Learning-Augmented Energy-Aware Scheduling of Precedence-Constrained Tasks

Yu Su[1], Jannie Yu[1], Vivek Anand[2], and Adam Wierman[1]

[1]California Institute of Technology

[2]The Pennsylvania State University

ABSTRACT

We study the scheduling problem of precedence-constrained tasks to balance between performance and energy consumption. To this point, scheduling to balance performance and energy has been limited to settings without dependencies between jobs. In this extended abstract, we consider a system with multiple servers capable of speed scaling and seek to schedule precedence constrained jobs 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 importance of a task in the precedence graph that is learned from workload data. We then propose a two-stage learning-augmented list scheduling algorithm which uses the learned pseudo-size approximation and achieves a provable approximation bound on the linear combination of performance and energy consumption, where the quality of the bound depends on that of the approximation of task pseudo-sizes, for both makespan and total weighted completion time.

1. INTRODUCTION

This extended abstract 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.Today this problem attracts attention due to the prominence of large-scale, general-purpose machine learning platforms, e.g. Google's TensorFlow, Facebook's PyTorch, and Microsoft's Azure Machine Learning (AzureML). Machine learning jobs on the cloud are often expressed as a directed acyclic graphs (DAG) 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. 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 CPUs, GPUs, TPUs, etc. This *device placement* problem has attracted considerable attention in recent years.

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 weighted completion time (via assigning equal weights to all the tasks), as is the makespan (via adding a dummy node of weight one as the final task with all other tasks assigned to weight zero). 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.

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 AI model can be as high as five times the lifetime emission of a car. The computation required for deep learning has been doubling every 3.4 months, resulting a 300,000x increase from 2012 to 2018. Indeed, the energy cost for an individual data center is on the order of million dollars and it has become a significant portion of operating cost for cloud platforms. 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 system performance and energy consumption is crucial to industry as well as societal goals of making cloud computing carbon neutral.

There has been considerable progress toward understanding how to schedule to balance performance and energy measures in simple settings, e.g., single and multi-server settings without dependencies between tasks. A focus within this line of work is on the question of co-designing scheduling of tasks and speed scaling of the server. On an algorithmic level, there are three common choices to conserve power consumption: static speed, gated static speed (a.k.a. sleep states, shutting down, power down), and speed scaling. Though the gated speed scheme is easier to implement in practice, speed scaling brings more potential to conserve energy by adjusting the speeds of a task depending on the priority of the task. If one task needs to be prioritized, then the scheduler assigns a fast speed to run the task at the cost of a high power consumption.

While there has been progress on studying speed scaling in simple scheduling problems, the question of how to balance

energy usage with traditional performance measures, e.g., 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., both the goal of partitioning the jobs across machines and of scheduling the jobs among a group of machines are NP-hard. Further, speed scaling adds considerable difficulty to the question of how to schedule tasks optimally to balance performance and energy consumption. 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-shelf solvers that work well for these problems on a relatively small scale in practice. In the presence of speed scaling, solving for the optimal schedule even for small scale problems becomes even more complex and computationally expensive in practice because the optimization problem is no longer linear. Given the hardness of the problem, a natural question is: *can we design a scheduler and speed scaling policy for precedence-constrained tasks that is provably near-optimal for a linear combination of performance and energy consumption?*

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 directed acyclic graph (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. 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 demand 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.

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 connected component. As a result, 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 assigning 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. Further, we assume that servers consume zero power when in idle, i.e., servers can run at zero speed.

Performance measure. The objective function we consider is $T + \lambda E$, a linear combination of a performance measure T and an energy measure E. The system operator can emphasize on either performance or energy consumption as desired via a choice of weight λ. For this extended abstract, we adopt total weighted completion time as the performance measure, and our results apply to both makespan and mean response time as well. As for energy consumption, our focus E is on total energy usage, which is the sum of energy usage of all tasks in the DAG.

Speed scaling. The scheduler has the ability to speed scale the server in order to trade off performance and energy consumption. In our model, the server chooses the speed s_j for task j. The energy consumption for a task running at speed s_j is modeled as the product of the instantaneous power $f(s_j)$ and running time t_j of that task, i.e., $f(s_j) \cdot t_j$. A common form of the power function in the literature is a polynomial, i.e., $f(s) = s^\alpha$ where $\alpha > 1$. 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 we can always adopt the average speed for running the task without any sacrifice on performance but potentially conserve more energy.

