac{p_1}{s_1^*},\ldots,\frac{p_n}{s_n^*}\right). \tag{14d}$$

Inequality (14a) follows from the fact that the optimal makespan should be lower bounded by either the running time of any chain of tasks in the DAG or the completion time as if there were no precedence constraints. Consider the scheduling problem in settings with identical machines: if we increase the size of any task, the optimal makespan should not become any shorter, if not any longer. This is why Inequality (14b) and Inequality (14c) hold true. Further, if we scale task size by a constant factor, then the optimal makespan scales correspondingly. Again, this can be seen by writing the linear optimization formulation of the scheduling problem in the case of identical machines.

(*iii*) *Combine (i) and (ii) to bound the overall objective.* By combining Inequality (12) and Inequality (14d), we achieve

$$T(S)+\lambda E(S) \le \max\left\{\max_j\frac{2-1/m}{\sqrt{1-\epsilon_j^-}},\max_j\sqrt{1+\epsilon_j^+}\right\}\cdot OPT. \tag{15}$$

6 Case Studies

In this section, we provide experimental results detailing the application of our learning-augmented algorithm to real-world traces of precedence-constrained tasks. We detail the traces and experimental setup in Section 6.1 and then describe three approaches for approximating the pseudo-sizes in Section 6.2. Finally, we discuss the experimental results in Section 6.3.

6.1 Experimental Overview

6.1.1 Setup. Our experimental results focus on two classes of graphs: (i) real-world task traces and (ii) synthetic task traces. The real-world traces are our focus, and the synthetic traces are used to provide insights into the performance observed using the real-world traces.

The real-world traces are interdisciplinary workloads from the Pegasus Workflow Management System obtained without modification [18]. These traces comprise a variety of domains. We restrict our workloads to graphs with fewer than 500 tasks for computational reasons.

(1) **1000genome** is a data-intensive bioinformatics workload. This workload is made up of 15 graphs with the number of tasks ranging from 52 to 492.
(2) **cycles** is a compute intensive agroecosystem workload. This workload is made up of 10 graphs with the number of tasks ranging from 67 to 438.
(3) **epigenomics** is a data-intensive bioinformatics workload. This workload is made up of two graphs with the number of tasks ranging from 61 to 294.
(4) **montage** is a compute-intensive astronomy workload. This workload is made up of two graphs with the number of tasks ranging from 58 to 472.

(5) **seismology** is a data-intensive seismology workload. This workload is made up of four graphs with the number of tasks ranging from 101 to 401.
(6) **soykb** is a data-intensive bioinformatics workload. This workload is made up of eight graphs with the number of tasks ranging from 96 to 416.

For these traces, we report the cost averaged over all graphs in the trace.
Our synthetic graphs are divided into three different classes.

(1) **Random Graphs** are generated by using an Erdos-Renyi mechanism with probability $p = 0.3$.
(2) **Random Forks** are random graphs constrained to be only fork. This is done by sorting the set of nodes by number. Then we randomly add edges from nodes with lower numbers to nodes with higher numbers.
(3) **Random Joins** are random graphs constrained to be only Joins. We do this by taking Random Fork graphs and reversing the edges.

For these graphs, we report results for selected numbers of tasks and number of machines in order to illustrate performance on both congested and uncongested schedules. For instance, given a graph with 200 tasks and three machines, the schedule is likely to be very congested due to having many tasks to complete on very few machines. However, the schedule for a graph with 10 tasks and 50 machines would be uncongested as there are enough machines. Each reported number in our plots is the average of 20 random DAGs for each workload type. For all of our experiments, we fix $\lambda = 1$; i.e., performance and energy are equally weighted. We limit our synthetic graphs to a size of 200 in our experiments due to computational constraints and the demands of the non-learning-based pseudosize estimation methods. Note that the learning-based methods scale beyond this size.

6.1.2 Baselines. To evaluate the performance of our algorithm, we compare to the solution of the optimization problem in Section 3 (OPT). This provides an optimality gap for our learning-augmented approach. However, because this formulation is computationally intractable, we use an upper bound on the optimal that is computationally tractable for small graphs instead. In particular, UPPER-OPT uses the **earliest time first (ETF)** algorithm to obtain an ordering, i.e., a task-to-machine assignment. Subsequently, we analytically compute pseudo-sizes using intuition from the single-server speed scaling. This is done by obtaining intervals where tasks run concurrently on different machines before speed scaling. With these intervals, the dependencies, and the ancestors of these tasks, we estimate the pseudo-size. The explicit procedure is given in Algorithm 3 in the appendix. This approach is computationally demanding for larger graphs and is not feasible operationally, due to high runtime, but yields a near-optimal bound with which to benchmark our learning-augmented algorithms.

