

Carbon- and Precedence-Aware Scheduling for Data Processing Clusters

Adam Lechowicz

University of Massachusetts Amherst

Noman Bashir

Massachusetts Institute of Technology

Adam Wierman

California Institute of Technology

Rohan Shenoy

University of California Berkeley

Mohammad Hajiesmaili

University of Massachusetts Amherst

Christina Delimitrou

Massachusetts Institute of Technology

Abstract

As large-scale data processing workloads continue to grow, their carbon footprint raises concerns. Prior research on carbon-aware schedulers has focused on shifting computation to align with availability of low-carbon energy, but these approaches assume that each task can be executed independently. In contrast, data processing jobs have precedence constraints (i.e., outputs of one task are inputs for another) that complicate decisions, since delaying an upstream “bottleneck” task to a low-carbon period will also block downstream tasks, impacting the entire job’s completion time. In this paper, we show that carbon-aware scheduling for data processing benefits from knowledge of both time-varying carbon and precedence constraints. Our main contribution is PCAPS, a carbon-aware scheduler that interfaces with modern ML scheduling policies to explicitly consider the precedence-driven importance of each task in addition to carbon. To illustrate the gains due to fine-grained task information, we also study CAP, a wrapper for any carbon-agnostic scheduler that adapts the key provisioning ideas of PCAPS. Our schedulers enable a configurable priority between carbon reduction and job completion time, and we give analytical results characterizing the trade-off between the two. Furthermore, our Spark prototype on a 100-node Kubernetes cluster shows that a moderate configuration of PCAPS reduces carbon footprint by up to 32.9% without significantly impacting the cluster’s total efficiency.

1 Introduction

Concerns about the climate impact of machine learning (ML) and artificial intelligence (AI) have primarily revolved around the carbon footprint during the training phase [28, 63] or, in some cases, the inference phase [37] of their life cycle. However, as the data requirements of foundation models have ballooned, the *data processing* tasks that must be completed before training account for almost one-third of the cumulative computation for an AI model during its life cycle [64]. Furthermore, foundation model *finetuning* generally trains a

model on a narrower data set that may require additional data processing [51] – as the finetuning of general purpose models (e.g., Llama) for specific tasks [41, 42] has gained traction, the comparative fraction of computational demand borne by data processing tasks is expected to grow.

Therefore, efforts towards responsible and sustainable development in AI must consider and optimize the carbon footprint of data processing. Even beyond sustainability, companies such as Microsoft have implemented *internal carbon pricing* for short- and long-term decisions [19, 57] that put financial responsibility on business divisions for each metric ton of operational CO₂ that they emit. In the data center context, most current schedulers do not consider the time-varying aspect of carbon intensity and the resulting compute-carbon impact – this must change to accommodate additional operational concerns such as carbon pricing.

Data processing frameworks (e.g., Apache Spark) ingest workloads that are composed of *precedence-constrained tasks*, where e.g., the outputs of one operation are the inputs to another [66]. There is a rich literature studying scheduling algorithms for this case of precedence-constrained tasks (e.g., represented as a directed acyclic graph (DAG)) that characterize large-scale data processing. From the theoretical side, optimal scheduling of precedence-constrained tasks (in terms of total completion time) is known to be NP-hard [36]. Although there has been progress in approximation techniques [11, 14, 15, 32, 39, 47, 58, 59], the hardness of the problem necessitates simple settings with relatively strong assumptions. From an experimental perspective, there have been several studies proposing data-driven and/or evolutionary approaches for scheduling, both in the general precedence-constrained tasks case and the specific data processing case [10, 16, 26, 30, 38, 48, 52, 65, 68]. In recent years, such works have leveraged learning techniques such as graph neural networks (GNNs) and reinforcement learning (RL) to learn an improved scheduling policy, showing significant improvements in experiments. However, owing to the complexities of these approaches, theoretical guarantees for learning-based approaches have proven difficult to obtain.

Beyond the singular objective of job completion time, a select few works have considered settings that are closer to the carbon-aware problem we study in this paper [24, 43, 59]. These *multi-objective* scheduling environments balance the objectives of e.g., reducing job completion time alongside another metric of interest, such as cost. For instance, several works have considered energy efficiency in tandem with job completion time, from both theoretical and experimental perspectives. Su et al. [59] study *energy-aware* list scheduling for precedence constrained tasks, giving theoretical bounds for an combined objective of energy consumption and performance. GreenHadoop [24] is a MapReduce framework for data centers with local renewable sources that predicts the future availability of carbon-free (“green”) electricity and schedules jobs accordingly, subject to deadlines for individual jobs. Liu et al. [43] consider job scheduling for low-carbon data center operation in a general model with both DAG and non-DAG jobs – they develop an RL-based scheduler that focuses primarily on increasing energy-efficiency.

Despite these previous works, focusing on *carbon-efficiency* rather than energy-efficiency requires different techniques. In particular, while carbon-efficiency and energy-efficiency are sometimes complementary objectives, they are often contradictory [27] – for instance, due to the *time-varying* nature of carbon intensity, it may be advantageous to scale up during low-carbon periods (i.e., sacrificing energy efficiency) in exchange for the ability to scale down during high-carbon periods. Works that *do* consider carbon emissions (e.g., GreenHadoop) use abstractions, such as job-level deadlines and “green” vs. “brown” energy, that do not adequately model the current m.o. in data centers.

To address this multi-objective setting while catering to realistic scenarios, we propose that a middle-ground approach is needed – namely, by drawing on techniques from the theoretical literature for precedence-constrained and carbon-aware scheduling, and simultaneously considering experimental advances, we seek a simple and interpretable framework that comes with guarantees in terms of the *trade-off* between job completion time and carbon savings.

In this paper, we propose PCAPS (Precedence- and Carbon-Aware Provisioning and Scheduling), a carbon-aware scheduler for data processing clusters. PCAPS is theoretically-inspired, leveraging a paradigm of interpretable and configurable *threshold-based design* that informs decisions at each time step based on e.g., the current carbon intensity and/or carbon price. In keeping with this inspiration, we give analytical results that characterize the trade-off between job completion time and carbon savings. PCAPS is also practically relevant, drawing on recent insights from ML-based DAG schedulers (e.g., Decima [48], LACHESIS [68], and others [26, 38, 65]). By interfacing with a score or probability distribution over available tasks, PCAPS defines a notion of *relative importance* (i.e., compared to other tasks) – this allows it to make fine-grained carbon-aware decisions that take the DAG’s structure

into account, such as continuing to schedule bottleneck tasks even if carbon intensity is high. See Section 4.1 for a formal description of PCAPS’s design.

As a simplification of PCAPS, we additionally propose and study CAP (Carbon-Aware Provisioning), which takes the provisioning ideas of PCAPS and generalizes them to interoperate with any carbon-agnostic scheduler. Without explicitly considering inter-task dependencies, CAP changes the resources available to the cluster, capturing an intuition that clusters should *throttle down* during high-carbon periods and vice versa [28, 54] – see Section 4.2 for a description.

We have implemented PCAPS and CAP as modules for Spark on Kubernetes and as extensions for a high-fidelity simulator [48]. Our experiments consider real and synthetic data processing workloads from Alibaba traces and TPC-H [1, 60], alongside real carbon intensity traces from six power grids [18]. In our prototype implementation, we evaluate PCAPS and CAP on a 100-node Spark cluster. We report the impact of carbon-aware policies on both job completion time and carbon savings, showing that PCAPS and CAP’s configurable design enables notable carbon reduction for mild increases in *end-to-end completion time*, which measures the total time to complete all jobs in a given experiment, measuring the system’s overall throughput and efficiency. We summarize our key contributions as follows:

1. PCAPS, a carbon-aware scheduler that interfaces with a probability distribution over stages of a DAG, such as those provided by ML schedulers. PCAPS incorporates carbon into decisions that arbitrage between stages of a job at a granular level, obtaining a favorable trade-off between carbon savings and job completion time.
2. CAP, a carbon-awareness module that dynamically adjusts cluster resources without replacing an existing scheduler (see Section 4.2). Compared to PCAPS, it obtains a worse trade-off between carbon and completion time in exchange for flexibility and ease of implementation.
3. We analyze the *carbon stretch factor* for PCAPS and CAP, which bounds the increase in job completion time due to carbon-aware actions (e.g., see Theorems 4.5 and 4.3).
4. We implement PCAPS and CAP as extensions for a high-fidelity Spark simulator, alongside proof-of-concept prototypes for Spark on Kubernetes (see Section 5). We evaluate our proposed carbon-aware schedulers against baselines and a state-of-the-art ML scheduler (see Section 6).

2 Problem and Motivation

This section formalizes the carbon-aware scheduling problem and motivates insights to contextualize our desiderata.

Our experiment code is available at <https://github.com/umass-solar/carbon-aware-dag/>.

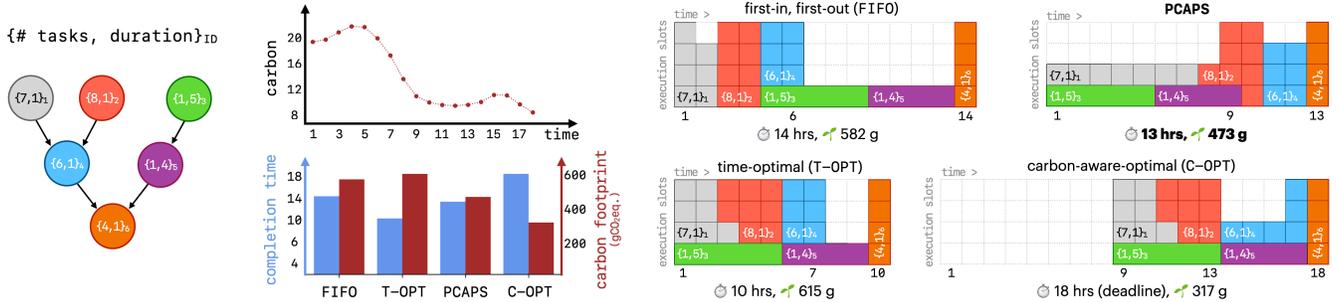


Figure 1: Four scheduling policies for a motivating DAG and 18-hour-long carbon intensity trace (on the left hand side). Compared to a carbon-agnostic FIFO scheduler, the time-optimal approach (T-OPT) prioritizes starting the green and purple stages early to reduce completion time. A carbon-aware-optimal approach (C-OPT) with a *deadline* to finish the DAG within 18 hours reduces carbon emissions by 51.2%, at the expense of increasing time by 28.5% compared to FIFO. By prioritizing green and purple stages during high-carbon periods, PCAPS reduces carbon emissions by 23.1% and still completes the job 7% earlier compared to FIFO.

2.1 Carbon-aware DAG scheduling problem

Each job is represented as a directed acyclic graph (DAG) $J = \{\mathcal{V}, \mathcal{E}\}$, where each node in \mathcal{V} is one of n tasks, and each edge in \mathcal{E} encodes precedence constraints between tasks – e.g., for tasks $j, j' \in \mathcal{V}$, an edge $j \rightarrow j'$ indicates that j' cannot start until after j has completed. A typical data processing cluster includes $K \geq 1$ machines (or executors). More than one job can simultaneously run on a cluster – e.g., given a set of current jobs $\{J\}$, the scheduler assigns tasks to machines over time while respecting precedence and capacity constraints. We index continuous time by $t \geq 0$.

The goal of a typical scheduler is *performance*, e.g., in terms of throughput, utilization, and average job completion time. In this work, we additionally consider the goal of *carbon-awareness* – with respect to a time varying carbon signal given by a function $c(t) : t \geq 1$, a carbon-aware scheduler’s objective is to minimize a combination of typical metrics (i.e., job completion time) and the overall carbon footprint (on both a per-job and a global, cluster basis).

Although future values of this carbon signal are unknown to the scheduler, in the rest of the paper, we follow prior work [5, 33] and assume that it is bounded by constants L and U that are known to the scheduler, where $L \leq c(t) \leq U$. In practice, the values of L and U can capture e.g., short-term forecasts of the best and worst carbon conditions on a given electric grid over the next 24 or 48 hours.

2.2 Prior work and motivation

Scheduling directed acyclic graphs (DAGs), or more broadly, precedence-constrained tasks, has been extensively studied. Classic results establish the difficulty of this problem: even in its simplest forms, DAG scheduling is NP-hard [36]. To address this, prior work has developed heuristic methods and approximation algorithms [11, 14, 15, 32, 39, 47, 58, 59], ranging from the well-known list scheduling algorithm [25], priority-based algorithms [55], to more complex approaches

such as genetic programming [10, 16, 52]. These methods often rely on simplifying assumptions, such as fixed task durations or centralized knowledge of the task graph.

In recent years, DAG scheduling has become a key problem in *data processing frameworks* such as Apache Airflow, Beam, and Spark, which use DAGs to represent workflows. In Spark, each node of a job’s DAG corresponds to a *stage*, which encapsulates operations (*tasks*) that can be executed in parallel over partitions of input data. Inter-stage dependencies impose precedence constraints: a stage can only begin once all “parent” stages have completed. Frameworks such as Spark typically implement simple scheduling strategies such as first-in, first-out (FIFO) and fair-share scheduling [22] – these are explainable and efficient in terms of overhead, but suboptimal in terms of job completion time.

Recent works that revisit scheduling for data processing have explored learning-based techniques, such as reinforcement learning (RL) methods that dynamically learn scheduling policies [26, 30, 38, 48, 65, 68]. Although these methods outperform default policies and hand-tuned heuristics in terms of job completion time, theoretical results for these techniques have proven difficult to obtain.

Carbon awareness adds a new dimension to the DAG scheduling problem – an online scheduler must consider the time-varying carbon intensity while choosing to assign resources to specific nodes in the job DAG(s), with an overarching goal of reducing carbon footprint, combined with traditional metrics such as job completion time – see Fig. 1 for an illustration of this desired behavior for PCAPS, FIFO, and optimal schedules. As discussed above, the state-of-the-art for carbon-agnostic DAG scheduling falls into two categories: theoretical models that focus on provably near-optimal schedules under idealized assumptions, and heuristic or learning-based methods that do not provide theoretical bounds but perform well in practice. In adding carbon-awareness to the problem, we consider a *middle ground* that balances between design goals of simplicity, interpretability, configurability, and

performance. In particular, we seek a carbon-aware scheduler that is tractable for theoretical insight, offering provable bounds on, e.g., the trade-off between carbon and job completion time while not sacrificing the efficiency gains that come from, e.g., learning DAG structure.

3 Theoretical Foundations

This section details theoretical underpinnings and intuition for our design in Section 4. Recent literature has studied carbon-aware scheduling problems with a theoretical lens [5, 33, 34, 35], spanning relatively simple suspend-resume [33] to settings considering scaling and uncertainty in job lengths [5]. In these online carbon-aware scheduling problems, the key challenge is the inherent uncertainty in future carbon intensity values due to the proliferation of intermittent renewable energy sources.

A common approach to manage this uncertainty is *threshold-based design* [4, 33, 63], that uses a predetermined and parameterized threshold function to inform decisions. Among studies that use this design paradigm, a common theoretical performance metric is *competitive ratio*, which is the worst-case ratio (≥ 1) between the cost of an online algorithm vs. that of an optimal solution. Algorithms designed using this metric are known to be pessimistic in practice [46, 53]. Moreover, existing theoretical studies on carbon-aware scheduling focus on simple settings where, e.g., the job is bound by a deadline, the objective is only to reduce carbon, and precedence constraints are not considered. However, threshold-based algorithms have been demonstrated to work well in practice: they are often close to optimal provided their inputs are reasonably accurate [13].

Carbon-aware DAG scheduling exhibits an inherent trade-off between carbon savings and job completion time (JCT). Although worst-case metrics (i.e., bounds with respect to an intractable offline solution) have limited utility in this setting, it is still useful to quantify a trade-off between carbon and JCT – to this end, we introduce two metrics that we use in the following sections. We start by introducing some notation: let $\text{OPT}_K(\mathcal{J})$ denote the optimal makespan for job \mathcal{J} , and let $\text{ALG}_K(\mathcal{J})$ denote the makespan for the schedule generated by some scheduler ALG (all using a maximum of K machines).

