A Novel Prediction Setup for Online Speed-Scaling

Antonios Antoniadis
University of Twente, Enschede, The Netherlands

Peyman Jabbarzade
University of Maryland, College Park, MD, USA

Golnoosh Shahkarami
Max Planck Institut für Informatik, Saarbrücken, Germany
Universität des Saarlandes, Saarbrücken, Germany

Abstract

Given the rapid rise in energy demand by data centers and computing systems in general, it is fundamental to incorporate energy considerations when designing (scheduling) algorithms. Machine learning can be a useful approach in practice by predicting the future load of the system based on, for example, historical data. However, the effectiveness of such an approach highly depends on the quality of the predictions and can be quite far from optimal when predictions are sub-par. On the other hand, while providing a worst-case guarantee, classical online algorithms can be pessimistic for large classes of inputs arising in practice.

This paper, in the spirit of the new area of machine learning augmented algorithms, attempts to obtain the best of both worlds for the classical, deadline based, online speed-scaling problem: Based on the introduction of a novel prediction setup, we develop algorithms that (i) obtain provably low energy-consumption in the presence of adequate predictions, and (ii) are robust against inadequate predictions, and (iii) are smooth, i.e., their performance gradually degrades as the prediction error increases.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases learning augmented algorithms, speed-scaling, energy-efficiency, scheduling theory, online algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.9

Related Version *Full Version*: https://arxiv.org/abs/2112.03082

1 Introduction

Energy is a major concern in society in general and computing environments in particular. Indeed, data centers alone are estimated to consume 200 terawatt-hours (TWh) per year, which is likely to increase by a factor of 15 by year 2030 [21]. Hardware manufacturers approach this problem by incorporating energy-saving capabilities into their hardware, with the most popular one being *dynamic speed scaling*, i.e., one can adjust the speed of the processor or device. A higher speed implies a higher energy consumption but also more processing capacity. In contrast, a lower speed incurs energy savings while being able to perform less processing per unit of time. Naturally, to take advantage of this energy-saving capability, scheduling algorithms need to decide on what speed to use at each timepoint and consider the energy consumption of the produced schedule alongside more "traditional" quality-of-service considerations.

This paper studies online, deadline-based *speed-scaling* scheduling, augmented with machine-learned predictions. More specifically, a set of jobs $\mathcal{J}$, each job $j \in \mathcal{J}$ with an associated release time r_j, deadline d_j and processing requirement w_j, arrives online and has to be scheduled on a single speed-scalable processor. A scheduling algorithm needs to decide for each timepoint t on: (i) the processor speed $s(t)$ and (ii) which job $j \in \mathcal{J}$ to execute at t

($j(t)$). Both decisions have to be made by the algorithm at any timepoint t while only having knowledge of the jobs with a release time equal to or less than t. A schedule is said to be feasible if the whole processing requirement of every job j is executed within the respective release time and deadline interval, i.e., if $\int_{t:j(t)=j} s(t)dt \geq w_j$. The energy consumption of a schedule, which we seek to minimize over all feasible schedules, is given by $\int_0^{+\infty} s(t)^\alpha dt$, where $\alpha > 1$ is a constant, which in practice is between 1.1 and 3 depending on the employed technology [15, 31]. The offline setting of the problem in which the complete job set $\mathcal{J}$ including their release times, deadlines, and workloads are known in advance was solved in the seminal paper by Yao, Demers, and Shenker [32] who gave an optimal offline algorithm called YDS. The arguably more interesting online setting in which the characteristics of a job j only become known at its release time r_j has been extensively studied [32, 10, 12, 11, 3], and the currently best known online algorithm is qOA, by Bansal et al. [11] achieving a competitive ratio of $4^\alpha/(2e^{1/2}\alpha^{1/4})$.

However, the purely online setting may be too restrictive in many practical scenarios for which one can predict – with reasonable accuracy – the characteristics of future jobs, for example, by employing a learning approach on historical data. *Learning augmented algorithms* is a very novel research area (arguably first introduced in 2018 by Lykouris and Vassilvitskii [23]) trying to capture such scenarios in which predictions of uncertain quality are available for future parts of the input. The goal in learning augmented algorithms is to design algorithms that are at the same time (i) *consistent*, i.e., obtain an improved competitive ratio in the presence of adequate predictions, (ii) *robust*, i.e., there is a worst case guarantee independently of the prediction accuracy (ideally within a constant factor of the competitive ratio of the best known online algorithm that does not employ any predictions) and (iii) *smooth*, i.e., the performance guarantee degrades gracefully with the quality of the predictions.