To understand the gap between our upper bound and the true optimal, we performed experiments for small systems where computing OPT is possible. These results are reported in Figure 1. The figure highlights that the gap in small systems is less than 3%.

Note that we do not include comparisons to existing methods because there are none for solving the performance plus energy objective we consider. Existing approaches either consider only the objective of makespan plus energy or are purely heuristic. Therefore, comparisons with existing approaches simply show the impact of optimizing a different objective. Instead, we chose to focus our experiments on the degree of suboptimality of our approach. Our experiments show that even with very simple learning methods to estimate pseudosizes, we obtain very close to the optimal performance. This is a stronger guarantee than we could obtain by comparing it with other algorithms.

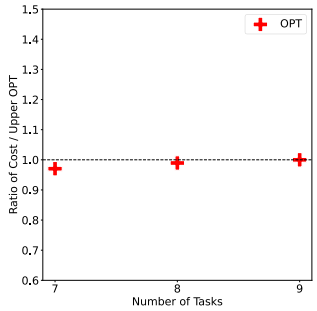


Fig. 1. Illustration of the gap between OPT and UPPER-OPT on small random workloads for two machines.

6.2 Estimating the Pseudo-size

Our proposed learning-augmented energy-aware list scheduling algorithm depends heavily on estimates of the pseudo-size. In this section, we detail two simple and effective learning approaches for obtaining these estimates. Additionally, we introduce a deterministic baseline approach for comparison purposes.

6.2.1 Baseline: Number of Descendants. We begin by introducing a simple, deterministic estimate of pseudo-sizes that first computes the task-to-machine assignment used under the ETF algorithm and then estimates the pseudo-size as the number of descendants of a particular task in the DAG. For simple graph structures like all-forks and uncongested schedules, this heuristic returns near-optimal pseudo-sizes due to the similarity of such scenarios with the single-server problem. However, on more complex structures and on congested systems, this heuristic does not work as well due to additional dependencies that need to be taken into account in order to achieve optimal performance. However, this approach has the benefit of simplicity and interpretability. We include this simple approach as a baseline for comparison with our learning-based approaches.

6.2.2 Why Learning-based Estimation? Our results in Section 6.3 show that a simple pseudo-size estimation such as in Section 6.2.1, though quick, is insufficient for a good approximation of an optimal schedule. In contrast, complex pseudo-size estimation procedures, like Algorithm 3, perform better yet fail to scale well due to their high computational complexity. Learning-based Pseudo-size Estimation measures allow us to achieve the best of both worlds by having both *fast inference time* and *improved accuracy*. By training a machine learning algorithm on a training set of DAGs similar to those seen in inference time, we optimize for pseudosize estimates similar to Algorithm 3, while at inference time, the trained model allows for fast predictions. For instance, for a 1,000 ER graph of size 200, deterministic approximation takes roughly 10 seconds, Algorithm 3 takes roughly 190 seconds, and the inference procedure of the learning-based measure takes roughly 15 seconds.

6.2.3 Dataset Synthesis for Learning-based Estimation. We now move to the details of our learning-based pseudo-size approximations. Before we can define the estimation procedures in the algorithms, we first need to describe how the training dataset is developed. Our learning-based methods seek to learn close-to-optimal pseudo-sizes from the graphs and additional features. For both of the methods described below, we generate the training data by using 80% of our graphs.

The specific node features we use in both Sections 6.2.4 and 6.2.5 are the number of descendants, out-degree betweenness centrality, and trophic levels, i.e., length of the shortest path from the root node of the DAG. In general, many other graph or non-graph features could be used for pseudo-size estimation depending on the job at hand. In our work, we do not use the number of machines as a feature as we regenerate the training data for every number of machines we consider. If only

one set of training data will be used for multiple machine numbers, this is an essential feature to consider

The final piece of the training data is to compute the pseudo-sizes. Please note that the pseudo-sizes of training data could be generated using many different principled or heuristic-based measures. In our work, we present one such pseudo-size estimation approach. Our approach uses a Relaxed Optimization Framework, defined as follows. The Relaxed Optimization Framework uses the same task to machine assignment computed by ETF and solves an optimization problem to obtain the optimal speed scaling for that task-to-machine assignment. Calculating the optimal pseudo-sizes from the speeds can be done using Equation (5). Unfortunately, Relaxed Optimization does not scale to a larger graph; it becomes computationally intractable quickly. The dataset synthesis for our entire synthetic traces for all number of machines is generated in under 5 hours on a 10th-generation i7 CPU with 16 GB RAM.

Note that each data point within our test data is a single task, not an entire graph. This means that our scheduling test set operates on each task independently. This allows us to predict pseudo-sizes at test time in an online fashion.