Definition 3.1 (Carbon Stretch Factor (CSF)). *Given a scheduling policy (e.g., FIFO), let AG denote the regular (i.e., carbon-agnostic) scheduling policy, and let CA denote a carbon-aware variant of the same scheduling policy. If a is an upper bound such that $\text{AG}_K(\mathcal{J}) \leq a \cdot \text{OPT}_K(\mathcal{J}) : \forall \mathcal{J}$, and b is an upper bound such that $\text{CA}_K(\mathcal{J}) \leq b \cdot \text{OPT}_K(\mathcal{J}) : \forall \mathcal{J}$, where $b \geq a$, then the **carbon stretch factor** is defined as b/a , which indicates (multiplicatively) how much worse the makespan of CA is compared to AG. Note that $b/a \geq 1$.*

To quantify carbon savings, we define $C_{\text{ALG}}(t)$ as the instantaneous carbon emissions at time t due to decisions by scheduler

ALG. It is a function of the number of executors active in ALG’s schedule at time t (denoted by $E_{\text{ALG}}(t)$) and the current carbon intensity: $C_{\text{ALG}}(t) := c(t)E_{\text{ALG}}(t)$.

Definition 3.2 (Carbon Savings). *Let AG and CA denote a carbon-agnostic and carbon-aware scheduler as outlined in Def 3.1. For a job \mathcal{J} , if AG runs from time step 0 until T (its completion time), and CA operates from time 0 to T' , then CA’s **carbon savings** are given by $\int_0^T C_{\text{AG}}(t) - \int_0^{T'} C_{\text{CA}}(t)$.*

Using CSF and carbon savings, we describe the desired behavior of a carbon-aware scheduler for data processing. A basic intuition in threshold-based designs is “hedging” between completing tasks now and waiting for lower-carbon periods that may arrive. To do this, thresholds rely on the *range* of carbon intensities that are expected to appear in the near future (i.e., L and U). In the context of CSF, this translates into two conditions that a scheduler should satisfy:

i) If the fluctuation of carbon intensity is *low* (e.g., L and U are close), the CSF should be close to 1, i.e., JCT should be close to that of the carbon-agnostic algorithm.

ii) If the fluctuation is *high* (e.g., L and U are not close), the CSF should be *finite*, i.e., the scheduler does not wait indefinitely to complete the job. In threshold-based designs, this is often met by imposing a *deadline* on the job [24, 33].

In the context of the DAG scheduling for data processing workloads, additional unique challenges exist. For instance, in the single job settings considered by prior work, specifying a deadline for each job is straightforward [5, 33]. However, on a cluster scale that considers multiple jobs of unknown length and different arrival times, setting a proper deadline quickly becomes complicated. Instead, our schedulers (see Section 4) guarantee a minimum amount of job progress whenever there are outstanding tasks in the queue.

Due to precedence constraints, carbon-aware scheduling actions that do not consider the structure of the DAG may inadvertently block bottleneck tasks from processing, having a large negative impact on JCT. This gives a third condition:

iii) When fluctuation is *high* (i.e., L and U are not close) and the system is in a high-carbon period, a scheduler should carefully consider the *structure* of a job’s DAG, prioritizing *bottleneck* tasks to use the limited cluster resources.

Conditions **i - iii** summarize the desired high-level behavior of a carbon-aware scheduler for data processing workloads. In the following section, we present PCAPS that takes into account the above conditions in its design.

4 Design

In this section, we present PCAPS, our Precedence- and Carbon-Aware Provisioning and Scheduling system, and then, as a flexible and easy-to-implement alternative, we present CAP (Carbon-Aware Provisioning).

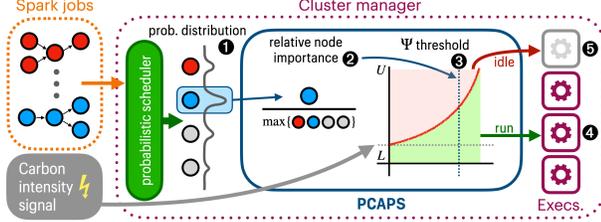


Figure 2: PCAPS interfaces with a probabilistic (PB) scheduling policy. Given a probability distribution over nodes ①, PCAPS computes a *relative importance* score ② that is used to determine which nodes should run based on the current carbon intensity ③ – e.g., bottleneck nodes impeding job completion run regardless of carbon ④, while less important nodes can be deferred for lower carbon periods ⑤.

4.1 PCAPS

From the discussion in Section 3, we seek an interpretable and configurable scheduler that satisfies the conditions **i** – **iii**). To this end, we introduce PCAPS (Precedence- and Carbon-Aware Provisioning and Scheduling), which interfaces with a probabilistic DAG scheduler such as Decima [48].

PCAPS’s key idea is a metric of *relative importance* (Def. 4.2) that is implicitly embedded in a probability distribution over tasks. We define a configurable carbon and importance-aware threshold function that uses the *relative importance* metric to make per-task fine-grained scheduling decisions – as illustrated in Fig. 2. Next, we detail how the theoretical literature on threshold-based algorithms inspires PCAPS, describe its operation, and discuss analytical results that characterize the trade-off between carbon savings and JCT.

PCAPS design. We first formalize a class of *probabilistic schedulers* that PCAPS interfaces with, giving a concrete example of an ML scheduler in this class.

Definition 4.1 (Probabilistic Scheduler). *At each scheduling event,¹ a probabilistic scheduler generates a distribution $\{p_{v,t} : v \in \mathcal{A}_t\}$, where \mathcal{A}_t denotes the set of tasks that are ready to be executed at time t .*

One example of a probabilistic scheduler is Decima [48], an RL-based scheduler for data processing workloads. Decima learns actions in the form of scores for each task – a masked softmax is applied to these scores to obtain a probability distribution over \mathcal{A}_t , and the next scheduled task is sampled from this distribution. Recall the motivation behind PCAPS: in addition to ramping down during high-carbon periods and ramping up during low-carbon periods, bottleneck tasks (i.e., tasks with a large score) should be scheduled even if the carbon intensity is high to reduce JCT. To this end, we define a notion of *relative importance* that compares the probability mass assigned to a single task v against other tasks in \mathcal{A}_t .

¹Scheduling events include job arrivals, task completions, and machines becoming available.

Algorithm 1 PCAPS (Precedence- and Carbon-Aware Provisioning and Scheduling)

- 1: **input:** hyperparameter γ , threshold function $\Psi_\gamma(\cdot)$, probabilistic (carbon-agnostic) scheduler PB
 - 2: **define:** a *scheduling event* occurs whenever PB is invoked or the carbon intensity $c(t)$ changes
 - 3: **while** cluster active at time $t \geq 0$ **do**
 - 4: **if** scheduling event at time t **then**
 - 5: Sample $v \in \mathcal{A}_t$ and probabilities $p_{v,t} : v \in \mathcal{A}_t$ from PB
 - 6: Compute relative importance $r_{v,t} = \frac{p_{v,t}}{\max_{u \in \mathcal{A}_t} p_{u,t}}$
 - 7: **if** $\Psi_\gamma(r_{v,t}) \geq c(t)$ **or** no machines currently busy **then**
 - 8: Send task v to an available machine at time t
 - 9: **else**
 - 10: Idle until next scheduling event
-

Definition 4.2 (Relative Importance). *Given a time $t \geq 0$ and node $v \in \mathcal{A}_t$, the relative importance $r_{v,t}$ is defined:*

$$r_{v,t} := \frac{p_{v,t}}{\max_{u \in \mathcal{A}_t} p_{u,t}} \in [0, 1].$$

If a task’s relative importance is closer to 1, the task is relatively *more important*, and a value closer to 0 implies the opposite. Note that if $|\mathcal{A}_t| = 1$ (i.e., only one task can be scheduled), the importance of that task is 1. Leveraging inspiration from threshold-based design, we define scheduling decisions using a threshold function Ψ_γ that considers the current carbon intensity and the relative importance of a task. $\gamma \in [0, 1]$ is a carbon-awareness parameter that controls the “strictness” of the function: $\gamma = 0$ recovers carbon-agnostic actions, while $\gamma = 1$ is maximally carbon-aware for tasks with low relative importance. We define Ψ_γ as:

$$\Psi_\gamma(r) := (\gamma L + (1 - \gamma)U) + [U - (\gamma L + (1 - \gamma)U)] \frac{\exp(\gamma r) - 1}{\exp(\gamma) - 1},$$

The function $\Psi_\gamma(\cdot)$ exhibits an exponential dependence on r , the relative importance of a task. This draws on continuous versions of online search [17, 69], where an exponential trade-off is derived by balancing the marginal reward of the current price against the risk that better prices exist in the future. We interpret relative importance analogously: high-importance tasks are scheduled regardless of carbon intensity to avoid the negative impact of not scheduling them, while low-importance tasks can be deferred to wait for lower-carbon periods with less impact on JCT. This threshold function is used in a *carbon-awareness filter* of sampled tasks before they are scheduled– we formalize this in Algorithm 1:

PCAPS’s carbon-awareness filter accomplishes all three of the motivation points defined in Section 2.2. It schedules (or defers) tasks based on the current carbon intensity $c(t)$, with the effect of reducing execution during high-carbon periods. Furthermore, the likelihood of a task being scheduled irrespective of the current carbon intensity is proportional to its importance in the DAG (i.e., in terms of precedence constraints). Note that $\Psi_\gamma(1) = U$ – tasks with high relative importance are always scheduled. While deferring a task has a negative

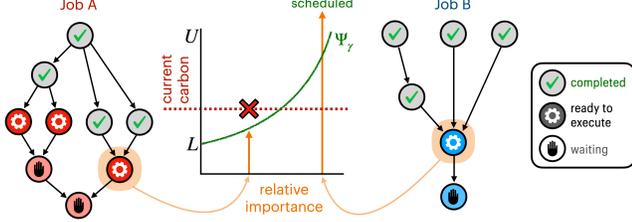


Figure 3: Illustrating PCAPS’s carbon-awareness filter. Jobs A and B are DAGs found in TPC-H queries and Alibaba traces, respectively [1, 60]. Highlighted nodes explain two scheduling outcomes. In job A, the sampled node has low relative importance, so it is deferred. In contrast, job B’s sampled node is a *bottleneck* task with high relative importance: even when the current carbon intensity is high, such tasks are scheduled to avoid increasing job completion time.

impact on an individual job’s JCT, PCAPS is optimized for the case where multiple DAGs share a cluster. Prioritizing outstanding bottleneck tasks across all jobs helps to manage the system’s *end-to-end completion time* (ECT) for e.g., a set of jobs. In Fig. 3, we illustrate the intuitions behind PCAPS’s carbon-awareness filter using two sample job DAGs.

Analytical results. We analyze the *carbon stretch factor* (Def. 3.1) and *carbon savings* (Def. 3.2) for PCAPS, with full proofs in Appendix B.1. PB denotes a carbon-agnostic probabilistic scheduler throughout. For an arbitrary job \mathcal{J} , we let $\mathcal{D}(\gamma, \mathbf{c}) \in [0, 1]$ denote a function that depends on the *expected amount of deferrals* (i.e., the tasks that PCAPS prevents from being scheduled, which depends on the carbon signal \mathbf{c}). See Appendix B.1.1 for a formal definition.

Theorem 4.3. *For time-varying carbon intensities given by \mathbf{c} , the carbon stretch factor of PCAPS is $1 + \frac{\mathcal{D}(\gamma, \mathbf{c})K}{2 - \frac{1}{K}}$.*

At a high level, $\mathcal{D}(\gamma, \mathbf{c})$ describes the fraction of tasks (in terms of total runtime) that are deferred by PCAPS with a given γ and carbon trace \mathbf{c} . It is ≤ 1 for any γ , and $\mathcal{D}(0, \mathbf{c}) = 0$ for any \mathbf{c} . As γ grows and PCAPS becomes “more carbon-aware”, $\mathcal{D}(\gamma, \mathbf{c})$ grows and CSF increases accordingly.

We also analyze the *carbon savings* of PCAPS. Suppose PB’s schedule finishes at time T , and PCAPS’s finishes at time T' (where $T \leq T'$). We let W denote the *excess work* that PCAPS must “make up” with respect to PB’s schedule (i.e., due to deferrals) – note that this implicitly depends on the carbon stretch factor. We let $\bar{s}_-^{(0,T)}$, $\bar{s}_+^{(0,T)}$, and $\bar{c}^{(T,T')}$ denote weighted average carbon intensity values based on the schedules of PB and PCAPS. In short, $\bar{s}_-^{(0,T)}$ captures the carbon emissions that PCAPS avoids due to deferrals between time 0 and time T , $\bar{s}_+^{(0,T)}$ captures the extra carbon (if any) incurred by PCAPS due to higher utilization relative to PB between time 0 and time T , while $\bar{c}^{(T,T')}$ captures the emissions that PCAPS incurs after time T . See Appendix B.1.1 for formal definitions of these quantities.

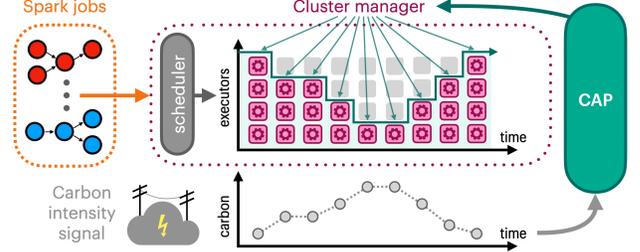


Figure 4: The CAP (Carbon-Aware Provisioning) module interacts directly with a *cluster manager* to specify the *amount of resources* (e.g., no. of machines) that can be used at any given time, based on a *carbon intensity signal*. CAP can be implemented without changes to an existing scheduling policy and/or the cluster manager.

Theorem 4.4. *For time-varying carbon intensities given by \mathbf{c} , PCAPS yields carbon savings of $W(\bar{s}_-^{(0,T)} - \bar{s}_+^{(0,T)} - \bar{c}^{(T,T')})$.*

Taken together, Theorem 4.3 and 4.4 characterize the carbon-time trade-off for PCAPS, implying that a larger CSF unlocks greater potential carbon savings.

4.2 Carbon-aware provisioning (CAP)

While PCAPS captures all three intuition points in Section 3 by interfacing with a probabilistic scheduler, many existing data processing schedulers use simple policies such as FIFO [22]. This naturally prompts the question of how PCAPS can be *simplified* to retain many of the same qualities, while inter-operating with *any* scheduler. In particular, PCAPS implicitly performs *resource provisioning*, changing the amount of resources available to the cluster based on carbon – this is a key technique used by prior work in carbon-aware scheduling [28, 54]. In this section, we introduce CAP (Carbon-Aware Provisioning), a simplified policy that applies a time-varying resource quota to the cluster and coexists with any underlying scheduler. In what follows, we motivate the design and discuss analytical results on its carbon-JCT trade-off.

CAP design. Given a cluster with K machines, the possible resource quotas are given by $\{0, 1, \dots, K\}$. This set calls to mind the k -search problem [45], where an online player must choose when to purchase k items over a deadline-constrained sequence of time-varying prices. Variants of k -search have been applied to carbon-aware scheduling with deadlines [27, 33]. CAP uses the k -search threshold set, which captures the trade-off between executing now and waiting for better carbon intensities. Instead of using a deadline, CAP frames the problem of determining a resource quota as *repeated rounds* of $(K - B)$ -search, where a minimum quota $B \in \{1, \dots, K\}$ always allows the cluster to use $\leq B$ machines, ensuring continuous progress on jobs. The thresholds are given by:

$$\Phi_B = U; \Phi_{i+B} = U - \left(U - \frac{U}{\alpha} \right) \left(1 + \frac{1}{(K-B)\alpha} \right)^{i-1} : i \in \{1, \dots, K-B\},$$

where α is the solution to $\left(1 + \frac{1}{(K-B)\alpha} \right)^{(K-B)} = \frac{U-L}{U(1-\frac{1}{\alpha})}$. Each

of these thresholds corresponds to a carbon intensity, and a quota is set based on how many values are *above* the current carbon intensity. Formally, the resource quota at time t is $r(t) \leftarrow \arg \max_{i \in \mathcal{R}} \Phi_i : \Phi_i \leq c(t)$. For ease of implementation, this quota is enforced without preemption; when machines become available, new task assignments are only allowed if $r(t)$ is greater than the number of busy machines.