Previous Predictions Setups and Our Setup

Online Speed-Scaling with machine learned predictions has been investigated before by Bamas et al. [8] who consider a prediction setup in a sense orthogonal to ours; the release times and deadlines of jobs are known in advance, and there is a prediction on the processing requirement. Although any input instance (with integer release times and deadlines) can be modeled in such a way (by considering all possible pairs of release times and deadlines and a processing requirement of zero for the pairs that do not correspond to a job), this can be computationally quite expensive. Bamas et al. present a consistent, robust, and smooth algorithm for the particular case in which the interval length of each job is the same and generalize their consistency and robustness results to the general case (in which each job can have an arbitrary interval length). For this more general setting, the proof of smoothness is omitted because "... the prediction model and the measure of error quickly get complex and notation heavy".

In the current paper, we consider the novel prediction setup in which predictions on the release times and deadlines are provided to the algorithm. To keep the model simple, we assume that the actual processing requirement of each job $j \in \mathcal{J}$, as well as the number of jobs n are known. It may be useful for the reader to think about our setup as having as many unit-size jobs as total processing volume in the instance, and a prediction on the release time and deadline of each such job. We note, however, that our actual setup requires significantly fewer predictions than this simplified one.

In this context, the main contribution of the current paper is to introduce a natural alternative prediction setup and error measure as well as an algorithm (SwP) within that setup, which possesses the desired properties of consistency, smoothness, and robustness in

the general setting. It should be pointed out that since the two papers consider different prediction settings and in turn also error measures, the algorithms as well as their guarantees are incomparable. However one can consider the two prediction setups as complementary of each other.

Our Contribution

We show how the predictions can be used to develop an algorithm called Scheduling with Predictions (SwP), that improves upon qOA when the predictions are reasonably accurate. More formally, in Section 3 we show the following theorem:

▶ **Theorem 1.** *Given a parameter λ, algorithm SwP achieves a competitive ratio of $\left(\frac{1}{1-\mu}\right)^{\alpha-1}\left(\frac{2\eta+1}{1-2\lambda}\right)^{\alpha-1}$ if $\eta \in (0,\lambda)$, and $2^{\alpha-1}\alpha^{\alpha}\left(\frac{1}{\mu}\right)^{\alpha-1}$ otherwise.*

Here, η is the *error* of the prediction (defined formally later) that captures the distance between the predicted and the actual input instances, and $0 \leq \lambda < 1/2, 0 \leq \mu \leq 1$ are two hyperparameters that can be thought of as the confidence in the prediction. Theorem 1 implies that SwP is at the same time *consistent*, *smooth* and *robust* where the exact consistency, smoothness and robustness depend on the choice of the hyperparameters λ and μ.

Additionally, in Section 4 we obtain improved results for the restricted case in which all jobs have a common deadline d, and we are given predictions regarding the release times of the jobs. The corresponding algorithm is called Common-Deadline-Scheduling with Predictions (CDSwP) and obtains the following improved competitive ratio:

▶ **Theorem 2.** *Given a parameter λ, algorithm CDSwP achieves a competitive ratio of $\left(\frac{1+\eta}{1-\lambda}\right)^{\alpha-1}$ if $\eta \in (0,\lambda)$, and $2^{\alpha}\left(\frac{1+\lambda}{1-\lambda}\right)^{\alpha-1}$ otherwise.*

Although restricted, this case seems to capture the difficulty of the online setting for the problem, as supported by the fact that the strongest lower bound of $e^{\alpha-1}/\alpha$ on the competitive ratio for online algorithms for the problem is proven on such an instance [11].

Finally, in Section 5 we present an empirical evaluation of our results on a real-world data-set, which suggests the practicality of our algorithm. The actual results are preceded by Section 2 which contains preliminary results and observations. All omitted proofs can be found in the supplementary material.