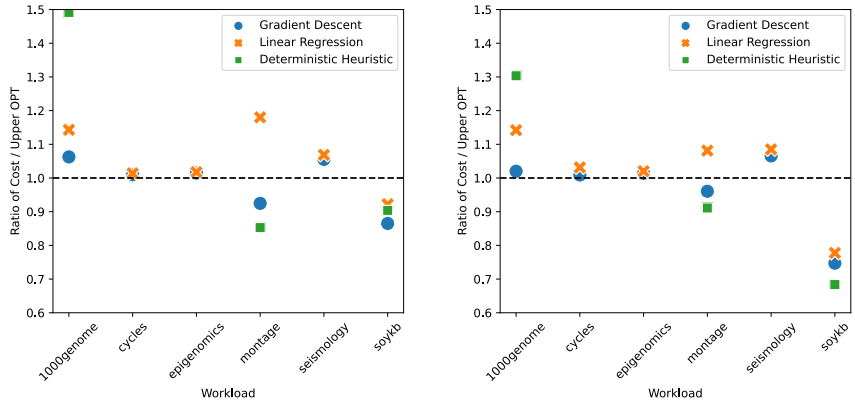
6.2.4 Learning-based Estimate: Linear Regression. In this method, we learn the optimal pseudo-size by regressing the dependent variable (pseudo-size) with the input features described above. In this approach, we treat each task, irrespective of the graph it came from, as a single data point in our dataset. We optimize for the weights of our regression model with an **Ordinary Least Squares (OLS)** loss function between the ground-truth pseudo-size returned by the Relaxed Optimization Framework and the labels predicted by the model. At test time, we evaluate our model using the total objective of the graph, the ordering generated by ETF, and the pseudo-size predicted by Linear Regression. Both training and inference are almost instantaneous.

6.2.5 Learning-based Estimate: Gradient Descent. As with the Linear Regression-based estimation, we again seek to learn the linear relationship between the features and the pseudo-sizes. However, here, the training procedure differs. Instead of using every task as an individual data point, we use an entire graph as a data point. For each of the tasks in each graph, we learn the directions of the weights that reduce the OLS error with a grid search (with step size 0.1). Then, we iterate through all of the graphs repeatedly until the total change in parameters is less than our stopping threshold of 0.001. After the model is trained, it is evaluated against the test set as described above in the case of Linear Regression-based estimation. The training for Gradient Descent takes roughly 30 minutes. Inference, however, like linear regression, is almost instantaneous.

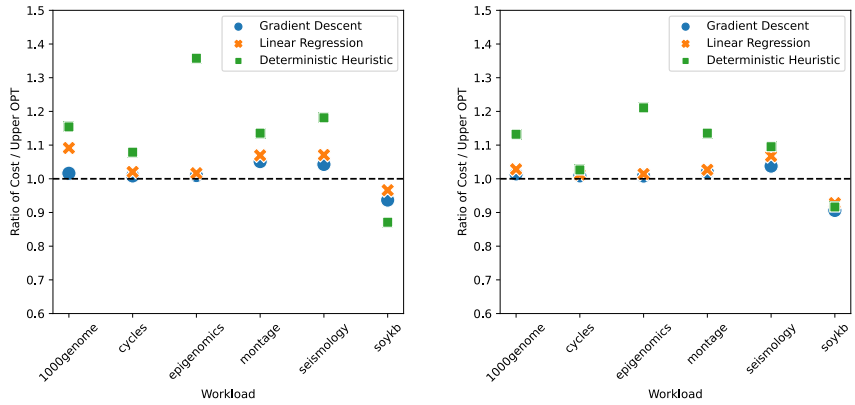
6.3 Results

Our main experimental results are presented in Figure 2. This figure shows the ratios of our learning-augmented algorithm to UPPER-OPT under real-world workloads given each of the three approaches for approximating pseudo-size described in the previous section. The figure shows the results for systems with 3, 10, and 50 machines. The results highlight that Gradient Descent outperforms the other two pseudo-size approximations, never performing more than 7% worse than UPPER-OPT. Linear Regression also performs well in nearly all situations, performing more than 10% worse than UPPER-OPT in only four cases. The deterministic approximation often is still within 10% of UPPER-OPT but has six cases where it does not achieve that ratio and, in some cases, is as much as 50% worse than UPPER-OPT.