Analytical results. We analyze the *carbon stretch factor* (Def. 3.1) and *carbon savings* (Def. 3.2) for CAP, with full proofs in Appendix B.2. AG denotes a carbon-agnostic baseline scheduler throughout. Suppose CAP’s schedule completes at time T' . We let $\mathcal{M}(B, \mathbf{c})$ denote the minimum resource cap specified by CAP at any point in its schedule (note this depends on the carbon signal \mathbf{c}). Formally, $\mathcal{M}(B, \mathbf{c}) := \arg \max_{i \in [K]} \Phi_i : \Phi_i \leq c(t) \forall t \in [0, T']$.

Theorem 4.5. *For time-varying carbon intensities given by \mathbf{c} , the carbon stretch factor of CAP is $\left(\frac{K}{\mathcal{M}(B, \mathbf{c})}\right)^2 \frac{2\mathcal{M}(B, \mathbf{c}) - 1}{2K - 1}$.*

We also analyze the *carbon savings* of CAP. If AG’s schedule finishes at time T (where $T \leq T'$), we use W as shorthand to denote the *excess work* that CAP must complete after time T (i.e., after AG has completed). As in Theorem 4.4, we let $\bar{s}^{(0, T)}$ and $\bar{c}^{(T, T')}$ denote weighted average carbon intensity values based on the schedules of AG and CAP, respectively – in short, $\bar{s}^{(0, T)}$ captures the carbon emissions that CAP avoids by deferring W amount of work relative to AG, while $\bar{c}^{(T, T')}$ captures the emissions that CAP incurs after time T . See Appendix B.2.2 for formal definitions of all three quantities.

Theorem 4.6. *For time-varying carbon intensities given by \mathbf{c} , CAP yields carbon savings of $W(\bar{s}^{(0, T)} - \bar{c}^{(T, T')})$.*

Theorem 4.5 and 4.6 imply that a larger CSF unlocks greater carbon savings for CAP. We explore the relative performance of PCAPS vs. CAP in our experiments, in Section 6.

5 Implementation

We have implemented proof-of-concepts of PCAPS and CAP for Apache Spark on Kubernetes – see Section 5.1 for details. We also conduct large-scale experiments in a realistic Spark simulator – see Section 5.2 for how we extend an existing simulator [48] to evaluate carbon-aware scheduling policies.

5.1 Spark and Kubernetes integration

Resource scaling & stage scheduling. In Spark deployed on a Kubernetes cluster, each application is submitted to the API server [6] that creates a “driver” running in a pod. We use Spark’s dynamic allocation feature, which enables the driver to create executor pods dynamically as needed by the application – these executors connect with the driver and execute application code. Kubernetes handles the scheduling of (driver and executor) pods for each application, while the Spark driver selects stages to execute within an application.

To implement CAP, we develop a Python daemon that gets carbon intensity from an API (e.g., Electricity Maps [18]) and adjusts the resources available to Spark. CAP sets a *resource quota* [2] within a dedicated namespace for Spark apps – our implementation adjusts *CPU and memory* quotas to correspond with a maximum number of executors. When the quota is *lowered*, existing pods are *not* preempted, but new pods are not scheduled until usage falls below the quota. We implemented PCAPS as a pluggable scheduling service that coordinates between Spark and Kubernetes. The service includes inference for Decima [48]. PCAPS gets carbon intensity from an API and collects context about the cluster and job states from Kubernetes and Spark.

While CAP can be implemented without modifications to Spark or Kubernetes, we made two key changes for PCAPS. First, we implemented a Kubernetes *scheduler plugin* [12] that communicates with PCAPS to determine which application should receive available resources. This builds on source code APIs exposed by the default kube-scheduler and requires building/configuring a custom scheduler pod. We restrict the scope of our plugin to a dedicated namespace for Spark apps. Next, we made changes to Spark [67] such that each application communicates with PCAPS before choosing the next stage for execution – Spark provides scripts to build a pod Docker image [50] based on a custom build.

Setting level of parallelism. In a Spark DAG, each *stage* (i.e., node) includes multiple tasks that are parallelizable over multiple executors. Setting a *parallelism limit* (number of executors working on a stage) is a key component of Spark scheduling (e.g., see [48, Section 5.2]). More executors are not necessarily better: assigning many executors to a stage that does not benefit from parallelism *blocks* them from working on other jobs in the queue. For *carbon-aware* scheduling, we enable PCAPS and CAP to set new parallelism limits for the current job each time a stage is scheduled, and particularly to set *lower* limits during high-carbon periods (e.g., see conditions **i**) and **ii**), Section 3).

In PCAPS, if a stage is deferred, it *idles* (see Alg. 1) the newly freed executors that prompted a scheduling event. Otherwise, the stage’s parallelism limit is set to $P' := \lceil P \cdot \min\{\exp(\gamma(L - c_t)), (1 - \gamma)\} \rceil$, where P is the limit chosen by Decima. This mirrors the exponential trade-off in PCAPS’s design – e.g., when the current carbon c_t is close to L the limit is set to $\lceil (1 - \gamma)P \rceil$, and as c_t grows, the limit decreases exponentially to 1. For CAP, given that the underlying scheduler specifies a parallelism limit P , CAP first attempts to schedule a stage with $P' = \lceil P \cdot r(t)/K \rceil$, where $r(t)/K$ is the ratio of the resource quota vs. the total number of executors. If the number of available executors is less than P' , the current stage takes all of the remaining available executors.

5.2 Spark simulator environment

Mao et al. [48] developed a simulator that is a faithful rep-

Table 1: Summary of carbon intensity trace characteristics, including the duration, granularity, minimum, maximum, mean, and coefficient of variation (*higher value implies more variation*) for carbon intensities.

Grid Code	Avg. Carbon Intensity (in $gCO_2eq./kWh$) [18]				
	Duration	Min.	Max.	Mean	Coeff. Var.
PJM	01/01/2020-	293	567	425	0.110
CAISO	12/31/2022	83	451	274	0.309
ON	Hourly	12	179	50	0.654
DE	granularity	130	765	440	0.280
NSW	26,304	267	817	647	0.143
ZA	data points	586	785	713	0.046

resentation of Spark’s *standalone mode* (i.e., where Spark is the cluster manager), achieving an error (i.e., in run times) of within 5% [48, Fig. 18]. This simulator captures all first-order effects of Spark execution (e.g., delays in executor movement, parallelism overheads) – it has since seen wide use in Spark contexts [3, 23, 29, 40, 49, 56]. We implement PCAPS and CAP as extensions to this simulator, which provides fast evaluation and flexibility. We make the following modifications:

► *Carbon accounting*: Each job’s carbon footprint is measured *ex post facto* to avoid impacting simulator fidelity. Once an experiment is complete, existing computations (e.g., executor times) and a carbon trace are used to tally the footprint.

► *CAP*: We implement CAP as a wrapper over three carbon-agnostic schedulers in the simulator: FIFO, Decima, and Weighted Fair (a heuristic tuned for the simulator’s test jobs).

► *PCAPS*: We implement PCAPS to interface with Decima, which provides a probability distribution over tasks.

With these modifications, the simulator allows us to quickly test many scenarios with a high degree of accuracy.

6 Evaluation

We evaluate our carbon-aware schedulers in a prototype cluster and a realistic Spark simulator, using workloads from TPC-H benchmarks [60] and Alibaba production DAG traces [1]. We answer the following questions:

1. How do PCAPS and CAP navigate the trade-off between carbon emissions and job completion time?
2. How do PCAPS and CAP adapt to changes in carbon intensity characteristics and workload characteristics?

6.1 Experimental setup

Carbon intensity traces. We use historical carbon traces from six regions [18] – each trace provides hourly carbon intensity data in grams of CO_2 equivalent per kilowatt-hour ($gCO_2eq./kWh$). The chosen power grids represent different energy generation mixes and thus different characteristics in

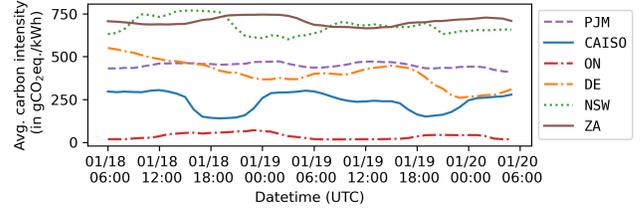


Figure 5: Time-varying carbon intensity for six grids (detailed in Table 1) over 48 hours in January 2021.

terms of average carbon intensity and variability; we evaluate how these impact the behavior of PCAPS and CAP. In Table 1 and Fig. 5, we give snapshots of each region, showing how grid characteristics impact time-varying carbon intensity. Larger *coefficients of variation* (the ratio of the standard deviation to the mean) correspond to greater renewable penetration – for instance, a large fraction of CAISO’s capacity is solar PV, while the capacity in ZA is predominantly coal.

To better observe the behavior of our carbon-aware schedulers, we follow prior work [24] and scale time in our experiments such that 1 minute of *real time* corresponds to 1 hour of *experiment time* – since carbon intensity is reported hourly, this approximates a scenario where each job works with large amounts of data and runs for several hours, as is becoming common in e.g., data curation for LLMs [7, 8, 9, 44, 61].

Workload traces. For workloads, we use TPC-H benchmarks [60] and real DAG traces from a production Alibaba cluster [1]. We construct workloads such that the inter-arrival times follow a Poisson distribution while specific jobs are randomly picked from the respective traces. In the main body, we consider an average inter-arrival time of 30 minutes (30 real-time seconds), with additional experiments measuring the impact of this parameter in Appendix A.2.

The TPC-H queries we experiment with operate on synthetic data with scales of 2 GB, 10 GB, and 50 GB – these correspond to average real durations of 180 seconds, 386 seconds, and 1,261 seconds when given a single executor. In our prototype experiments, we also construct workloads based on DAG information from the Alibaba trace [1]. These DAGs exhibit a realistic power law distribution (many DAGs of short duration, few DAGs of long duration), they have 66 nodes on average, and an average total duration (on one executor) of 7,989 seconds. We scale all durations by $1/60$ to match our experiment scale – this yields jobs that take 2.2 real-time minutes to complete on average.

In the simulator, each experiment is run over a full carbon trace (spanning three years of data). In the prototype, each experiment is run for several trials, where each starts at a uniformly randomly chosen time in the carbon trace, and new carbon intensities are reported once per (real-time) minute. In both implementations, the upper and lower bounds of U and L correspond to the maximum and minimum forecasted carbon intensities over a lookahead window of 48 hours.

Baselines. We compare against the following baselines:

- **Default Spark/Kubernetes behavior (default):** The

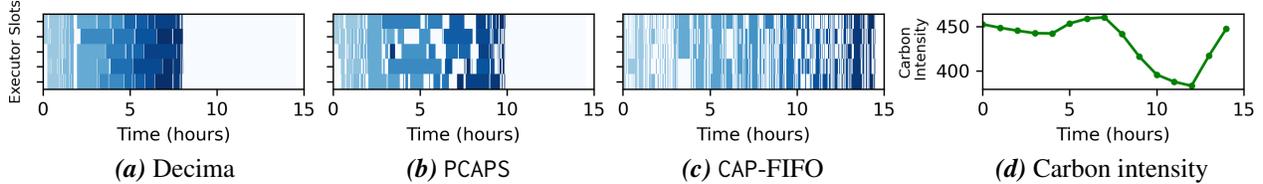


Figure 6: Visualizing executor usage over time for three schedulers, (a) Decima, (b) PCAPS, and (c) CAP-FIFO in a small simulator cluster with 5 executors and 20 TPC-H jobs, over a 15 hour period in the DE grid. (d) In the executor plots, each job is a unique shade of blue, while “idle” executors are indicated by a white background.

default behavior of Spark on Kubernetes – Spark uses first in, first out (FIFO) to choose stages within a job, while the Kubernetes scheduler mediates between pods of each job during execution [21]. In the simulator’s Spark standalone mode, this baseline implements only the FIFO scheduling.

► **Decima**: An RL scheduler for Spark that is optimized for job completion time [48]. We use the simulator’s training environment to train Decima for 20,000 epochs.

► **Weighted Fair**: A heuristic that assigns executors proportionally to each job’s workload, with tuned weights to improve performance on the simulated workloads [48].

► **GreenHadoop**: A MapReduce framework proposed to leverage green energy by matching workloads with the availability of solar [24]. This framework predates Spark, so we adapt its key ideas for DAG scheduling in the simulator – see Appendix A.1 for implementation details.

Metrics. We use three metrics to evaluate the fidelity of our approach in reducing carbon footprint without significantly impacting the completion time.

► **Carbon Footprint**: We report the carbon footprint for various scheduling policies as a percentage decrease of the carbon-agnostic default baseline unless stated otherwise. The values are in the range of $[-100\%, \infty)$, with negative values indicating a carbon reduction and positive values indicating an increase relative to the baseline.

► **Job Completion Time (JCT)**: We report the average job completion time across all the jobs in each experimental run. We report JCT as a fraction of the average JCT for the carbon-agnostic default baseline unless stated otherwise. The values can be in the $(0, \infty)$ range, with below 1 indicating a reduction in JCT and above 1 indicating an increase in JCT.

► **End-to-end Completion Time (ECT)**: We report the total time to complete all the jobs in a given experiment as the end-to-end completion time (ECT) as a fraction of the ECT for carbon-agnostic default. Its values lie in the same range as JCT. However, while JCT focuses on individual jobs, ECT represents the system’s throughput and efficiency. Also, as PCAPS and CAP focus on minimizing the total carbon footprint for a set of jobs and not the instantaneous rate of carbon consumption, ECT is a better metric for performance with respect to time in our case.

6.2 Carbon-aware schedulers in action

Before moving to our main results, we demonstrate the carbon-aware behavior of our schedulers in Fig. 6, which visualizes the schedules generated by Decima, PCAPS and CAP-FIFO during a short period in the DE grid on the simulator. PCAPS makes fine-grained scheduling decisions, idling specific executors during the high-carbon period ($t = (5, 8)$) while keeping bottleneck tasks running, achieving the lowest carbon footprint of the three schedules shown. This is in contrast to CAP-FIFO, which applies a resource quota uniformly across the cluster without consideration of bottlenecks – note the gaps in CAP-FIFO’s schedule that are straight vertical lines across multiple executors. Just before $t = 5$, a large gap in the schedule indicates that CAP-FIFO cannot run tasks because it did not prioritize bottleneck tasks early on.

Table 2: Summary of prototype results averaged over all six carbon traces. Each metric is normalized with respect to the Spark / Kubernetes default. PCAPS and CAP are configured to be moderately carbon aware.

<i>Metric normalized w.r.t. Default</i>	Default	Decima [48]	CAP	PCAPS
CO ₂ Reduction (%)	0%	1.2%	24.7%	32.9%
Avg. ECT	1.0	0.857	1.126	1.013
Avg. JCT	1.0	0.852	1.996	1.381

6.3 Prototype experiments

Our prototype is deployed on an OpenStack cluster running Kubernetes v1.31 and Spark v3.5.3 (both modified per Section 5) in Chameleon Cloud [31]. Our testbed consists of 51 m1.xlarge virtual machines, each with 8 VCPUs and 16GB of RAM. One VM is designated as the control plane node, while the remaining 50 are workers, each hosting two executor pods. Our Spark configuration allocates 4 VCPUs and 7GB of RAM to each of the 100 executors². To avoid a known issue with Spark’s dynamic allocation feature that can cause it to hang on Kubernetes [21], we configure an upper limit of 25 executors that can be allocated to any single job. We implement a carbon intensity API that replays historical traces to test our carbon-aware schedulers in the prototype. In our prototype, we implement default and Decima as the

²Spark’s default *memory overhead factor* is 10%. The difference between 7GB ($\times 2$) and the 16GB RAM of each worker is to accommodate this [20].