1.1 Related Work

1.1.1 Online Energy-Efficient Scheduling

As already mentioned, speed-scaling was first studied from an algorithmic point of view by [32]. They studied the deadline-based version of the problem also considered here, and in addition to providing the optimal offline algorithm called *YDS*, two online algorithms called *Optimal Available (OA)* and *Average Rate (AVR)*. OA recalculates an optimal offline schedule for the remaining instance at each release time, whereas AVR "spreads" the processing volume equally between its release time and deadline in order to determine the speed for each timepoint t. The actual schedule then is simply an *Earliest Deadline First (EDF)* schedule with these speeds. They show that AVR obtains a competitive ratio of $2^{\alpha-1}\alpha^{\alpha}$ which is essentially tight as shown by [10]. Algorithm OA, on the other hand, was analyzed by [11] who proved a tight competitive ratio of α^{α}.

The currently best known algorithm for the problem, at least for modern processors which satisfy $\alpha = 3$ is the aforementioned qOA algorithm, which for any parameter $q \geq 1$ sets the speed of the processor to be q times the speed that the optimal offline algorithm would run the jobs in the current state. Algorithm qOA attains a competitive ratio of $4^\alpha/(2e^{1/2}\alpha^{1/4})$, for $q = 2 - 1/\alpha \approx 1.667$.

The multiprocessor version of online, deadline-based, speed-scaling has also been studied, see [3, 5] as well as other objectives, for example, flow time [4, 13]. We refer the interested reader to surveys [2, 20].

1.1.2 Further Results on Learning Augmented Algorithms

[23] was arguably the seminal paper in the area, considered the online caching problem. Subsequently, [27] considered the ski-rental problem as well as non-clairvoyant scheduling. Similar to the current work, the robustness and consistency guarantees were given as a function of a hyperparameter that is part of the input to the algorithm. Both the caching and the ski-rental problem have since been extensively studied in the literature (see for example [28, 6, 30] and [29, 18]).

Several other online problems have been investigated through the lens of learning-augmented algorithms and results of similar flavor were obtained. Examples include scheduling and queuing problems [26, 24], online selection and matching problems [7, 16], or the more general framework of online primal-dual algorithms [9]. We direct the interested reader to a recent survey [25].

2 Preliminaries

We consider *online, deadline-based speed-scaling* as described in the introduction. Given a scheduling algorithm A on the set jobs $\mathcal{J}$, the energy consumption of A on $\mathcal{J}$ is denoted by $\mathcal{E}_A(\mathcal{J})$. When clear from the context, we may write $\mathcal{E}_A$ instead of $\mathcal{E}_A(J)$ to simplify the notation.

As usual for online problems, the performance guarantees are given by employing *competitive analysis*. Following the speed-scaling literature (see for example [11]) we use the *strict competitive ratio*. Formally, the (strict) competitive ratio of algorithm A for the online, deadline-based speed-scaling problem, on input instance I is given by

$$\max_I \frac{\mathcal{E}_{A(I)}}{\mathcal{E}_{YDS(I)}},$$

where $\mathcal{E}_{A(I)}$ is the cost that algorithm A incurs on instance I, and the maximum is taken over all possible input instances I. The competitive ratio in many cases will depend on the prediction error.

Prediction Setup

The algorithm initially gets information about the number of jobs n, the corresponding processing volumes $w_j, \forall j \in \mathcal{J}$, as well as for every job $j \in \mathcal{J}$ a prediction p_j for the release time r_j and another prediction q_j for the deadline d_j. Again, the actual values of r_j and d_j only become known at timepoint r_j. Let $R = \{r_1, \dots r_n\}$, $D = \{d_1, \dots d_n\}$, $P = \{p_1, \dots p_n\}$ and $Q = \{q_1, \dots q_n\}$. Note that in the special case where all jobs have a common deadline d we naturally only obtain predictions for the release times.

The quality of the prediction is measured in terms of a *prediction error* η, which intuitively η measures the distance between the predicted values and the actual ones. We start by defining the individual prediction error η_i for each job $i \in \mathcal{J}$.

▶ **Definition 3.** *Let the prediction error for job i be* $\eta_i = \max\left\{\frac{|p_i - r_i|}{q_i - p_i}, \frac{|q_i - d_i|}{q_i - p_i}\right\}$.