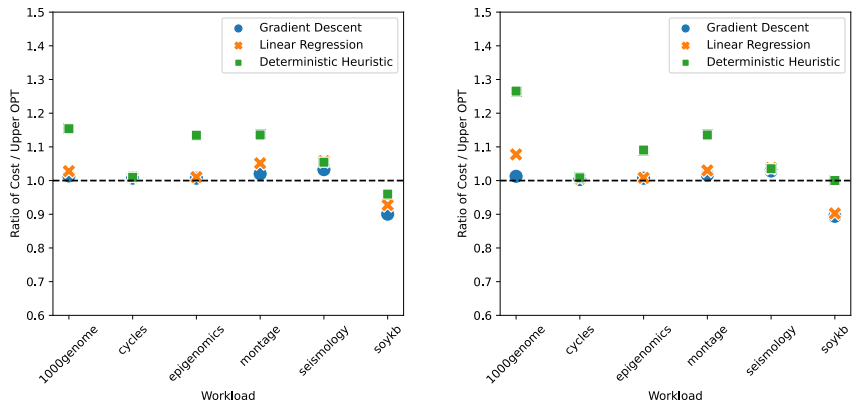
To further understand the behavior of our pseudo-size approximations, we use synthetic DAG structures for the precedence constraints. Figure 3 shows the performance of the three approximations in settings with various synthetic DAGs. This plot highlights again that Gradient Descent performs well across DAGs of different structures, while each of the other approaches has multiple



(a) 3 Machines (b) 5 Machines

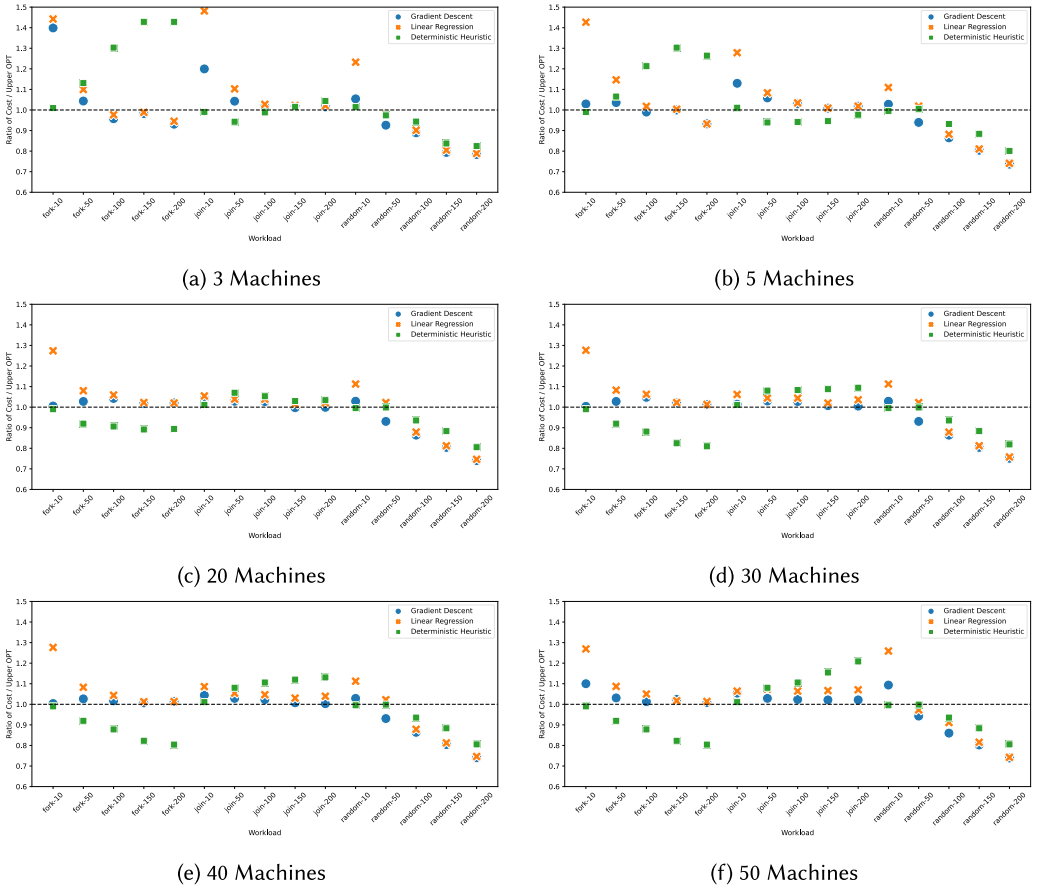


(c) 20 Machines (d) 30 Machines



(e) 40 Machines (f) 50 Machines

Fig. 2. Ratio of pseudo-size estimation methods to UPPER-OPT on real-world workloads for systems of different sizes.



(a) 3 Machines (b) 5 Machines

(c) 20 Machines (d) 30 Machines

(e) 40 Machines (f) 50 Machines

Fig. 3. Ratio of pseudo-size estimation methods to UPPER-OPT on synthetic workloads for systems of different sizes.

situations where the performance is significantly sub-optimal depending on the DAG structure. Interestingly, the results for the deterministic estimation highlight the importance of the congestion of the system. The approximation performs better when the system is less congested; i.e., the ratio of tasks to machines is smaller.