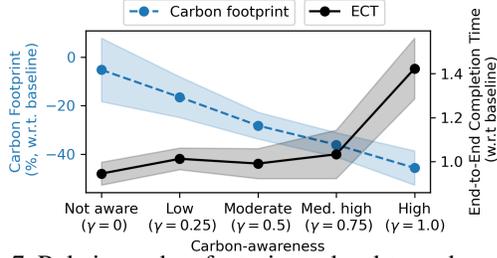


Figure 7: Relative carbon footprint and end-to-end completion times (w.r.t. the Spark/Kubernetes default) for PCAPS in the prototype, with five different degrees of carbon-awareness (γ). The shaded region denotes the standard deviation across 10 random trials.

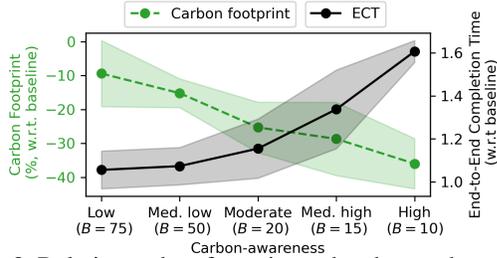


Figure 8: Relative carbon footprint and end-to-end completion times (w.r.t. the Spark/Kubernetes default) for CAP in the prototype, with five different degrees of carbon-awareness (B). The shaded region denotes the standard deviation across 10 random trials.

baselines. Unless stated otherwise, the results are averaged over the batch sizes of 25, 50, and 100 jobs. Furthermore, the results for each experimental configuration are averaged over 10 trials.

Results. Table 2 presents the results for our prototype experiments. PCAPS and CAP configured to be *moderately carbon-aware* (CAP is configured with $B = 20$ and PCAPS is configured with $\gamma = 0.5$) achieve average carbon reductions of 32.8% and 24.6% compared to the default baseline, respectively. Compared to Decima, PCAPS reduces carbon by 32.1%.

In terms of JCT, PCAPS performs significantly worse than the default and Decima baselines; JCT increases by 38.1% and 62% with respect to the default and Decima, respectively. This is expected since Decima targets minimizing average JCT. However, the key objective of PCAPS is to reduce the total carbon footprint without increasing ECT. PCAPS increases average ECT by only 12.4% compared to Decima and 1.3% compared to the default. CAP also performs well and increases average ECT by 12.6% on top of the default.

Trade-offs between carbon and job completion time. We next test several parameter settings for PCAPS and CAP to configure their carbon awareness in the DE grid region with batches of 50 TPC-H or Alibaba jobs. Fig. 7 plots the carbon-time trade-off for five settings of PCAPS relative to the Spark/Kubernetes default. Increasing the carbon awareness of PCAPS improves carbon savings at the expense of longer ECT, with a most pronounced effect for values of γ

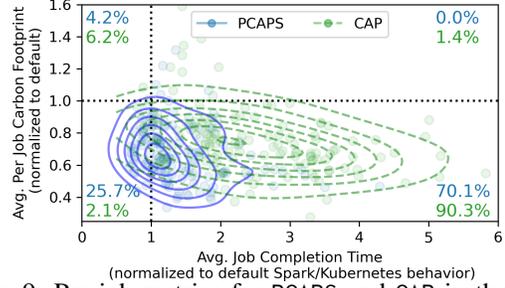


Figure 9: Per-job metrics for PCAPS and CAP in the prototype – each point represents a single trial’s average JCT and per-job carbon footprint. Trials are normalized so that the Spark/Kubernetes default is represented by (1, 1) – this splits the plot into quadrants, and each is annotated with the percentage of trials it holds (PCAPS is the top percentage, and CAP is the bottom). Contour lines outline a Gaussian KDE for each point cluster.

approaching 1. Conversely, Fig. 8 plots the same carbon-time trade-off for five settings of CAP; CAP sacrifices more in ECT (relative to PCAPS) for the same amount of carbon savings.

On a per-job level, we observe similar trends in Fig. 9, which plots average JCT and average *per-job* carbon footprint across trials of prototype experiments where PCAPS and CAP are configured to be moderately carbon-aware. Contour lines represent Gaussian kernel density estimators of the *outcome distribution* – note that the location of each “hot spot” represents the scheduler average. Splitting the plot into quadrants, we observe that PCAPS improves on the baseline scheduler’s per-job carbon footprint in 95.8% of trials, corresponding to the lower two quadrants. PCAPS improves on *both* carbon and completion time in 25.7% of cases, while CAP does so in only 2.1% of cases.

Effects of carbon intensity trace characteristics. Next, we analyze the effect of grid characteristics on the carbon-time trade-offs of our carbon-aware schedulers using subsets of each carbon trace (via 30 trials with 25, 50, and 100 jobs).

Fig. 10 plots the carbon reduction and average ECT of PCAPS, CAP, and Decima. Decima is carbon-agnostic and shows a minimal reduction in carbon that stays relatively constant across all regions. PCAPS and CAP incorporate grid behavior into their decisions, and we observe a positive relationship between the variability of a carbon trace and the resulting carbon reduction. For example, in ZA where carbon is relatively constant, high-carbon periods do not prompt PCAPS to defer tasks since the *future potential reductions* are insignificant. In contrast, high-carbon periods in the CAISO grid correspond to nighttime scenarios on the grid, where the prospect of daytime solar bolsters future potential reductions. These interactions are further illustrated through ECT – grid regions with more intermittent and variable energy mixes drive increases in ECT in exchange for more carbon reduction, since PCAPS and CAP *wait* for potential reductions.

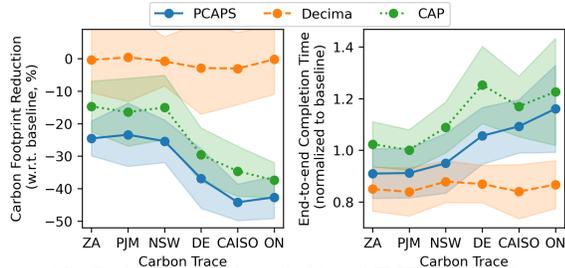


Figure 10: Carbon reduction (*left*) and ECT (*right*) for PCAPS, CAP, and Decima in six grid regions. Shaded regions denote standard deviation across 30 trials.

6.4 Simulator experiments

In the simulator, we evaluate PCAPS and CAP using TPC-H workloads, comparing them against **Weighted Fair** and **GreenHadoop** baselines, in addition to the Decima and default baselines from prototype experiments. We renamed the default baseline as FIFO for simulation-based experiments for accurate representation.

Simulator fidelity. To establish the fidelity of our simulator, we illustrate the granular effect of differences for a batch of 50 TPC-H jobs in [Appendix A.1.2](#). A notable difference between the prototype and the simulator is the relative performance of the main baseline (FIFO in the simulator, Spark/Kubernetes default in the prototype). In short, the simulator’s FIFO scheduler *over-assigns* executors to individual jobs, blocking others from entering service (thus increasing JCT) – this also increases its relative carbon footprint compared to the default behavior of our prototype.

Results. [Table 3](#) presents our top-line results showing PCAPS and CAP achieve significant reductions in carbon emissions compared to the baselines. Configured to be moderately carbon-aware, PCAPS achieves an average reduction of 23.1% compared to Decima, and a reduction of 39.7% compared to FIFO. CAP achieves an average carbon reduction of 22.7% when implemented on top of FIFO, 25.1% on top of Weighted Fair, and 14.5% on top of Decima. For 25, 50, and 100 jobs, PCAPS increases average end-to-end completion time (ECT) by 7.7% with respect to Decima, which is only a 4.5% degradation compared to FIFO. For CAP, average ECT increases by 10.8% when implemented on top of (and compared to) FIFO, 4.0% on top of Weighted Fair, and 9.3% on top of Decima.

At a per-job level, PCAPS increases the average job completion time (JCT) by 119.57% compared to Decima. Similarly, CAP increases average JCT by 127.4%, 86.6%, and 126.8% when implemented with FIFO, Weighted Fair, and Decima respectively. Larger increases in JCT relative to ECT happen because PCAPS allows more queue build-up during high-carbon periods, but “makes up for lost time” in low-carbon periods.

Trade-offs between carbon and job completion time. In the results summary, we observe positive carbon reduction in exchange for degradation in JCT and ECT. Since PCAPS and CAP can be configured to be more or less carbon-aware, we

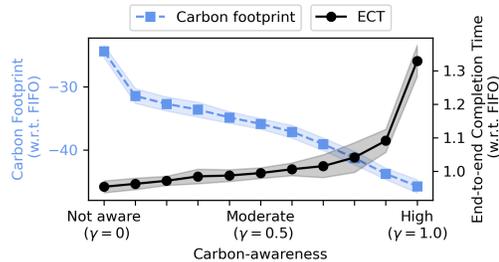


Figure 11: Relative carbon footprint and end-to-end completion times (with respect to FIFO) for PCAPS in simulator experiments, given different values of γ that correspond to degrees of carbon-awareness. Shaded region denotes standard deviation across carbon trace.

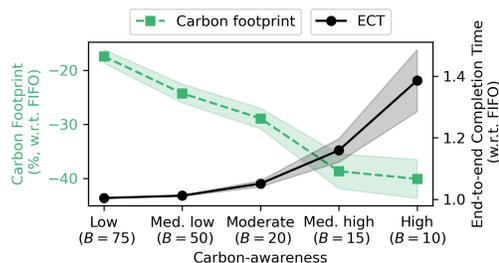


Figure 12: Relative carbon footprint and end-to-end completion times (with respect to FIFO) for CAP-FIFO in simulator experiments, given different values of B that correspond to degrees of carbon-awareness. Shaded region denotes standard deviation across carbon trace.

explore this *trade-off* in the DE grid with batches of 50 jobs. We vary hyperparameters γ (PCAPS) and B (CAP) to measure the impact of configuration on both carbon and JCT.

In [Fig. 11](#), we illustrate this trade-off for PCAPS compared against FIFO. As the carbon-awareness of PCAPS increases (indicated by the value of γ), the carbon savings of PCAPS improve at the expense of longer ECT. This effect is most pronounced for large values of γ approaching 1, because PCAPS defers many tasks to lower carbon periods. Conversely, [Fig. 12](#) illustrates the trade-off for CAP-FIFO, showing a similar trend of improving carbon at the expense of longer ECT. Compared to [Fig. 11](#), CAP-FIFO sacrifices more in terms of ECT for the same or lower amounts of carbon savings, and the increase in completion time begins earlier (at lower degrees of carbon-awareness).

Advantages of relative importance. Between PCAPS and CAP-Decima, the carbon-agnostic scheduler is identical – thus, performance differences can be attributed to the key ideas behind PCAPS, namely relative importance (see [Section 4.1](#)). In what follows, we examine this in detail using the DE grid region with batches of 50 jobs. We configure PCAPS and CAP-Decima with ten parameter settings for varying carbon-awareness. [Fig. 13](#) plots the result of this experiment, where each dot denotes the outcome of one trial. We fit a cubic polynomial to the outcomes of both methods to illustrate the key trend: PCAPS exhibits a strictly better trade-off between carbon footprint and ECT. For trials where either method achieves

Table 3: Summary of results for simulator experiments averaged over all 6 tested carbon traces. Each metric is normalized with respect to the default Spark FIFO behavior. PCAPS and CAP are configured to be moderately carbon-aware, and end-to-end completion time measures the total flow time for batches of jobs arriving continuously.

Metric (normalized with respect to FIFO)	FIFO	W. Fair	Decima [48]	GreenHadoop [24]	CAP			PCAPS
					FIFO	W. Fair	Decima	
Carbon Reduction (%)	0%	12.1%	21.5%	8.2%	22.7%	34.2%	31.1%	39.7%
Avg. End-to-End Completion Time	1.0	0.972	0.970	1.077	1.108	1.011	1.061	1.045
Avg. Job Completion Time	1.0	0.652	0.654	1.918	2.274	1.217	1.479	1.436

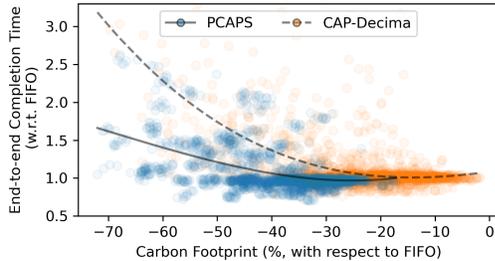


Figure 13: Relative carbon footprint vs. end-to-end completion time for PCAPS and CAP-Decima in simulator experiments, given varying parameters $\gamma \in [0.1, 1.0]$ and $B \in \{5, 10, \dots, 85\}$ that correspond to degrees of carbon-awareness. Each dot represents an individual trial, and lines represent a cubic polynomial of best fit.

carbon savings between 35% and 45%, PCAPS increases ECT by an average of 7.9%, while CAP-Decima increases it by an average of 42.7%. Conversely, for those trials where either method increases ECT by between 0% and 10%, PCAPS achieves average carbon savings of 35.6%, while CAP-Decima achieves an average savings of only 20.1%.

Effects of carbon intensity trace characteristics. The top-line results in Table 3 average over all six grid regions. We configure both schedulers to be moderately carbon-aware and characterize each carbon trace based on its *coefficient of variation*. In Fig. 14, we plot the carbon reduction and ECT for each of CAP-FIFO, PCAPS, and Decima. Decima’s carbon reduction relative to the default is higher relative to that observed in the prototype – this is due to differences between Spark’s standalone FIFO scheduler and the Spark/Kubernetes behavior of our prototype (see Appendix A.1.2). Similarly, PCAPS’s “baseline” carbon reductions increase alongside Decima’s, and CAP gains relative ground compared to CAP-FIFO in the simulator. We observe similar trends overall – grid regions with more intermittent and variable energy mixes due to renewables drive increases in both carbon reduction and ECT, as observed in Fig. 10.

6.5 Takeaways

Through evaluation in a realistic simulator and a prototype cluster, we show that a moderately carbon-aware PCAPS reduces carbon emissions by up to 39.7% in exchange for modest increases ($< 10\%$) in ECT for batches of 25, 50, and 100 data processing jobs. CAP configured to be moderately carbon-aware ($B = 20$) is also effective, reducing carbon by up to

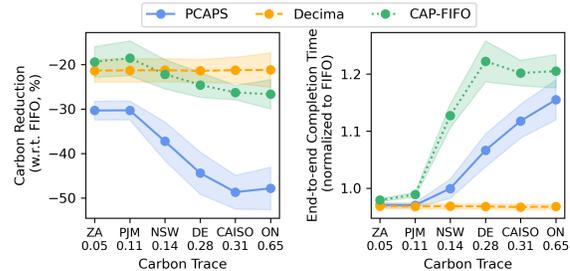


Figure 14: Carbon reduction (left) and increase in end-to-end completion time (right) for CAP, PCAPS, and Decima (relative to FIFO) in six grid regions. Shaded areas denote the standard deviation across a carbon trace.

25.1% with respect to the scheduler it is implemented on, in exchange for slightly larger increases in ECT. While CAP does not take dependencies into account and is suboptimal in terms of the carbon-time trade-off (see Fig. 13), it is easier to implement and more general than PCAPS. Intuitively, the performance of all carbon-aware techniques exhibits a dependence on the time-varying behavior of the power grid. Greater carbon savings can be achieved in regions with more renewables (thus more variability) in carbon intensity.

7 Conclusion

PCAPS demonstrates that an augmented scheduler that directly takes both the carbon cost of computation and precedence constraints (e.g., in the DAG of a data processing job) into consideration can achieve a favorable trade-off between carbon savings and completion time. Through experiments, we show that PCAPS’s configurability enables a scheduling policy that meaningfully reduces the carbon footprint of data processing without prohibitive increases in completion time. Furthermore, our detailed analytical and experimental study of CAP provides another avenue towards configurable carbon-aware scheduling that is broadly applicable and easy-to-implement.