The optimization problem. We formally define the problem via an optimization formulation. Let $x_{i,j}$ be a a binary decision variable that indicates whether task j is assigned to machine i, i.e, $x_{i,j} = 1$ if task j runs on machine i. 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:

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

$$x_{i,j} \in \{0,1\} \qquad \forall i, j \quad \text{(1a)}$$

$$\sum_i x_{i,j} = 1 \qquad \forall j \quad \text{(1b)}$$

$$C_{j'} + \frac{p_j}{s_j} \le C_j \qquad j' \prec j \quad \text{(1c)}$$

$$\sum_j \omega_j C_j \le T \qquad \forall j \quad \text{(1d)}$$

$$\sum_j f(s_j)\frac{p_j}{s_j} = E \quad \text{(1e)}$$

$$\sum_i x_{i,j} x_{i,j'} = q_{j,j'} \qquad \forall j, j' \text{ if } j! = j' \quad \text{(1f)}$$

$$C_{j'} - C_j \ge \frac{p_{j'}}{s_{j'}} \text{ or } C_j - C_{j'} \ge \frac{p_j}{s_j} \qquad \text{if } q_{j,j'} = 1 \quad \text{(1g)}$$

The optimal value of this problem is denoted as OPT. Constraint (1b) requires every task to be scheduled on some machine. Constraint (1c) guarantees that for any successor predecessor pair, the successor task will not start until the predecessor completes. In Constraint (1d), T represents the total weighted completion time, and it can be simplified to align with other performance measures, such as makespan and mean response time. In Constraint (1e), E is the sum of multiplication of power function per unit time and running time. Constraint (1f) 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 (1g). By addition of an auxiliary variable, we can further simplify constraint (1g) but we will not dive into details due to limited space.

We emphasize that there are many possible formulations for the optimization problem. For example, we can derive another optimization formulation with a time-indexed program, but it would remain nonlinear due to speed constraints. Thus, we adopt the above formulation for simplicity. Given the optimization formulation, it is straightforward to conclude that searching for the global optimal of this complex problem is hard and computationally expen-

sive. Thus, we focus on approximation algorithms.

3. ALGORITHM DESIGN

Inspired by the single server case, we introduce the notion of *pseudo-size* to quantify importance of tasks in the DAG. Intuitively, a task of significance should be prioritized to run while a task of low priority should be set to run at a slow speed. The pseudo-size of a task depends on various features of the precedence graph, e.g., degree of nodes, number of children, etc. In practice, experts can extract these features and feed them to an off-shelf learning algorithm to obtain an approximation. This is a much simpler learning task than directly learning the optimal schedule and speed scaling policy together for two reasons. First, conditional on an approximation of pseudo-size, we are able to reduce the general problem, where it is hard to find an optimal solution, to a much simpler scheduling problem in the case of identical machines. This not only mitigates the required computations in process, but also makes it possible to compute a theoretical bound on the final schedule. Second, since the concept of pseudo-size stems from one 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, pseudo-size of a task quantifies magnitude of externalities it has on other tasks in the graph. This interpretation also brings knowledge of relevant features during the learning stage.

The single server case. We start by diving into the one server problem. As shown in [3], the optimal speed for running a task without precedence constraints is proportional to square root of the number of tasks that are waiting for its completion. The same intuition holds true in the single server case even if there are precedence constraints among 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 (1) with $m = 1$. For any given feasible ordering in which tasks are indexed with respect to the 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.*

For a large λ, i.e., emphasizing on energy consumption, 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. In this case, optimal speeds of a task is exactly 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.

Pseudo-size. The pseudo-size of a task is a measure of importance of the task via quantifying magnitude of externalities it has on other tasks in the DAG. If a task has many tasks waiting for its completion, then the scheduler tends prioritize the task via assigning a fast running speed. This parallels with 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 β_j for that task multiplied by the scaling parameter γ_j, i.e., $\beta_j = \gamma_j \cdot (s_j)^2$, where $\gamma_j = \frac{\lambda}{\omega_j}$.*

The scaling parameter depends on the choice of performance measure as well as the weight λ. For any relatively large scale problem, optimal speed is almost impossible to determine. As a result, a good approximation via an approximation of the pseudo-size becomes significant.

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 refined list scheduling in [1] for 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, i.e., $f(S) = T(S) + \lambda \cdot E(S)$. Our main result is the following learning-augmented approximation ratio.

THEOREM 3.2 (TOTAL WEIGHTED COMPLETION TIME). *Given a good pseudo-size approximation for tasks in a DAG, i.e. $(1-\epsilon_j^-)\cdot\beta_j^* \le \beta_j \le (1+\epsilon_j^+)\cdot\beta_j^*$, then any schedule S created by the above algorithm satisfies:*

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

Note that the approximation ratio is small as long as the pseudo-size is approximated well.

Makespan. If makespan is adopted, we can simply apply any list scheduling algorithm after the transformation. For a produced schedule, we prove the following learning-augmented approximation ratio.

THEOREM 3.3 (MAKESPAN). *Given a good pseudo-size approximation for tasks in a DAG, i.e. $(1-\epsilon_j^-)\cdot\beta_j^* \le \beta_j \le (1+\epsilon_j^+)\cdot\beta_j^*$, then any schedule S created by the above algorithm satisfies:*

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

To the best of our knowledge, this provides the first bound 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 [2]. Moreover, their techniques in [2] does not apply to the case of total weighted completion time.

4. REFERENCES

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

[2] K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. *Theory of Computing Systems*, 43(1):67–80, 2008.

[3] A. Wierman, L. L. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems. In *IEEE INFOCOM 2009*, pages 2007–2015. IEEE, 2009.