These experimental results show that with even relatively simple pseudo-size estimation methods like linear regression and gradient descent, we are able to obtain close-to-optimal performance. This empirically validates the facts that (1) obtaining "accurate enough" psuedo-size estimates from historical data is possible, and (2) the algorithmic framework is robust to realistic estimation errors of the pseudo-size estimates.

Our final set of experiments contrasts the performance of our algorithm for jointly scheduling and performing dynamic speed scaling with algorithms that choose a fixed, static speed and then perform list scheduling. These results highlight the benefits associated with dynamic speed scaling in multi-server systems. Figure 4 shows the performance of both the Gradient Descent and Linear Regression pseudo-size approximations along with the performance of static speed scaling, for a range of speeds. The figure highlights that, when the static speed is chosen optimally, it can nearly match the performance of dynamic speed scaling, with a gap that depends on the specifics of the workload and number of machines. However, if the static speed is not chosen optimally, then

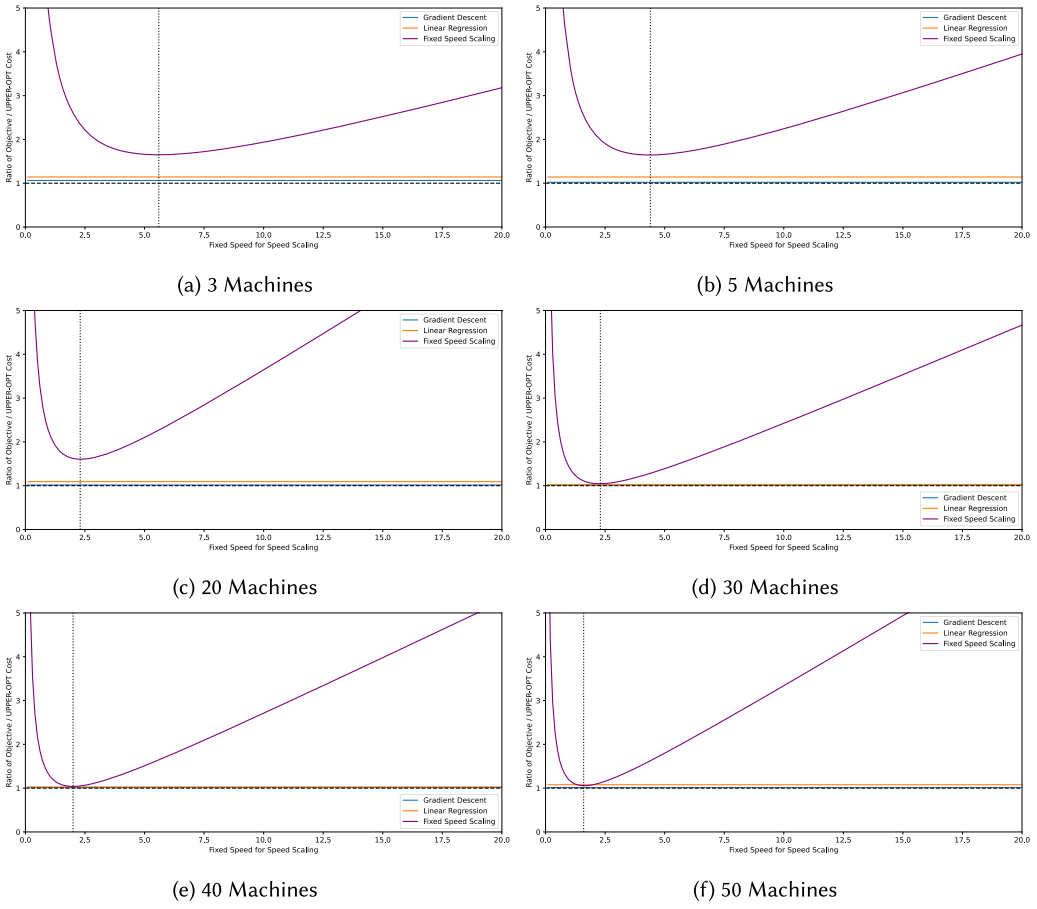


Fig. 4. Comparison of dynamic speed scaling and static speed scaling using the 1000genome workload. The dotted vertical line shows the optimal static speed.

the performance degrades quickly. Combined with the observation that the optimal static speed depends delicately on the workload and system (e.g., the optimal speed ranging from 2 to 6 in our three examples systems for the same workload), one can see that a significant benefit of dynamic speed scaling is the robustness it provides compared to static speed scaling.