References

- [1] Alibaba. Cluster data collected from production clusters in alibaba for cluster management research, 2018. URL <https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2018>.
- [2] The Kubernetes Authors. Resource Quotas – kuber-

- netes documentation. <https://kubernetes.io/docs/concepts/policy/resource-quotas/>, 2025. [Accessed 23-01-2025].
- [3] Vivek Bengre, M Reza HoseinyFarahabady, Mohammad Pivezhandi, Albert Y Zomaya, and Ali Jannesari. A learning-based scheduler for high volume processing in data warehouse using graph neural networks. In *International Conference on Parallel and Distributed Computing: Applications and Technologies*, pages 175–186. Springer, 2021.
- [4] Roozbeh Bostandoost, Walid A. Hanafy, Adam Lechowicz, Noman Bashir, Prashant Shenoy, and Mohammad Hajiesmaili. Data-driven Algorithm Selection for Carbon-Aware Scheduling. In *Proceedings of the 3rd Workshop on Sustainable Computer Systems, HotCarbon '24*, July 2024.
- [5] Roozbeh Bostandoost, Adam Lechowicz, Walid A. Hanafy, Noman Bashir, Prashant Shenoy, and Mohammad Hajiesmaili. LACS: Learning-Augmented Algorithms for Carbon-Aware Resource Scaling with Uncertain Demand. In *Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems, e-Energy '24*, page 27–45, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704802. doi: 10.1145/3632775.3661942. URL <https://doi.org/10.1145/3632775.3661942>.
- [6] Eric Brewer. Kubernetes and the path to cloud native. Santa Clara, CA, July 2015. USENIX Association.
- [7] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL <https://arxiv.org/abs/2005.14165>.
- [8] Maximilian Böther, Dan Graur, Xiaozhe Yao, and Ana Klimovic. Decluttering the data mess in llm training. Austin, 2024. HotInfra 2024. doi: 10.3929/ethz-b-000717691. 2nd Workshop on Hot Topics in System Infrastructure (HotInfra 2024); Conference Location: Austin, TX, USA; Conference Date: November 3, 2024.
- [9] Daoyuan Chen, Yilun Huang, Zhijian Ma, Hesun Chen, Xuchen Pan, Ce Ge, Dawei Gao, Yuexiang Xie, Zhaoyang Liu, Jinyang Gao, Yaliang Li, Bolin Ding, and Jingren Zhou. Data-juicer: A one-stop data processing system for large language models. In *Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS '24*, page 120–134, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704222. doi: 10.1145/3626246.3653385. URL <https://doi.org/10.1145/3626246.3653385>.
- [10] Runwei Cheng, Mitsuo Gen, and Yasuhiro Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms—i. representation. *Computers & industrial engineering*, 30(4):983–997, 1996.
- [11] Fabián A Chudak and David B Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Journal of Algorithms*, 30(2):323–343, February 1999. ISSN 0196-6774. doi: 10.1006/jagm.1998.0987. URL <http://dx.doi.org/10.1006/jagm.1998.0987>.
- [12] Kubernetes Community. Scheduler plugins, 2021. URL <https://github.com/kubernetes-sigs/scheduler-plugins>.
- [13] Mohammadreza Daneshvaramoli, Helia Karisani, Adam Lechowicz, Bo Sun, Cameron Musco, and Mohammad Hajiesmaili. Competitive algorithms for online knapsack with succinct predictions, 2024. URL <https://arxiv.org/abs/2406.18752>.
- [14] Sami Davies, Janardhan Kulkarni, Thomas Rothvoss, Jakub Tarnawski, and Yihao Zhang. Scheduling with communication delays via lp hierarchies and clustering, 2020. URL <https://arxiv.org/abs/2004.09682>.
- [15] Sami Davies, Janardhan Kulkarni, Thomas Rothvoss, Jakub Tarnawski, and Yihao Zhang. *Scheduling with Communication Delays via LP Hierarchies and Clustering II: Weighted Completion Times on Related Machines*, page 2958–2977. Society for Industrial and Applied Mathematics, January 2021. ISBN 9781611976465. doi: 10.1137/1.9781611976465.176. URL <http://dx.doi.org/10.1137/1.9781611976465.176>.
- [16] Lawrence Davis. Job shop scheduling with genetic algorithms. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, pages 136–140. Psychology Press, 2014.
- [17] Ran El-Yaniv, Amos Fiat, Richard M. Karp, and G. Turpin. Optimal Search and One-Way Trading Online Algorithms. *Algorithmica*, 30(1):101–139, May 2001.
- [18] Electricity Maps. Electricity Map. <https://www.electricitymap.org/map>, Accessed September 2023.

- [19] Jessica Fan, Werner Rehm, Giulia Siccardo, and McKinsey & Company. The state of internal carbon pricing. <https://www.mckinsey.com/capabilities/strategy-and-corporate-finance/our-insights/the-state-of-internal-carbon-pricing>, 2021.
- [20] The Apache Software Foundation. Configuration – Spark Documentation. <https://spark.apache.org/docs/3.5.3/configuration.html>, 2024. [Accessed 12-12-2024].
- [21] The Apache Software Foundation. Running Spark on Kubernetes – Spark Documentation. <https://spark.apache.org/docs/3.5.3/running-on-kubernetes.html>, 2024. [Accessed 12-12-2024].
- [22] The Apache Software Foundation. Job Scheduling – Spark Documentation. <https://spark.apache.org/docs/3.5.3/job-scheduling.html>, 2024. [Accessed 12-12-2024].
- [23] Arkadiy Gertsman. A faster reinforcement learning approach to efficient job scheduling in apache spark. Master’s thesis, University of Illinois at Urbana-Champaign, 2023.
- [24] Íñigo Goiri, Kien Le, Thu D. Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. GreenHadoop: Leveraging Green Energy in Data-Processing Frameworks. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys ’12, page 57–70, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312233. doi: 10.1145/2168836.2168843. URL <https://doi.org/10.1145/2168836.2168843>.
- [25] R. L. Graham. Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, 45(9):1563–1581, 1966. doi: 10.1002/j.1538-7305.1966.tb01709.x.
- [26] Nathan Grinsztajn, Olivier Beaumont, Emmanuel Jeanot, and Philippe Preux. Geometric deep reinforcement learning for dynamic dag scheduling. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, page 258–265. IEEE, December 2020. doi: 10.1109/ssci47803.2020.9308278. URL <http://dx.doi.org/10.1109/SSCI47803.2020.9308278>.
- [27] Walid A. Hanafy, Roozbeh Bostandoost, Noman Bashir, David Irwin, Mohammad Hajiesmaili, and Prashant Shenoy. The War of the Efficiencies: Understanding the Tension between Carbon and Energy Optimization. In *Proc. of the 2nd Workshop on Sustainable Computer Systems*. ACM, Jul 2023.
- [28] Walid A. Hanafy, Qianlin Liang, Noman Bashir, David Irwin, and Prashant Shenoy. CarbonScaler: Leveraging Cloud Workload Elasticity for Optimizing Carbon-Efficiency. *Proc. of the ACM on Measurement and Analysis of Computing Systems*, 7(3), Dec 2023.
- [29] Zhibo Hu, Chen Wang, Helen, Paik, Yanfeng Shu, and Liming Zhu. Learning interpretable scheduling algorithms for data processing clusters, 2024. URL <https://arxiv.org/abs/2405.19131>.
- [30] Muhammed Tawfiqul Islam, Shanika Karunasekera, and Rajkumar Buyya. Performance and cost-efficient spark job scheduling based on deep reinforcement learning in cloud computing environments. *IEEE Transactions on Parallel and Distributed Systems*, 33(7):1695–1710, 2021.
- [31] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. Lessons learned from the chameleon testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC ’20)*. USENIX Association, July 2020.
- [32] Alexandra Anna Lassota, Alexander Lindermayr, Nicole Megow, and Jens Schlöter. Minimalistic predictions to schedule jobs with online precedence constraints. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 18563–18583. PMLR, 23–29 Jul 2023. URL <https://proceedings.mlr.press/v202/lassota23a.html>.
- [33] Adam Lechowicz, Nicolas Christianson, Jinhang Zuo, Noman Bashir, Mohammad Hajiesmaili, Adam Wierman, and Prashant Shenoy. The Online Pause and Resume Problem: Optimal Algorithms and An Application to Carbon-Aware Load Shifting. *Proc. of the ACM on Measurement and Analysis of Computing Systems*, 7(3), Dec 2023.
- [34] Adam Lechowicz, Nicolas Christianson, Bo Sun, Noman Bashir, Mohammad Hajiesmaili, Adam Wierman, and Prashant Shenoy. Online Conversion with Switching Costs: Robust and Learning-augmented Algorithms. In *Proc. of the 2024 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, New York, NY, USA, June 2024. Association for Computing Machinery.

- [35] Adam Lechowicz, Nicolas Christianson, Bo Sun, Norman Bashir, Mohammad Hajiesmaili, Adam Wierman, and Prashant Shenoy. Chasing Convex Functions with Long-term Constraints. In *Proceedings of the 41st International Conference on Machine Learning, ICML'24*. JMLR, 2024.
- [36] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978. ISSN 0030364X, 15265463. URL <http://www.jstor.org/stable/169889>.
- [37] Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. Clover: Toward Sustainable AI with Carbon-Aware Machine Learning Inference Service. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '23*, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701092. doi: 10.1145/3581784.3607034. URL <https://doi.org/10.1145/3581784.3607034>.
- [38] Hongjian Li, Liang Lu, Wenhui Shi, Gangfan Tan, and Hao Luo. Energy-aware scheduling for spark job based on deep reinforcement learning in cloud. *Computing*, 105(8):1717–1743, March 2023. ISSN 1436-5057. doi: 10.1007/s00607-023-01171-z. URL <http://dx.doi.org/10.1007/s00607-023-01171-z>.
- [39] Shi Li. 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)*, page 283–294. IEEE, October 2017. doi: 10.1109/focs.2017.34. URL <http://dx.doi.org/10.1109/FOCS.2017.34>.
- [40] Xinran Li and Zhaohao Ding. Cost efficient job scheduling scheme for large scale data center. In *2023 IEEE/IAS Industrial and Commercial Power System Asia (I&CPS Asia)*, pages 2267–2272, 2023. doi: 10.1109/ICPSAsia58343.2023.10294452.
- [41] Xinyu Lin, Wenjie Wang, Yongqi Li, Shuo Yang, Fuli Feng, Yinwei Wei, and Tat-Seng Chua. Data-efficient fine-tuning for llm-based recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '24*, page 365–374, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3657807. URL <https://doi.org/10.1145/3626772.3657807>.
- [42] Qidong Liu, Xian Wu, Xiangyu Zhao, Yuanshao Zhu, Derong Xu, Feng Tian, and Yefeng Zheng. When moe meets llms: Parameter efficient fine-tuning for multi-task medical applications. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '24*, page 1104–1114, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3657722. URL <https://doi.org/10.1145/3626772.3657722>.
- [43] Wenyu Liu, Yuejun Yan, Yimeng Sun, Hongju Mao, Ming Cheng, Peng Wang, and Zhaohao Ding. Online job scheduling scheme for low-carbon data center operation: An information and energy nexus perspective. *Applied Energy*, 338:120918, 2023.
- [44] Yiheng Liu, Hao He, Tianle Han, Xu Zhang, Mengyuan Liu, Jiaming Tian, Yutong Zhang, Jiaqi Wang, Xiaohui Gao, Tianyang Zhong, Yi Pan, Shaochen Xu, Zihao Wu, Zhengliang Liu, Xin Zhang, Shu Zhang, Xintao Hu, Tuo Zhang, Ning Qiang, Tianming Liu, and Bao Ge. Understanding llms: A comprehensive overview from training to inference, 2024. URL <https://arxiv.org/abs/2401.02038>.
- [45] Julian Lorenz, Konstantinos Panagiotou, and Angelika Steger. Optimal Algorithms for k-Search with Application in Option Pricing. *Algorithmica*, 55(2):311–328, August 2008.
- [46] Thodoris Lykouris and Sergei Vassilvtiskii. Competitive Caching with Machine Learned Advice. In Jennifer Dy and Andreas Krause, editors, *Proc. of the 35th International Conference on Machine Learning*, volume 80 of *Proc. of Machine Learning Research*, pages 3296–3305. PMLR, 10–15 Jul 2018.
- [47] Biswaroop Maiti, Rajmohan Rajaraman, David Stalf, Zoya Svitkina, and Aravindan Vijayaraghavan. Scheduling precedence-constrained jobs on related machines with communication delay, 2020. URL <https://arxiv.org/abs/2004.10776>.
- [48] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning Scheduling Algorithms for Data Processing Clusters. In *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM '19*, page 270–288, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359566. doi: 10.1145/3341302.3342080. URL <https://doi.org/10.1145/3341302.3342080>.
- [49] Yamini Mathur. Torgraphina: A scheduler for data processing during high-frequency job arrival using upside down reinforcement learning. Master’s thesis, Iowa State University, 2023.
- [50] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

- [51] Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. On the stability of fine-tuning {bert}: Misconceptions, explanations, and strong baselines. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=nzplWnVAyah>.
- [52] Ferdinando Pezzella, Gianluca Morganti, and Giampiero Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & operations research*, 35(10):3202–3212, 2008.
- [53] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving Online Algorithms via ML Predictions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
- [54] Ana Radovanovic, Ross Koningstein, Ian Schneider, Bokan Chen, Alexandre Duarte, Binz Roy, Diyue Xiao, Maya Haridasan, Patrick Hung, Nick Care, et al. Carbon-Aware Computing for Datacenters. *IEEE Transactions on Power Systems*, 2022.
- [55] Veronique Sels, Nele Gheysen, and Mario Vanhoucke. A comparison of priority rules for the job shop scheduling problem under different flow time-and tardiness-related objective functions. *International Journal of Production Research*, 50(15):4255–4270, 2012.
- [56] Jungeun Shin, Diana Arroyo, Asser Tantawi, Chen Wang, Alaa Youssef, and Rakesh Nagi. Cloud-native workflow scheduling using a hybrid priority rule, dynamic resource allocation, and dynamic task partition. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*, SoCC ’24, page 830–846, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400712869. doi: 10.1145/3698038.3698551. URL <https://doi.org/10.1145/3698038.3698551>.
- [57] Brad Smith and Microsoft Corporation. We’re increasing our carbon fee as we double down on sustainability. <https://blogs.microsoft.com/on-the-issues/2019/04/15/were-increasing-our-carbon-fee-as-we-double-down-on-sustainability/>, 2019.
- [58] Yu Su, Shai Vardi, Xiaoqi Ren, and Adam Wierman. Communication-aware scheduling of precedence-constrained tasks on related machines. *Operations Research Letters*, 51(6):709–716, 2023. ISSN 0167-6377. doi: <https://doi.org/10.1016/j.orl.2023.11.001>. URL <https://www.sciencedirect.com/science/article/pii/S0167637723001815>.
- [59] Yu Su, Vivek Anand, Jannie Yu, Jian Tan, and Adam Wierman. Learning-augmented energy-aware list scheduling for precedence-constrained tasks. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 2024. doi: 10.1145/3680278. URL <https://doi.org/10.1145/3680278>.
- [60] TPC-H. The tpc-h benchmarks, 2018. URL <https://www.tpc.org/tpch/>.
- [61] Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Will we run out of data? limits of llm scaling based on human-generated data, 2024. URL <https://arxiv.org/abs/2211.04325>.
- [62] WattTime. WattTime. <https://www.watttime.org>, Accessed June 2024.
- [63] Philipp Wiesner, Ilja Behnke, Dominik Scheinert, Kordian Gontarska, and Lauritz Thamsen. Let’s Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud. In *Proceedings of the 22nd International Middleware Conference*, Middleware ’21, page 260–272, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385343. doi: 10.1145/3464298.3493399. URL <https://doi.org/10.1145/3464298.3493399>.
- [64] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, et al. Sustainable AI: Environmental Implications, Challenges and Opportunities. *Proceedings of Machine Learning and Systems (MLSys)*, 4:795–813, 2022.
- [65] Qing Wu, Zhiwei Wu, Yuehui Zhuang, and Yuxia Cheng. Adaptive DAG Tasks Scheduling with Deep Reinforcement Learning, page 477–490. Springer International Publishing, 2018. ISBN 9783030050542. doi: 10.1007/978-3-030-05054-2_37. URL http://dx.doi.org/10.1007/978-3-030-05054-2_37.
- [66] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI’12, page 2, USA, 2012. USENIX Association.
- [67] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016. ISSN 0001-0782. doi: 10.1145/2934664. URL <https://doi.org/10.1145/2934664>.