Note that we implicitly assume that $p_i \leq q_i$ for all $i \in \mathcal{J}$ since otherwise, it is immediately obvious that the quality of the predictions is low, and one could just run a classical online algorithm for the problem. Furthermore, if the instance has a common deadline then η_i simplifies to $\eta_i = \frac{|p_i - r_i|}{d - p_i}$.

The (total) prediction error η of an input instance is then given by $\eta = \max_i \eta_i$. We call this *max-norm-error*.

▶ **Definition 4.** *We say that the total error η is a* max-norm-error *if η is given by the infinity norm of the vector of the respective errors for each job. More formally,*

$$\eta = \|\boldsymbol{\eta}\|_\infty = \max(\eta_1, \eta_2, \ldots \eta_n).$$

Performance Guarantees

In the following we formalize the performance guarantees used to evaluate our algorithms.

▶ **Definition 5.** *We say that an algorithm within the above prediction setup is:*

- Consistent, *if its competitive ratio is strictly better than that of the best online algorithm without predictions for the problem, whenever $\eta = 0$.*
- Robust, *if its competitive ratio is within a constant factor from that of the best online algorithm without predictions for the problem. Note that Robustness is independent of the prediction quality.*
- Smooth, *if its competitive ratio is a smooth function of η.*

Shrinking of Intervals

The most straightforward way to consider the predictions would arguably be to blindly trust the predictions, i.e., schedule jobs assuming that the predicted instance is the actual instance.

Consider the instance J_{PQ} (resp. J_{RD}) in which every job has the corresponding predicted (resp. actual) release time and deadline. The naive algorithm would compute the optimal offline schedule $YDS(J_{PQ})$ and try to schedule tasks according to it. If the predictions are perfectly accurate, then this clearly is an optimal schedule, and the best one can do. However, if the predictions are even slightly inaccurate, then the resulting schedule may be infeasible. Moreover, our goal is to have a robust algorithm, which cannot be obtained by following the predictions blindly. For these reasons, one has to trust the predictions more cautiously and not blindly.

One of our crucial ideas is to slightly shrink the interval between each job's release time and deadline before scheduling it. The intuition is that if the predictions are only slightly off, then a YDS schedule for the newly obtained instance will be feasible at a slight increase in energy consumption over the YDS schedule of the predicted instance. The following lemmas formalize this intuition. We note that a similar result is also presented in [8]; however, given that the actual setups are different new proofs are required (the proofs of these lemmas can be found in the supplementary material).

▶ **Lemma 6.** *Consider a common deadline instance $\mathcal{J}$, and another common deadline instance $\hat{\mathcal{J}}$ constructed from $\mathcal{J}$ such that every job $\hat{j}_i \in \hat{\mathcal{J}}$ has workload $\hat{w}_i = w_i$, $\hat{d} = d$, and $\hat{r}_i = r_i + (1 - c_i) \cdot (d - r_i)$ for some shrinking parameter $0 < c_i \leq 1$. Set $c = \max_i c_i$. Then,*

$$\mathcal{E}_{YDS}(\hat{\mathcal{J}}) \leq (1/c)^{\alpha-1} \mathcal{E}_{YDS}(\mathcal{J}).$$

▶ **Lemma 7.** *Consider a (general) instance $\mathcal{J}$, and another instance $\hat{\mathcal{J}}$ in which every job $j_i \in \mathcal{J}$ corresponds to a job $\hat{j}_i \in \hat{\mathcal{J}}$ with workload $\hat{w}_i = w_i$, $\hat{r}_i = r_i + \frac{1-c}{2} \cdot (d_i - r_i)$ and $\hat{d}_i = d_i - \frac{1-c}{2} \cdot (d_i - r_i)$ for some shrinking parameter $0 < c \leq 1$. Then,*

$$\mathcal{E}_{YDS}(\hat{\mathcal{J}}) \leq (1/c)^{\alpha-1} \mathcal{E}_{YDS}(\mathcal{J}).$$

It will be useful to bound the energy consumption of (the possibly infeasible for the original input instance) schedule $YDS(J_{PQ})$. We compute the energy consumption of schedule $YDS(J_{PQ})$ in the following lemma.