7 Concluding Remarks

In this article, we consider the problem of scheduling precedence-constrained tasks to minimize a linear combination of performance and energy consumption. Our main results provide worst-case guarantees when the performance metric is either makespan or total weighted completion time. Further, we demonstrate that simple learning-based approaches perform accurate enough estimation of pseudo-size to achieve near-optimal performance on real-world precedence-constrained tasks. A variety of research questions are raised by our results. Most importantly, for particular structures of DAGs, is it possible to further refine the bounds by incorporating properties of these DAGs? Plus, is it possible to use this learning-augmented approach in the context of other energy-performance scheduling problems involving precedence constraints, e.g., scheduling to minimize completion time subject to an energy budget constraint or scheduling to minimize energy usage

subject to completion time constraints? Additionally, in stochastic settings, i.e., when the task sizes are unknown, could we generalize our bounds to account for this uncertainty?

Appendix

A Pseudo-size Calculation

Within the calculation of our upper bound on the optimal cast, UPPER-OPT, we calculate pseudo-sizes using Algorithm 3. The main idea behind the algorithm is to first obtain the same shared children (shared_subset) and their shared children for each interval set. These are generally the most important tasks to consider when computing pseudo-size and are inspired by the optimal

ALGORITHM 3: Procedure for Update_Pseudo-size

```
Input: DAG G = (V, E), interval set interval_set
Output: pseudo-size p
p = [0] * len(G)
/* Make Sharing Subset and Children                                    */
for interval in reversed(interval_set) do
    if No shared children for nodes in interval or no children of nodes in next interval then
        for node in interval do
            sharing_subset[node] = node
            children[node] = next task on same machine as node
        end
    end
    else
        for node in interval do
            sharing_subset[node] = interval
            children[node] = next task on same machine as node
        end
    end
    /* Calculating Psize using sharing_subset and children             */
    if len(sharing_subset) == 1 then
        node = sharing_subset.pop()
        for child in sharing_subset do
            p[node] += p[child]
        end
        p[node] += 1
    end
    else
        temp_p = 0
        for child in children do
            temp_p += p[child]
        end
        for task in sharing_subset do
            p[task] = (temp_p+len(sharing_subset)) / len(sharing_subset)
        end
    end
end
```

speeds to run at in the single-server world. After computing these quantities, based on the pseudo-sizes of the children and the length of shared_subset, we analytically compute the pseudo-size. Note that this approach is too computationally demanding for deployment but provides a useful bound for benchmarking of our pseudo-size approximation algorithms.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*. 265–283.

[2] Susanne Albers and Hiroshi Fujiwara. 2007. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms (TALG)* 3, 4 (2007), 49–es.

[3] Susanne Albers, Fabian Müller, and Swen Schmelzer. 2014. Speed scaling on parallel processors. *Algorithmica* 68, 2 (2014), 404–425.

[4] Dario Amodei and Danny Hernandez. 2018. AI and Compute. https://openai.com/blog/ai-and-compute/

[5] Evripidis Bampis, Dimitrios Letsios, and Giorgio Lucarelli. 2014. A note on multiprocessor speed scaling with precedence constraints. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*. 138–142.

[6] Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. 2008. Average rate speed scaling. In *Latin American Symposium on Theoretical Informatics*. Springer, 240–251.

[7] Nikhil Bansal, Ho-Leung Chan, Tak-Wah Lam, and Lap-Kei Lee. 2008. Scheduling for speed bounded processors. In *International Colloquium on Automata, Languages, and Programming*. Springer, 409–420.

[8] Nikhil Bansal, Ho-Leung Chan, Kirk Pruhs, and Dmitriy Katz. 2009. Improved bounds for speed scaling in devices obeying the cube-root rule. In *International Colloquium on Automata, Languages, and Programming*. Springer, 144–155.

[9] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. 2004. Dynamic speed scaling to manage energy and temperature. In *45th Annual IEEE Symposium on Foundations of Computer Science*. 520–529.

[10] Nikhil Bansal, Kirk Pruhs, and Cliff Stein. 2010. Speed scaling for weighted flow time. *SIAM Journal on Computing* 39, 4 (2010), 1294–1308.

[11] Sanjeev Baskiyar and Rabab Abdel-Kader. 2010. Energy aware DAG scheduling on heterogeneous systems. *Cluster Computing* 13, 4 (2010), 373–383.

[12] Rodrigo A. Carrasco, Garud Iyengar, and Cliff Stein. 2018. Resource cost aware scheduling. *European Journal of Operational Research* 269, 2 (2018), 621–632.

[13] David Chappell. 2015. Introducing azure machine learning. In *A Guide for Technical Professionals*. . Microsoft Corporation.

[14] Jian-Jia Chen, Heng-Ruey Hsu, Kai-Hsiang Chuang, Chia-Lin Yang, Ai-Chun Pang, and Tei-Wei Kuo. 2004. Multiprocessor energy-efficient scheduling with task migration considerations. In *Proceedings. 16th Euromicro Conference on Real-Time Systems, 2004 (ECRTS'04)*. IEEE, 101–108.