- [68] Yunfan Zhou, Xijun Li, Jinhong Luo, Mingxuan Yuan, Jia Zeng, and Jianguo Yao. Learning to optimize dag scheduling in heterogeneous environment. In *2022 23rd IEEE International Conference on Mobile Data Management (MDM)*, pages 137–146, 2022. doi: 10.1109/MDM55031.2022.00040.
- [69] Yunhong Zhou, Deeparnab Chakrabarty, and Rajan Lukose. Budget Constrained Bidding in Keyword Auctions and Online Knapsack Problems. In *Lecture Notes in Computer Science*, pages 566–576. Springer Berlin Heidelberg, 2008.

Appendix

A Evaluation Supplements

In this section, we give additional experiment setup details and results that are deferred from the main body (i.e., in [Section 6](#))

A.1 Deferred setup details

A.1.1 GreenHadoop [24] adaptation and implementation

This implementation begins by considering green (renewable) energy and brown (non-renewable) energy values available in the carbon traces obtained from Electricity Maps [18]. The system derives a "green window" by iterating over future minutes and summing the portion of executor capacity that can be powered purely by renewable energy sources until it meets the outstanding workload. Similarly, a "brown window" is derived by computing the number of executor minutes to finish the outstanding workload, assuming the system is run at maximum executor capacity. These two windows bracket the best-case scenarios of optimizing purely for carbon and for time.

To balance carbon usage and JCT, the system derives a final window size via a convex combination of the green and brown windows, modulated by a tunable parameter, θ , which details the carbon-awareness of the system. By default, θ is set to 0.5, which balances between 0 (carbon-agnostic) and 1 (fully-carbon-aware) At each scheduling decision, the system utilizes all of the available green energy in the future timesteps, and computes the fraction of available brown energy required to complete all jobs by the end of our convex window. Within that determined executor limit, tasks are dispatched using a standard FIFO queue, similar to how inter-job dependencies are managed in Hadoop. While this approach separates the carbon-aware resource provisioning from the job ordering logic, it does not embed the carbon-importance of tasks within each job.

A.1.2 Differences between Spark standalone FIFO baseline and default Spark / Kubernetes behavior

In the main body, in [Section 6.3](#), we discuss some notable differences between the *baselines* in the prototype experiments and the simulator experiments. In the simulator experiments, we use the default Spark standalone cluster scheduler (i.e., FIFO) as a baseline. In contrast, as the prototype is implemented on top of Spark and Kubernetes, we use the combined default behavior of the Spark DAG scheduler and Kubernetes scheduler as a baseline.

The difference between these baselines is particularly pronounced when comparing e.g., the relative carbon footprint of Decima against the primary baseline – in the simulator, Decima’s carbon footprint is a 21.5% improvement over FIFO, while in the prototype Decima improves on the default behavior by only 1.2% – similar trends are evident in average job completion time. This difference in performance can be attributed to a difference in how per-job parallelism limits are set by the FIFO scheduler in the simulated Spark standalone environment vs. the Spark/Kubernetes behavior configured on our prototype cluster. In the Spark standalone mode of operation, the default FIFO behavior assigns up to N executors to each stage of a job, where N is the number of tasks within said stage. In contrast, our Spark/Kubernetes cluster scheduler is configured to assign up to 25 executors across all stages of any job to avoid an issue where executors from previously completed stages continue to use cluster resources, causing Spark to hang.

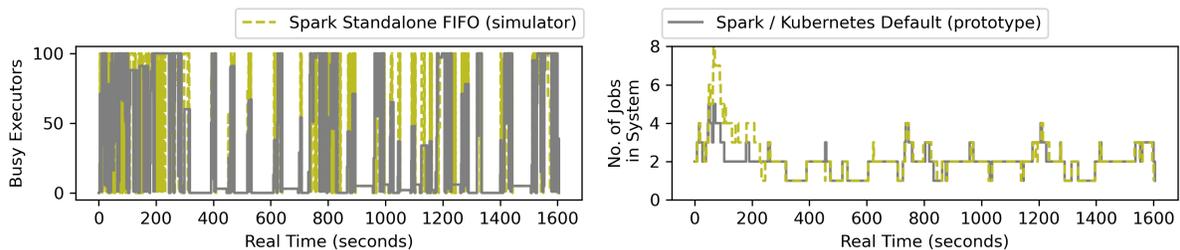


Figure 15: Executor usage (left) and number of jobs in the system (right) for an identical group of 50 TPC-H jobs in both the simulator (FIFO) and prototype (Spark / Kubernetes default) clusters, both with a maximum of 100 executors.

In [Fig. 15](#), we illustrate the granular effect of these different policies for an experiment of 50 TPC-H jobs with identical ordering and identical interarrival times in the simulator and prototype, respectively. The left-hand side plots the number of busy executors, while the right-hand side plots the number of jobs in the system. Note that the number of busy executors in the Spark /

Kubernetes prototype more frequently drops below 100, particularly when the number of jobs in the system is low (e.g., between 1000 - 1200 seconds). This corresponds to the executor cap configured for the Spark / Kubernetes default behavior, which results in more efficient executor usage and reduced blocking (i.e., when a job enters the system, it is more likely that there will be free executors to work on it). As a result, the Spark / Kubernetes behavior improves on the Spark standalone FIFO scheduler by 18.8% in carbon footprint and 22.1% in average job completion time in an experiment with 50 TPC-H jobs, mirroring the broader trends observed in our simulator and prototype experiments.

A.2 Deferred experiments

In this section, we present the results of additional experiments in the simulator and the prototype that are omitted from the main body due to space considerations.

A.2.1 Impact of total number of jobs

In the main results presented in Section 6, we evaluate the performance of tested algorithms under experiments with 25, 50, and 100 continuously arriving jobs. We explore the impact of varying this number of jobs in each experiment below, with PCAPS and CAP configured to be moderately carbon-aware, in the DE grid region.

Fig. 16 plots the average carbon reduction, end-to-end completion time, and average per-job completion time with respect to FIFO for PCAPS, Decima, and CAP on top of FIFO in the simulator environment, using experiments with 12, 25, 50, 100, and 200 jobs. We find that the relative ordering of all three techniques largely stays constant, although measuring results for a small number of jobs (e.g., 12, 25) is intuitively prone to more variance as the end-to-end results are more easily impacted by e.g., one or two random jobs. Out of the three metrics, carbon footprint is the most stable. As the number of jobs increases, all metrics “converge” to some average behavior. One interesting exception is CAP-FIFO – due to the “blocking” behavior of the FIFO scheduler in the simulator that is more prone to queue build-up (e.g., see Appendix A.1.2), a larger total number of jobs results in longer job completion times for CAP-FIFO.

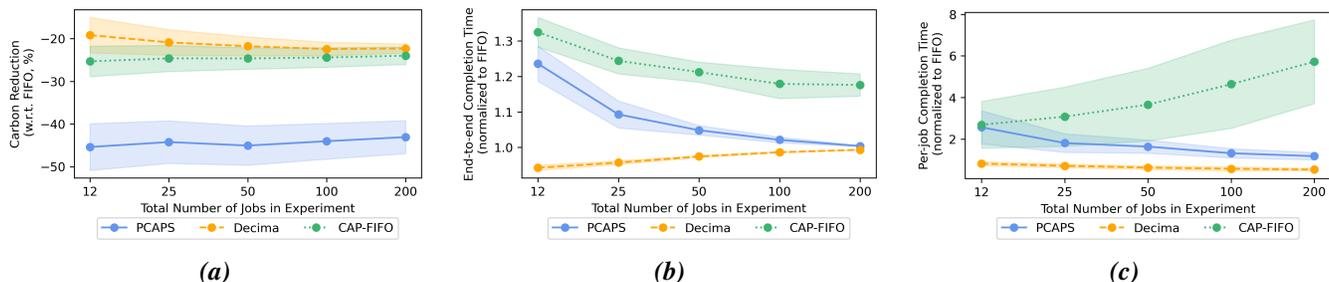


Figure 16: (a) Carbon reduction, (b) end-to-end completion time, and (c) average job completion time achieved by PCAPS, CAP-FIFO, and Decima (relative to FIFO) in a single grid region for varying experiment sizes. Shaded regions denote the standard deviation across the entire carbon trace.

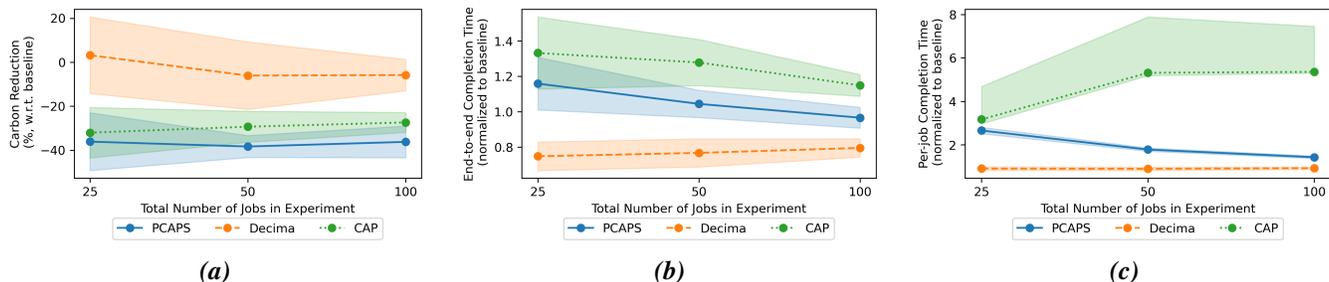


Figure 17: (a) Carbon reduction, (b) end-to-end completion time, and (c) average job completion time achieved by PCAPS, CAP, and Decima (relative to the Spark/Kubernetes default) in a single grid region for varying experiment sizes. Shaded regions denote the standard deviation across 10 random trials.

In Fig. 17, we plot the same metrics with respect to the default Spark/Kubernetes scheduler in the prototype implementation for PCAPS, Decima, and CAP, using experiments with 25, 50, and 100 jobs. These results generally mirror those shown in the

simulator above, although CAP implemented on top of the default Spark/Kubernetes behavior does not exhibit the same increase in per-job completion time for larger experiment sizes that CAP-FIFO does in the simulator – this is because the blocking behavior of FIFO is more pronounced than in the default Spark/Kubernetes scheduler (e.g., see [Appendix A.1.2](#)).

A.2.2 Impacts of submission rate

In the main results presented in [Section 6](#), we evaluate the performance of tested algorithms under continuous job arrivals following a Poisson process with an average interarrival time of $1/\lambda = 30$ minutes (30 seconds in real time). In what follows, we explore the impact of varying this interarrival time below, where note that smaller interarrival times imply that the cluster is more heavily utilized. PCAPS and CAP algorithms are both configured to be moderately carbon-aware, and the tested grid region is DE.

In [Fig. 18](#), we plot the average carbon reduction, end-to-end completion time, and average per-job completion time with respect to FIFO for PCAPS, Decima, and CAP-FIFO in the simulator environment. We find that the relative ordering of algorithm performance remain largely the same, but differences arise particularly for small interarrival times – in these scenarios where the cluster is more heavily utilized, we find that both PCAPS and Decima benefit from more intelligent scheduling decisions with respect to FIFO, which manifests in higher carbon reductions relative to FIFO and lower end-to-end completion times.

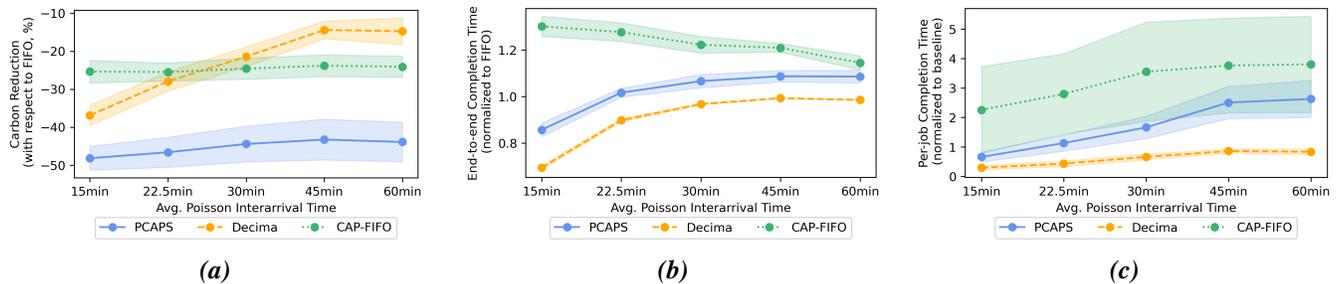


Figure 18: **(a)** Carbon reduction, **(b)** end-to-end completion time, and **(c)** average job completion time achieved by PCAPS, CAP-FIFO, and Decima (relative to FIFO) in a single grid region for varying Poisson interarrival times. Shaded regions denote the standard deviation across the entire carbon trace.

In [Fig. 19](#), we plot the same metrics with respect to the default Spark/Kubernetes scheduler in the prototype implementation for PCAPS, Decima, and CAP. These results largely mirror those shown in the simulator above – the most notable change is the improved performance of PCAPS and Decima (in terms of both end-to-end completion time and carbon footprint) for small interarrival times, i.e., when the cluster is heavily utilized.

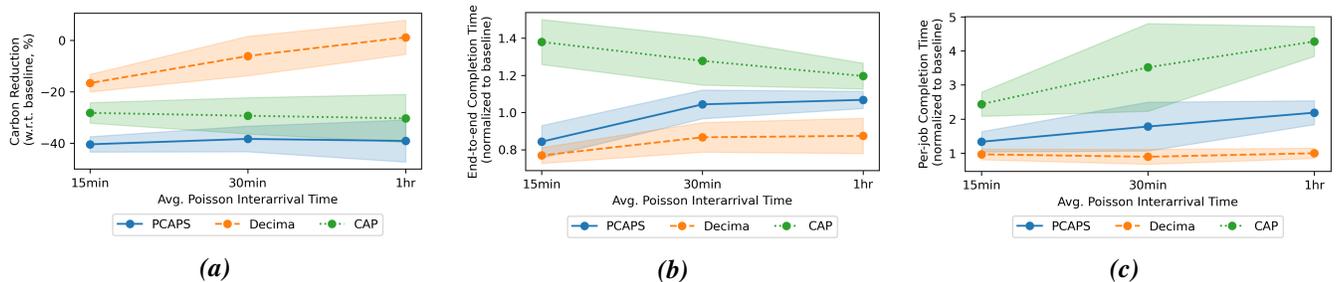


Figure 19: **(a)** Carbon reduction, **(b)** end-to-end completion time, and **(c)** average job completion time achieved by PCAPS, CAP, and Decima (relative to the Spark/Kubernetes default) in a single grid region for varying Poisson interarrival times. Shaded regions denote the standard deviation across 10 random trials.

A.2.3 Carbon-awareness logic latency

The logic of PCAPS and CAP naturally introduce latency due to the overhead required to make carbon-aware decisions. We quantify this *scheduling overhead* for FIFO, Decima, CAP-FIFO, and PCAPS in the simulator below, in the DE grid region over 1000 simulated trials. Note that we report the latency in real-time (i.e., without the scaling mentioned in [Section 6.1](#)). These measurements do not include the latency of e.g., calls to an external carbon intensity API.

In Fig. 20(a), we plot the average latency (in milliseconds) of invoking each scheduler once, given that there is exactly $\{1, 5, 10, 25, 50, 75, 100\}$ outstanding TPC-H jobs in the queue. We find that simple decision rule policies (FIFO and CAP-FIFO) exhibit consistently low latencies below 5 milliseconds, where CAP adds an overhead of a few milliseconds. In contrast, Decima and PCAPS, which both use a GNN+RL model to run inference at each invocation, intuitively exhibit latency that grows as a function of the number of outstanding jobs. Relative to Decima, PCAPS adds an overhead of a few milliseconds at each invocation, and this overhead is constant (i.e., it does not grow with the number of jobs in the queue).