▶ **Lemma 8.** *For any $\eta \geq 0$ there holds*

$$\mathcal{E}_{YDS(J_{PQ})} \leq (2\eta + 1)^{\alpha-1} \mathcal{E}_{YDS(J_{RD})}.$$

Proof. Consider two sets $P^* = \{p_1^*, \dots p_n^*\}$ and $Q^* = \{q_1^*, \dots q_n^*\}$, with $p_i^* = p_i - \eta_i(q_i - p_i)$ and $q_i^* = q_i + \eta_i(q_i - p_i)$.

By the definition of η_i, p_i^* and q_i^*, we have $(r_i, d_i) \subseteq (p_i^*, q_i^*)$, and therefore

$$\mathcal{E}_{YDS(J_{P^*Q^*})} \leq \mathcal{E}_{YDS(J_{RD})}.$$

By having $c = \frac{1}{(2\eta+1)}$, and $J = J_{P^*Q^*}$ $(r_i = p_i^*, d_i = q_i^*)$ in Lemma 7, we obtain $J' = J_{PQ}$ and therefore,

$$\mathcal{E}_{YDS(J_{PQ})} \leq (2\eta + 1)^{\alpha-1} \mathcal{E}_{YDS(J_{P^*Q^*})}. \quad \blacktriangleleft$$

Using Lemma 6, we can obtain a similar result for common deadline instances.

▶ **Corollary 9.** *In common deadline instances for any parameter $\eta \geq 0$, there holds*

$$\mathcal{E}_{YDS(J_P)} \leq (\eta + 1)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_R)}.$$

The idea of shrinking intervals as described above will be useful for the general case as well as the restricted common deadline case.

How much each algorithm will shrink the predicted job intervals will depend on the *confidence*. This will be denoted by a *confidence parameter* $0 < \lambda \leq 1/2$ that will be given as input to the respective algorithm. In the following, we define the *shrunk prediction set* of release times and deadlines parametrized by this λ, and use the above lemmas to argue about how this "shrinking" actually affects the energy consumption of the corresponding YDS-schedule.

▶ **Definition 10.** *Let $P' = \{p'_1, \dots, p'_n\}$ and $Q' = \{q'_1, \dots, q'_n\}$ be the shrunk prediction set of release times and deadlines respectively in which $p'_i = \lfloor p_i + \lambda(q_i - p_i) \rfloor$ and $q'_i = \lceil q_i - \lambda(q_i - p_i) \rceil$ for all $i \in [n]$.*

We first observe that any schedule that considers the sets P' and Q' as the actual release times and deadlines of the jobs will be feasible, as long as the error η is not larger than λ.

▶ **Observation 11.** *Under the assumption that $\eta \in (0, \lambda)$, it follows that $r_i \leq p'_i$ and $q'_i \leq d_i$ hold for every job i.*

Therefore, the schedule $YDS(J_{P'Q'})$ is feasible. Although shrinking the intervals and then running YDS is not a robust algorithm, it will be useful to bound its energy consumption when $\eta \leq \lambda$ holds.

▶ **Lemma 12.** *For any $\eta \in (0, \lambda)$ there holds*

$$\mathcal{E}_{YDS(J_{P'Q'})} \leq \left(\frac{2\eta+1}{1-2\lambda}\right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_{RD})}.$$

Proof.

$$\mathcal{E}_{YDS(J_{P'Q'})} \leq \frac{1}{(1-2\lambda)^{(\alpha-1)}} \cdot \mathcal{E}_{YDS(J_{PQ})} \leq \left(\frac{2\eta+1}{1-2\lambda}\right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_{RD})}. \tag{1}$$

By Lemma 7 we have the first inequality in (1), and the second inequality holds because of Lemma 8. ◀

Similarly for common deadline instances, since we shrink from one side, we obtain a better competitive ratio.

▶ **Corollary 13.** *For any $\eta \in (0, \lambda)$ in common deadline instances there holds*

$$\mathcal{E}_{YDS(J_{P'})} \leq \left(\frac{1+\eta}{1-\lambda}\right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_R)}.$$

Proof.

$$\mathcal{E}_{YDS(J_{P'})} \leq \frac{1}{(1-\lambda)^{(\alpha-1)}} \cdot \mathcal{E}_{YDS(J_P)} \leq \left(\frac{1+\eta}{1-\lambda}\right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_R)}.$$

By Lemma 6 we have the first inequality, and the second inequality holds because of Corollary 9. ◀

3 General Case

In this section we present algorithm ScheduleWithPredictions(λ, μ) (SwP(λ, μ) for short) for the general learning-augmented speed-scaling setting. Parameter $0 \leq \lambda < 1/2$ describes for which range of prediction errors we would like to obtain an improved competitive ratio. The smaller the λ, the smaller that range but the better the corresponding competitive ratio for $\eta < \lambda$. On the other hand, parameter $0 \leq \mu \leq 1$ allows us to set the desired trade-off between consistency and robustness. As we will see, perfect predictions and $\lambda = \mu = 0$ would give a competitive ratio of 1.