[15] Nicolas Christianson, Junxuan Shen, and Adam Wierman. 2023. Optimal robustness-consistency tradeoffs for learning-augmented metrical task systems. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 9377–9399.

[16] Fabián A. Chudak and David B. Shmoys. 1999. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Journal of Algorithms* 30, 2 (1999), 323–343.

[17] Edward Grady Coffman and John L. Bruno. 1976. *Computer and Job-shop Scheduling Theory*. John Wiley & Sons.

[18] Workflow Commons. 2021. Pegasus Instances. https://github.com/wfcommons/pegasus-instances

[19] Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. 2015. Energy-efficient resource allocation and provisioning framework for cloud data centers. *IEEE Transactions on Network and Service Management* 12, 3 (2015), 377–391.

[20] Sami Davies, Janardhan Kulkarni, Thomas Rothvoss, Jakub Tarnawski, and Yihao Zhang. 2020. Scheduling with communication delays via LP hierarchies and clustering. *arXiv preprint arXiv:2004.09682* (2020).

[21] Sami Davies, Janardhan Kulkarni, Thomas Rothvoss, Jakub Tarnawski, and Yihao Zhang. 2021. Scheduling with communication delays via LP hierarchies and clustering II: Weighted completion times on related machines. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA'21)*. SIAM, 2958–2977.

[22] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. 2021. Faster matchings via learned duals. *Advances in Neural Information Processing Systems* 34 (2021), 10393–10406.

[23] Anshul Gandhi, Mor Harchol-Balter, and Michael A. Kozuch. 2012. Are sleep states effective in data centers? In *2012 International Green Computing Conference (IGCC'12)*. IEEE, 1–10.

[24] Yuanxiang Gao, Li Chen, and Baochun Li. 2018. Spotlight: Optimizing device placement for training deep neural networks. In *International Conference on Machine Learning*. PMLR, 1676–1684.
[25] Noah Golowich and Ankur Moitra. 2022. Can Q-learning be improved with advice? In *Conference on Learning Theory*. PMLR, 4548–4619.
[26] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. 2020. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research* 21, 248 (2020), 1–43.
[27] Dorit S. Hochbaum and David B. Shmoys. 1987. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)* 34, 1 (1987), 144–162.
[28] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2018. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965* (2018).
[29] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. 2007. Algorithms for power savings. *ACM Transactions on Algorithms (TALG)* 3, 4 (2007), 41–es.
[30] Atefeh Khosravi, Lachlan L. H. Andrew, and Rajkumar Buyya. 2017. Dynamic VM placement method for minimizing energy and carbon cost in geographically distributed cloud data centers. *IEEE Transactions on Sustainable Computing* 2, 2 (2017), 183–196.
[31] Tim Kraska, Alex Beutel, Ed Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *Proceedings of International Conference on Management of Data*.
[32] Mohit Kumar, Subhash Chander Sharma, Anubhav Goel, and Santar Pal Singh. 2019. A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications* 143 (2019), 1–33.
[33] Woo-Cheol Kwon and Taewhan Kim. 2005. Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Transactions on Embedded Computing Systems (TECS)* 4, 1 (2005), 211–230.
[34] Alexandra Anna Lassota, Alexander Lindermayr, Nicole Megow, and Jens Schlöter. 2023. Minimalistic predictions to schedule jobs with online precedence constraints. In *International Conference on Machine Learning*. PMLR, 18563–18583.
[35] Young Choon Lee and Albert Y. Zomaya. 2009. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE, 92–99.
[36] Young Choon Lee and Albert Y. Zomaya. 2010. Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Transactions on Parallel and Distributed Systems* 22, 8 (2010), 1374–1381.
[37] Jan Karel Lenstra and A. H. G. Rinnooy Kan. 1978. Complexity of scheduling under precedence constraints. *Operations Research* 26, 1 (1978), 22–35.
[38] Keqin Li. 1999. Analysis of the list scheduling algorithm for precedence constrained parallel tasks. *Journal of Combinatorial Optimization* 3, 1 (1999), 73–88.
[39] Minming Li, Becky Jie Liu, and Frances F. Yao. 2006. Min-energy voltage allocation for tree-structured tasks. *Journal of Combinatorial Optimization* 11, 3 (2006), 305–319.
[40] S. Li. 2017. Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS'17)*. 283–294. https://doi.org/10.1109/FOCS.2017.34
[41] Shi Li. 2020. Towards PTAS for Precedence Constrained Scheduling via Combinatorial Algorithms. https://doi.org/10.48550/ARXIV.2004.01231
[42] Biswaroop Maiti, Rajmohan Rajaraman, David Stalfa, Zoya Svitkina, and Aravindan Vijayaraghavan. 2020. Scheduling precedence-constrained jobs on related machines with communication delay. *arXiv preprint arXiv:2004.10776* (2020).
[43] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*. 270–288.
[44] Jing Mei, Kenli Li, and Keqin Li. 2014. Energy-aware task scheduling in heterogeneous computing environments. *Cluster Computing* 17, 2 (2014), 537–550.
[45] Mohand Mezmaz, Nouredine Melab, Yacine Kessaci, Young Choon Lee, E.-G. Talbi, Albert Y. Zomaya, and Daniel Tuyttens. 2011. A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *Journal of Parallel and Distributed Computing* 71, 11 (2011), 1497–1508.
[46] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. 2017. Device placement optimization with reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2430–2439.
[47] Michael Mitzenmacher. 2019. Scheduling with predictions and the price of misprediction. *arXiv preprint arXiv:1902.00732* (2019).