In Fig. 20(b), we plot the average latency over a full experiment, where the initial number of jobs in the queue is one of $\{1, 5, 10, 25, 50, 75, 100\}$ – note that this latency goes down over time as jobs are completed. It is computed as a ratio between the total time spent in the scheduler and the number of scheduler invocations over the experiment length. The findings in this metric are largely similar, with lower overall averages due to the averaging over a full experiment (as opposed to averaging over a single queue length). Overall, these results imply that the latency impact of carbon-awareness is minimal – in the context of long-running big-data workloads, the sub-20 millisecond latencies observed are likely to be insignificant compared to other overheads on the cluster.

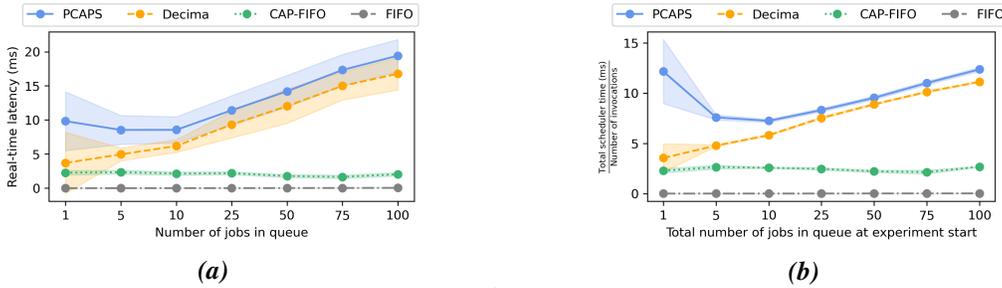


Figure 20: (a) Average latency with N jobs in the queue and (b) average normalized time in the scheduler for PCAPS, CAP-FIFO, Decima, and FIFO in a single grid region for varying experiment sizes. Shaded region denotes the standard deviation across all 1000 trials.

B Deferred Analytical Results and Discussion

In this section, we give full proofs for analytical results and detailed discussion for the PCAPS and CAP scheduler designs introduced in Section 4.

B.1 Deferred Proofs and Discussion from Section 4.1 (Precedence- and carbon-aware provisioning and scheduling)

In this section, we discuss and prove the analytical results for the PCAPS scheduler, introduced in Section 4.1. Throughout this section, we let PB denote a *carbon-agnostic, probabilistic* baseline scheduler, as outlined in Def. 4.1.

For the sake of analysis, in the following results we leverage the classic makespan bound of Graham’s list scheduling algorithm [25], which is known to produce a schedule that is a $(2 - 1/k)$ -approximation for the optimal makespan on K identical machines. Note that any carbon-agnostic probabilistic scheduler is an instance of list scheduling where the list (of tasks) is random, and the next task is assigned to a machine as soon as it becomes idle.

Recall the definition of PCAPS’s carbon-awareness filter, parameterized by the Ψ_γ function (see Algorithm 1). Ψ_γ exhibits an exponential dependence on r , the relative importance of a task. This is inspired by literature on one-way trading and related online problems [17, 69], where such an exponential trade-off is derived by balancing the marginal reward of the current price against the risk that better prices may arrive in the future. For DAG scheduling, we interpret relative importance in an analogous way: tasks with high importance have a substantial negative impact if they are not scheduled, so they are almost always scheduled at the current “price” (i.e., carbon intensity). On the other hand, tasks with low relative importance (e.g., short tasks) may not significantly affect the DAG’s completion time if they are deferred to start later, so they can “wait” for better carbon intensities.

To analyze the carbon stretch factor of PCAPS, we define $\mathcal{D}(\gamma, \mathbf{c}) \in [0, 1]$, a quantity that describes the fraction of tasks (in terms of total runtime) that are deferred by PCAPS when scheduling with a given value of γ and given carbon trace \mathbf{c} . It is ≤ 1 for any value of γ , and $\mathcal{D}(0, \mathbf{c}) = 0$ for any \mathbf{c} . As γ grows and PCAPS becomes “more carbon-aware”, $\mathcal{D}(\gamma, \mathbf{c})$ grows in expectation.

B.1.1 Proof of Theorem 4.3

We are ready to prove [Theorem 4.3](#), which states that for a given carbon intensity trace \mathbf{c} , PCAPS's carbon stretch factor is $1 + \frac{\mathcal{D}(\gamma, \mathbf{c})K}{2 - \frac{1}{K}}$.

Proof. Let $\text{PB}_K(\mathcal{J})$ denote the schedule produced by the carbon-agnostic probabilistic baseline scheduler (e.g., Decima) using K machines, and let $\text{PCAPS}_K(\mathcal{J})$ denote the schedule produced by PCAPS for the same job \mathcal{J} with n tasks and a maximum of K machines. We denote the carbon intensity trace by $\mathbf{c} := \{c(t) : t \geq 0\}$.

We henceforth use PB_K^{D} to denote the instance of PB that PCAPS interfaces with.

Consider the perspective of a single node v : suppose that PB samples node v to be scheduled at time $t \geq 0$, where node v has probability $p_{v,t} > 0$ and $v \in \mathcal{A}_t$. By definition of PB, as soon as a task (node) is sampled, it runs on the available machine in PB's schedule.

Now we consider the same node sampled by PB_K^{D} . We denote D_v as a random variable that denotes the number of times that node v is *not* scheduled (i.e., deferred) when it is sampled by PB_K^{D} . It is defined as:

$$D_v = \sum_{t \in \mathcal{T}_v} \mathbb{I}\{c(t) > \Psi_{\gamma}(r_{v,t})\},$$

where \mathcal{T}_v denotes the times at which node v is sampled by PB_K^{D} . In the worst-case, observe that whenever a task v is sampled but not scheduled, a deferral increases the total makespan by at most X_v , where X_v is the runtime of task v . Consider the edge case where all other tasks in \mathcal{A}_t (i.e., all other tasks that are ready to run at the same time as task v) are scheduled on other machines at time t , and all further tasks (i.e., tasks that have not yet been completed but also were not in \mathcal{A}_t) are successors of v (i.e., they cannot run until v is completed).

In this case, assuming that the other tasks in \mathcal{A}_t run to completion, there will be some time step $t' > t$ such that $\mathcal{A}_{t'} = \{v\}$ – i.e., the only task available to run is task v . As soon as this is the case, v will run – this is because the *relative importance* of any task in a set of size 1 is always 1. Thus, in the worst-case, the schedule length increases by exactly X_v – this is the case if all of the other tasks in \mathcal{A}_t finish at the same time t' (i.e., no overlap with v).

This gives the following makespan bound in terms of D_v :

$$\mathbb{E}[\text{PCAPS}_K(\mathcal{J})] \leq \mathbb{E}[\text{PB}_K(\mathcal{J})] + \mathbb{E}\left[\sum_{v \in \mathcal{J}} D_v X_v\right]$$

By linearity of expectation, we have:

$$\mathbb{E}[\text{PCAPS}_K(\mathcal{J})] \leq \mathbb{E}[\text{PB}_K(\mathcal{J})] + \sum_{v \in \mathcal{J}} \mathbb{E}[D_v] X_v$$

Consider the *total* number of deferrals $D = \sum_{v \in \mathcal{J}} D_v$, and note that D must be $\leq n - 1$ – since at least one machine is active at all times, the maximum number of deferrals is that which causes the schedule to drop to a single machine across the entire time interval. This immediately implies that $\mathbb{E}[D] \leq n - 1$. Define a sorted list $\{X'_i : i \in n\}$ such that $X'_0 = \max_{v \in \mathcal{A}} X_v, \dots, X'_n = \min_{v \in \mathcal{A}} X_v$, and note that we can upper bound the above as follows:

$$\mathbb{E}[\text{PB}_K(\mathcal{J})] + \sum_{v \in \mathcal{J}} \mathbb{E}[D_v] X_v \leq \mathbb{E}[\text{PB}_K(\mathcal{J})] + \sum_{i=0}^{\mathbb{E}[D]} X'_i,$$

Note that this is a worst-case assumption – in reality, the tasks with low relative importance (that are likely to be deferred) are unlikely to be the longest tasks for any reasonable scheduler PB. Note that $\sum_{i=0}^n X'_i = \sum_{v \in \mathcal{J}} X_v = \text{OPT}_1(\mathcal{J})$, i.e., the optimal makespan using a single machine. To simplify the above bound, we can define a function $\mathcal{D}(\gamma, \mathbf{c})$ as follows:

$$\mathcal{D}(\gamma, \mathbf{c}) = \frac{\sum_{i=0}^{\mathbb{E}[D]} X'_i}{\text{OPT}_1(\mathcal{J})}.$$

Note that $\mathcal{D}(\gamma, \mathbf{c}) \leq 1$ for any γ and any \mathbf{c} , and $\mathcal{D}(0, \mathbf{c}) = 0$ for any \mathbf{c} . In practice, $\mathbb{E}[D]$ can be estimated from e.g., historical carbon traces and the characteristic behavior of PB (i.e., in terms of relative importance). We have the following bound:

$$\mathbb{E}[\text{PCAPS}_K(\mathcal{J})] \leq \mathbb{E}[\text{PB}_K(\mathcal{J})] + \mathcal{D}(\gamma, \mathbf{c}) \text{OPT}_1(\mathcal{J}).$$

This gives insight into the tuning of hyperparameter γ – for a low-carbon period \mathbf{c}' where a practitioner desires full throughput, one should tune γ such that $\mathcal{D}(\gamma, \mathbf{c}') \approx 0$.

Since PB follows the list scheduling paradigm of scheduling tasks in an order that respects precedence constraints, it inherits the following worst-case theoretical bound on its makespan:

$$\mathbb{E}[\text{PB}_K(\mathcal{J})] \leq \left(2 - \frac{1}{K}\right) \text{OPT}_K(\mathcal{J}).$$

From the proof of [Theorem 4.5](#), we also have the following bound for $\text{OPT}_1(\mathcal{J})$:

$$\text{OPT}_1(\mathcal{J}) \leq K \cdot \text{OPT}_K(\mathcal{J}).$$

Combining these results, we have the following:

$$\mathbb{E}[\text{PCAPS}_K(\mathcal{J})] \leq \left(2 - \frac{1}{K} + \mathcal{D}(\gamma, \mathbf{c})K\right) \text{OPT}_K(\mathcal{J}).$$

Combined with the bound for PB, this shows that PCAPS has a carbon stretch factor of $1 + \frac{\mathcal{D}(\gamma, \mathbf{c})K}{2 - \frac{1}{K}}$. □

We now turn to the question of carbon savings, and the result stated in [Theorem 4.4](#). First, for ease of analysis, we define a *discretized time model* that is motivated by the empirical fact that carbon intensities are reported in discrete time intervals [[18](#), [62](#)]. Assuming that new carbon intensity values arrive at integers in continuous time, we define a discretized carbon signal for any discrete time step t as $c_t := c(t') : t' \in [t, t+1)$, where note that $c(t')$ does not change on the interval $t \in [t, t+1)$.

Slightly abusing notation, we let $C_{\text{PCAPS}}(t)$ denote the carbon emissions of PCAPS’s schedule during discrete time step t , and let $C_{\text{PB}}(t)$ denote the carbon emissions of PB at discrete time step t , respectively. The schedules generated by PCAPS and PB each use a certain number of machines during any discrete time interval – to capture this, we let $E_t^{\text{PCAPS}} : t' \in [t, t+1)$ denote the number of active machines during discrete time step t in PCAPS’s schedule, and $E_t^{\text{PB}} \leq K$ denotes the same for PB’s schedule. In what follows, we use W as shorthand to denote the *excess work* that PCAPS must “make up” with respect to PB’s schedule (i.e., due to deferral actions). Formally, $W = \sum_{i=0}^T \max\{E_i^{\text{PB}} - E_i^{\text{PCAPS}}, 0\}$.

B.1.2 Proof of [Theorem 4.4](#)

In what follows, we prove [Theorem 4.4](#), which states that for a given carbon intensity trace \mathbf{c} , PCAPS yields carbon savings of $W \left(\bar{s}_-^{(0,T)} - \bar{s}_+^{(0,T)} - \bar{c}^{(T,T')} \right)$ compared to a carbon-agnostic baseline PB.

Proof. We let $C_s(t)$ denote the *carbon savings* of PCAPS at discrete time step t . Formally, we have:

$$C_s(t) = \begin{cases} C_{\text{PB}}(t) - C_{\text{PCAPS}}(t) & 1 \leq t \leq T, \\ -C_{\text{PCAPS}}(t) & T < t \leq T'. \end{cases}$$

By definition, we have the following by substituting for the carbon emissions of PB and PCAPS:

$$C_s(t) = \begin{cases} E_t^{\text{PB}} c_t - E_t^{\text{PCAPS}} c_t & 1 \leq t \leq T, \\ -E_t^{\text{PCAPS}} c_t & T < t \leq T'. \end{cases}$$

Summing over all time steps, we have that the carbon savings is given by:

$$\sum_{i=0}^{T'} C_s(i) = \sum_{i=0}^T (E_i^{\text{PB}} - E_i^{\text{PCAPS}}) c_i - \sum_{i=T+1}^{T'} E_i^{\text{PCAPS}} c_i$$

Note that because of PCAPS’s job-specific decisions, it is not necessarily the case that $E_t^{\text{PB}} \geq E_t^{\text{PCAPS}}$ for any $t \leq T$ – namely, if PB’s schedule is constrained by a bottleneck task during a low-carbon time step, PCAPS’s schedule may use that low-carbon time step to schedule deferred tasks from previous time steps.

Thus, to begin simplifying this expression, we consider two cases for the sum from 0 to T as follows:

$$\begin{aligned} \sum_{i=0}^{T'} C_s(i) &= \sum_{i=0}^T (E_i^{\text{PB}} - E_i^{\text{PCAPS}}) c_i \mathbb{I}(E_i^{\text{PB}} \geq E_i^{\text{PCAPS}}), \\ &\quad - \sum_{i=0}^T (E_i^{\text{PCAPS}} - E_i^{\text{PB}}) c_i \mathbb{I}(E_i^{\text{PB}} < E_i^{\text{PCAPS}}) - \sum_{i=T+1}^{T'} E_i^{\text{PCAPS}} c_i. \end{aligned}$$

We define three terms that capture the *weighted average* carbon intensity per unit of work that is deferred, opportunistically completed, or completed later as follows. Note that $\sum_{i=0}^T (E_i^{\text{PB}} - E_i^{\text{PCAPS}}) = \sum_{i=T+1}^{T'} E_i^{\text{PCAPS}}$.

We let $\bar{s}_-^{(0,T)}$ denote the weighted average carbon intensity of the machine time (work) that is *deferred* in PCAPS's schedule (i.e., carbon saved due to PCAPS's carbon-aware filtering):

$$\bar{s}_-^{(0,T)} = \frac{\sum_{i=0}^T (E_i^{\text{PB}} - E_i^{\text{PCAPS}}) c_i \mathbb{I}(E_i^{\text{PB}} \geq E_i^{\text{PCAPS}})}{W}$$

Furthermore, we let $\bar{s}_+^{(0,T)}$ denote the weighted average carbon intensity of the machine time (work) that is *opportunistically completed* in PCAPS's schedule (i.e., when PCAPS does more work than PB, likely during a low-carbon period):

$$\bar{s}_+^{(0,T)} = \frac{\sum_{i=0}^T (E_i^{\text{PCAPS}} - E_i^{\text{PB}}) c_i \mathbb{I}(E_i^{\text{PB}} < E_i^{\text{PCAPS}})}{W}$$

Finally, we let $\bar{c}^{(T,T')}$ denote the weighted average carbon intensity of the work completed by PCAPS after time T :

$$\bar{c}^{(T,T')} = \frac{\sum_{i=T+1}^{T'} E_i^{\text{PCAPS}} c_i}{W}$$

Under the above definitions, we can pose the total carbon savings as:

$$\sum_{i=0}^{T'} C_s(i) = W \left(\bar{s}_-^{(0,T)} - \bar{s}_+^{(0,T)} - \bar{c}^{(T,T')} \right)$$

□

We note that an adversary could construct instances such that PCAPS uses more carbon than a carbon-agnostic baseline – for instance, consider the case where the carbon intensity at each time step is strictly increasing over time. In such a scenario, the “carbon-optimal” solution simply finishes the job as soon as it can, and such a scenario implies that $\bar{c}^{(T,T')} + \bar{s}_+^{(0,T)} > \bar{s}_-^{(0,T)}$, meaning that PCAPS's carbon savings are negative. However, we note such scenarios for the carbon intensity on the grid are unrealistic. In reality, grid carbon intensity exhibits *diurnal* (i.e., *daily*) *patterns* that mediate this effect – see Fig. 5 for an example.