[48] Alix Munier, Maurice Queyranne, and Andreas S. Schulz. 1998. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In *International Conference on Integer Programming and Combinatorial Optimization*. Springer, 367–382.

[49] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *Workshop Autodiff at Neural Information Processing Systems*.

[50] Ashkan Paya and Dan C. Marinescu. 2015. Energy-aware load balancing and application scaling for the cloud ecosystem. *IEEE Transactions on Cloud Computing* 5, 1 (2015), 15–27.

[51] Kirk Pruhs, Rob van Stee, and Patchrawat Uthaisombut. 2008. Speed scaling of tasks with precedence constraints. *Theory of Computing Systems* 43, 1 (2008), 67–80.

[52] Manish Purohit, Zoya Svitkina, and Ravi Kumar. 2018. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems*. 9661–9670.

[53] Dhruv Rohatgi. 2020. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1834–1845.

[54] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2019. Green AI. *arXiv preprint arXiv:1907.10597* (2019).

[55] Mohsen Sharifi, Saeed Shahrivari, and Hadi Salimi. 2013. PASTA: A power-aware solution to scheduling of precedence-constrained tasks on heterogeneous computing resources. *Computing* 95, 1 (2013), 67–88.

[56] Venkateswaran Shekar and Baback Izadi. 2010. Energy aware scheduling for DAG structured applications on heterogeneous and DVS enabled processors. In *International Conference on Green Computing*. IEEE, 495–502.

[57] Pawan Singh, Baseem Khan, Ankit Vidyarthi, Hassan Haes Alhelou, and Pierluigi Siano. 2019. Energy-aware online non-clairvoyant scheduling using speed scaling with arbitrary power function. *Applied Sciences* 9, 7 (2019), 1467.

[58] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243* (2019).

[59] Yu Su, Xiaoqi Ren, Shai Vardi, and Adam Wierman. 2020. Communication-aware scheduling of precedence-constrained tasks on related machines. *arXiv preprint arXiv:2004.14639* (2020).

[60] Navpreet Kaur Walia, Navdeep Kaur, Majed Alowaidi, Kamaljeet Singh Bhatia, Shailendra Mishra, Naveen Kumar Sharma, Sunil Kumar Sharma, and Harsimrat Kaur. 2021. An energy-efficient hybrid scheduling algorithm for task scheduling in the cloud computing environments. *IEEE Access* 9 (2021), 117325–117337. https://doi.org/10.1109/ACCESS.2021.3105727

[61] Sean Wallace, Xu Yang, Venkatram Vishwanath, William E. Allcock, Susan Coghlan, Michael E. Papka, and Zhiling Lan. 2016. A data driven scheduling approach for power management on hpc systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'16)*. IEEE, 656–666.

[62] Lizhe Wang, Gregor Von Laszewski, Jay Dayal, and Fugang Wang. 2010. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE, 368–377.

[63] Adam Wierman, Lachlan L. H. Andrew, and Ao Tang. 2009. Power-aware speed scaling in processor sharing systems. In *IEEE INFOCOM 2009*. IEEE, 2007–2015.

[64] Frances Yao, Alan Demers, and Scott Shenker. 1995. A scheduling model for reduced CPU energy. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*. IEEE, 374–382.

[65] Haitao Yuan, Jing Bi, Jia Zhang, and MengChu Zhou. 2022. Energy consumption and performance optimized task scheduling in distributed data centers. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52, 9 (2022), 5506–5517. https://doi.org/10.1109/TSMC.2021.3128430

[66] Yanli Zhu, Xiaoping Yang, Yi Hong, Youfang Leng, and Chuanwen Luo. 2021. Efficient energy utilization with device placement and scheduling in the internet of things. *Wireless Communications and Mobile Computing* 2021, 1 (2021), 5921181.

Received 1 May 2023; revised 25 March 2024; accepted 7 June 2024