The above result contextualizes the total carbon savings achieved by PCAPS for a single job, but we also consider the average carbon savings at each (discrete) carbon intensity interval as follows.

Let $\rho_{\text{PCAPS}}(c)$ denote the average machine utilization for PCAPS's schedule conditioned on the fact that the current carbon intensity is $c = c_t$. Denoting the set of discrete time steps with carbon intensity c by \mathcal{T}_c , we have the following: $\rho_{\text{PCAPS}}(c) = \lim_{T \rightarrow \infty} \frac{\sum_{i \in \mathcal{T}_c} E_i^{\text{PCAPS}} / K}{|\mathcal{T}_c|}$. Note that $\rho_{\text{PCAPS}}(c)$ can be estimated based on e.g., the observed relative importances of tasks produced by PB – this *distribution* of relative importances maps (via Ψ_γ) into a distribution of carbon intensity values – the fraction of these values that lie below c is proportional to $\rho_{\text{PCAPS}}(c)$, since the fraction of values above c correspond to tasks that would be deferred by PCAPS.

Corollary B.1. *In a setting where there are always jobs with outstanding tasks in the data processing queue, the average carbon savings of PCAPS at any given discrete time step t is given by $(\rho_{\text{PB}} K - \rho_{\text{PCAPS}}(c_t) K) c_t$.*

Proof. In this setting, the expression of the *average* carbon savings at any given discrete time step simplifies as follows:

Let ρ_{PB} denote the average machine utilization of PB's schedule, i.e., $\rho_{\text{PB}} = \lim_{T \rightarrow \infty} \frac{\sum_{i=0}^T E_i^{\text{PB}} / K}{T}$.

Then by Theorem 4.4, the average carbon savings \bar{C}_s at any time step t is given by the following:

$$\bar{C}_s(t) = (\rho_{\text{PB}} K - \rho_{\text{PCAPS}}(c_t) K) c_t.$$

□

B.2 Deferred Proofs and Discussion from Section 4.2 (Carbon-aware provisioning (CAP))

In this section, we discuss and prove the analytical results for CAP, introduced in Section 4.2. Throughout this section, we let AG denote a *carbon-agnostic* baseline scheduler.

For the sake of analysis, in the following results we leverage the classic makespan bound of Graham’s list scheduling algorithm [25], which is known to produce a schedule that is a $(2 - 1/K)$ -approximation for the optimal makespan on K identical machines. Note that FIFO is an instance of list scheduling where the list (of tasks) is a FIFO queue, and the next task is assigned to a machine as soon as it becomes idle.

Suppose that for a job \mathcal{J} , CAP’s schedule completes it at time $T' \geq 0$. Note that if $c(t) = L$ for all time steps, the schedule of CAP is identical to that of AG because CAP always sets $r(t) = K$. Otherwise, we let $\text{OPT}_K(\mathcal{J})$ denote the optimal makespan on K machines, and let $\mathcal{M}(B, \mathbf{c})$ denote the minimum resource cap specified by CAP at any point in its schedule (note this depends on the carbon signal \mathbf{c}). Formally, we let: $\mathcal{M}(B, \mathbf{c}) = \arg \max_{i \in [K]} \Phi_i : \Phi_i \leq c(t) \forall t \in [0, T']$.

B.2.1 Proof of Theorem 4.5

We are now ready to prove Theorem 4.5, which states that CAP’s carbon stretch factor is $\left(\frac{K}{\mathcal{M}(B, \mathbf{c})}\right)^2 \frac{2\mathcal{M}(B, \mathbf{c})-1}{2K-1}$. We prove the statement by first showing that CAP’s makespan is at most $\left(\frac{2K}{\mathcal{M}(B, \mathbf{c})} - \frac{K}{\mathcal{M}(B, \mathbf{c})^2}\right) \text{OPT}_K(\mathcal{J})$.

Proof. Let $\text{CAP}_K(\mathcal{J} \mid \mathcal{M}(B, \mathbf{c}))$ denote the makespan of CAP given K machines conditioned on the value of $\mathcal{M}(B, \mathbf{c})$, and let $\text{AG}_K(\mathcal{J})$ denote the makespan of AG (i.e., Graham’s list scheduling with K machines).

First, note that the following holds by [25]:

$$\text{AG}_K(\mathcal{J}) \leq \left(2 - \frac{1}{K}\right) \text{OPT}_K(\mathcal{J}).$$

Second, note that the makespan of $\text{CAP}_K(\mathcal{J} \mid \mathcal{M}(B, \mathbf{c}))$ is upper-bounded by that of $\text{AG}_{\mathcal{M}(B, \mathbf{c})}(\mathcal{J})$. By definition of $\mathcal{M}(B, \mathbf{c})$, CAP always has $\mathcal{M}(B, \mathbf{c})$ machines available, which is the same as $\text{AG}_{\mathcal{M}(B, \mathbf{c})}(\mathcal{J})$. If any other machines become available and process jobs during the schedule of $\text{CAP}_K(\mathcal{J} \mid \mathcal{M}(B, \mathbf{c}))$, these *strictly help* the makespan with respect to $\text{AG}_{\mathcal{M}(B, \mathbf{c})}(\mathcal{J})$. Thus, we have:

$$\text{CAP}_K(\mathcal{J} \mid \mathcal{M}(B, \mathbf{c})) \leq \text{AG}_{\mathcal{M}(B, \mathbf{c})}(\mathcal{J})$$

Furthermore, we have the following relationship between the optimal makespans (for the same job) when given different amounts of machines. Letting $\mathcal{M}(B, \mathbf{c}) \leq K$, we have that:

$$\text{OPT}_{\mathcal{M}(B, \mathbf{c})}(\mathcal{J}) \leq \frac{K}{\mathcal{M}(B, \mathbf{c})} \text{OPT}_K(\mathcal{J}).$$

To observe this, consider the limiting case as $\mathcal{M}(B, \mathbf{c}) \rightarrow 1$. When $\mathcal{M}(B, \mathbf{c}) = 1$, the optimal makespan contains no “gaps”, in the sense that the single machine is always being utilized. If the job is perfectly parallelizable and subdividable, we have that $\text{OPT}_{\mathcal{M}(B, \mathbf{c})}(\mathcal{J}) = \frac{K}{\mathcal{M}(B, \mathbf{c})} \text{OPT}_K(\mathcal{J})$ by a geometric proof (i.e., $\text{OPT}_K(\mathcal{J})$ has a makespan that is $1/K$ as long as $\text{OPT}_1(\mathcal{J})$). For any other job, as the number of machines increases, the utilization of machines worsens.

We give a visual example of such a job \mathcal{J} with $N = 10$ tasks in Fig. 21 – observe that respecting the precedence constraints in the case with $K = 3$ machines necessarily forces a makespan that is greater than the hypothetical best makespan if jobs are perfectly parallelizable and subdividable. Thus, we have that scaling $\text{OPT}_K(\mathcal{J})$ by $\frac{K}{\mathcal{M}(B, \mathbf{c})}$ is always an upper bound on the optimal makespan $\text{OPT}_{\mathcal{M}(B, \mathbf{c})}(\mathcal{J})$.

Combining the above bounds, we obtain the following:

$$\begin{aligned} \text{CAP}_K(\mathcal{J} \mid \mathcal{M}(B, \mathbf{c})) &\leq \text{AG}_{\mathcal{M}(B, \mathbf{c})}(\mathcal{J}), \\ &\leq \left(2 - \frac{1}{\mathcal{M}(B, \mathbf{c})}\right) \text{OPT}_{\mathcal{M}(B, \mathbf{c})}(\mathcal{J}), \\ &\leq \left(\frac{2K}{\mathcal{M}(B, \mathbf{c})} - \frac{K}{\mathcal{M}(B, \mathbf{c})^2}\right) \text{OPT}_K(\mathcal{J}). \end{aligned}$$

This gives that the carbon stretch factor (Definition 3.1) of CAP is given by $\left(\frac{K}{\mathcal{M}(B, \mathbf{c})}\right)^2 \frac{2\mathcal{M}(B, \mathbf{c})-1}{2K-1}$. □

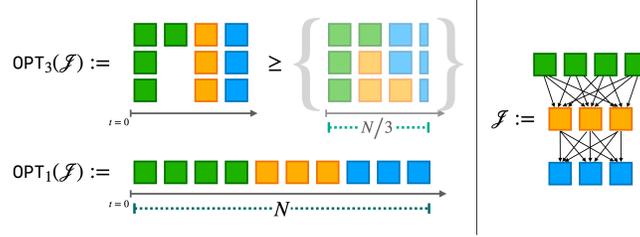


Figure 21: An example to contextualize how the optimal makespan differs when a job is given different amounts of machines. In the case of a single machine (i.e., $\text{OPT}_1(\mathcal{J})$), the precedence constraints defined by the DAG on the right-hand side of the figure are non-binding – there is always one or more tasks that are ready to execute. As the number of machines increases, situations arise where some machines must be idle (e.g., in $\text{OPT}_3(\mathcal{J})$), indicated by “blank slots” in the optimal schedule. The relation between makespans is then formalized by considering a hypothetical schedule (in brackets) that ignores precedence constraints and subdivides individual tasks across machines – while this is an infeasible schedule, it forms a *lower bound* on the makespan of any feasible one.

We now turn to the question of carbon savings, and the result stated in [Theorem 4.6](#). For ease of analysis, we again consider a *discretized time model* as defined in [Appendix B.1.2](#).

Slightly abusing notation, we let $C_{\text{CAP}}(t)$ denote the carbon emissions of CAP’s schedule during discrete time step t , and let $C_{\text{AG}}(t)$ denote the carbon emissions of AG at discrete time step t , respectively. Schedules generated by CAP and AG each use a certain number of machines during any discrete time interval – to capture this, we let $E_t^{\text{CAP}} \leq r(t') : t' \in [t, t+1)$ denote the number of active machines during discrete time step t in CAP’s schedule. Note that $r(t')$ is constant on the interval $t \in [t, t+1)$, and that E_t^{CAP} *need not be* an integer – i.e., if a machine is active for only half of the discrete time interval, that machine contributes fractionally to E_t^{CAP} . We let $E_t^{\text{AG}} \leq K$ denote the same quantity for AG’s schedule.

On a per-job basis, let T denote the makespan of AG (i.e., $T = \text{AG}_K(\mathcal{J})$), where note that $T \leq T'$. In what follows, we use W as shorthand to denote the *excess work* that CAP must complete after time T (i.e., after AG has completed). Formally, $W = \sum_{i=0}^T E_i^{\text{AG}} - E_i^{\text{CAP}}$. We also define quantities $\bar{s}^{(0,T)}$ and $\bar{c}^{(T,T')}$, which are weighted averages based on a combination of the carbon intensity and schedules of AG and CAP, respectively. These can be interpreted as the realization of carbon intensities that CAP “waited for” – in other words, it deferred some work in between time 0 and time T (saving $\bar{s}^{(0,T)}$ amount of carbon), so it must make up the difference after time T .

B.2.2 Proof of [Theorem 4.6](#)

We are ready to prove [Theorem 4.5](#), which states that CAP yields carbon savings compared to a carbon-agnostic baseline of $W \left(\bar{s}^{(0,T)} - \bar{c}^{(T,T')} \right)$.

Proof. Slightly abusing notation, we let $C_s(t)$ denote the *carbon savings* of CAP at discrete time step t . Formally, we have:

$$C_s(t) = \begin{cases} C_{\text{AG}}(t) - C_{\text{CAP}}(t) & 1 \leq t \leq T, \\ -C_{\text{CAP}}(t) & T < t \leq T'. \end{cases}$$

By definition, we have the following by substituting for the carbon emissions of ECA and CAP:

$$C_s(t) = \begin{cases} E_t^{\text{AG}} c_t - E_t^{\text{CAP}} c_t & 1 \leq t \leq T, \\ -E_t^{\text{CAP}} c_t & T < t \leq T'. \end{cases}$$

Summing over all time steps, we have that the carbon savings is given by:

$$\sum_{i=0}^{T'} C_s(i) = \sum_{i=0}^T (E_i^{\text{AG}} - E_i^{\text{CAP}}) c_i - \sum_{i=T+1}^{T'} E_i^{\text{CAP}} c_i$$

To simplify this expression, we define two terms that capture the *weighted average* carbon intensity per unit of work completed/deferred. First, note that $\sum_{i=0}^T (E_i^{\text{AG}} - E_i^{\text{CAP}}) = \sum_{i=T+1}^{T'} E_i^{\text{CAP}} = W$.

Formally, we let $\bar{s}^{(0,T)}$ denote the weighted average carbon intensity of the deferred work W that is completed by AG before time T but must be completed after time T by CAP:

$$\bar{s}^{(0,T)} = \frac{\sum_{i=0}^T (E_i^{\text{AG}} - E_i^{\text{CAP}}) c_i}{W}$$

Similarly, we let $\bar{c}^{(T,T')}$ denote the weighted average carbon intensity of the deferred work W that is completed by CAP after time T :

$$\bar{c}^{(T,T')} = \frac{\sum_{i=T+1}^{T'} E_i^{\text{CAP}} c_i}{W}$$

Under the above definitions, we can pose the total carbon savings as:

$$\sum_{i=0}^{T'} C_s(i) = W \left(\bar{s}^{(0,T)} - \bar{c}^{(T,T')} \right)$$

□

The above result contextualizes the total carbon savings achieved by CAP for a single job, but we may also consider the average carbon savings at each (discrete) carbon intensity interval as follows. We let $\rho_{\text{AG}} \in [0, 1)$ denote the average cluster utilization of AG, and we let $\rho_{\text{CAP}} \in [0, 1)$ denote the average cluster utilization of CAP. In general, since CAP allows jobs to use less than or equal the amount of resource that AG allows, we expect $\rho_{\text{AG}} \leq \rho_{\text{CAP}}$ as for the same number of jobs submitted (e.g., during a given period), jobs will take up a greater proportion of the resources that CAP allows.

Corollary B.2. *In a setting where there are always jobs with outstanding tasks in the data processing queue, the average carbon savings of CAP at any given discrete time step t is given by $(\rho_{\text{AG}}K - \rho_{\text{CAP}}r_t)\Phi_{r_t+B}$.*

Proof. In a setting where there are always jobs with outstanding tasks in the data processing queue, the expression of the average carbon savings at any given discrete time step simplifies as follows:

Let ρ_{AG} denote the average machine utilization of AG's schedule, i.e., $\rho_{\text{AG}} = \lim_{T \rightarrow \infty} \frac{\sum_{i=0}^T E_i^{\text{AG}}/K}{T}$, and let ρ_{CAP} denote the same for CAP's schedule, i.e., $\rho_{\text{CAP}} = \lim_{T \rightarrow \infty} \frac{\sum_{i=0}^T E_i^{\text{CAP}}/j_i}{T}$

Then the average carbon savings \bar{C}_s at any time step t is given by the following, where $r_t := r(t') : t' \in [t, t+1)$:

$$\begin{aligned} \bar{C}_s(t) &= (\rho_{\text{AG}}K - \rho_{\text{CAP}}r_t) c_t, \\ &\geq (\rho_{\text{AG}}K - \rho_{\text{CAP}}r_t) \Phi_{r_t+B}. \end{aligned}$$

□