Inspired by [8] algorithm SwP begins by partitioning each time slot $I_t = [t, t+1), t \in \mathbb{Z}$ into two parts: $I_t^\ell = [t, t + (1-\mu))$ and $I_t^r = [t + (1-\mu), t+1)$. We call I_t^ℓ the *left part*, and I_t^r the *right part* of time slot I_t. The idea is to reserve the left parts of time-slots for following the prediction, and the right parts of the time-slots are, roughly speaking, intended for safeguarding against inaccurate predictions. A key component of our algorithm consists of elegantly and dynamically distributing the processing volume of each job upon its arrival among the two parts. This distribution is crucial in order to obtain a trade-off between consistency and robustness, based on the parameters λ and μ. The algorithm consists of two steps, the *preprocessing* and the *online* step which we now describe in more detail.

Preprocessing: Partition left parts into intervals and assign jobs to them

Upon receiving the predictions (P, Q), SwP computes a YDS-schedule S' for instance (P', Q') – which is obtained by "shrinking" (P, Q) as described above. Although S' may not be feasible for the actual instance (R, D), it will be used to partition the left parts into intervals and subsequently assign each such interval of the partition to a specific job.

To this end, let $I_t(j) := [t + a_t(j), t + b_t(j)) \subseteq I_t$ be the maximal subinterval of I_t during which j is executed under S'. Note that $I_t(j)$ could be empty for some combinations of j and t. Furthermore, since by definition there are no release times or deadlines within I_t, and YDS schedules according to EDF, there can be at most one execution interval of j within I_t. Let, for every job j and left part I_t^ℓ, $I_t^\ell(j) := [t + a_t^\ell(j), t + b_t^\ell(j))$, where $a_t^\ell(j) = a_t(j)/(1-\mu)$ and $b_t^\ell(j) = b_t(j)/(1-\mu)$, be the subinterval of I_t^ℓ assigned to job j.

To obtain some intuition, scheduling the whole processing volume of each job j at a uniform speed throughout intervals $I_t^\ell(j)$ would result in a "compressed" version of $YDS(P'Q')$ where each time-slot is sped-up by a factor of $1/(1-\mu)$ to fit in the left part only, thus having an energy consumption increased by a factor of $(1/(1-\mu))^\alpha$ over that of $YDS(P'Q')$. Although (as we will see) such a compressed schedule would be consistent, it may not be robust (or even feasible) in the presence of subpar predictions. For this reason, we will eventually only schedule part of the volume of each job in the associated left parts whenever feasible, and the remaining volume will be processed on right parts.

Online Step: Job arrivals and processing

SwP needs to decide exactly when each job is to be processed within each time-slot and at what speed. This is done by (i) distributing the processing volume of each job j to right parts of different time-slots I_t and associated left parts $I_t^\ell(j)$ upon its arrival, and (ii) feasibly scheduling the whole volume assigned to the current time-slot (both to its left and right part), within the time-slot itself. In the following we discuss how this is accomplished.

(i) Job Arrivals: Upon arrival of job j at r_j, let $\delta_j = w_j/(d_j - r_j)$ be its density and $\ell(j) := \sum_{t \in [r_j, d_j)} |I_t^\ell(j)|$ be the total processing time reserved for job j on the left parts during the preprocessing step that can actually be feasibly used for job j. Furthermore let $V_t(j)$ be the total volume currently (from jobs $1, 2, \ldots j-1$) assigned to I_t^r, for all t (thus $V_t(1) = 0$).

The algorithm assigns some amount of volume y_j^t (to be determined later) of job j to interval I_t^r (thus $V_t(j+1) := V_t(j) + y_j^t$), for all $t \in [r_j, d_j)$, with $0 \le y_j^t \le \delta_j$. Finally the remaining volume $X_j := w_j - \sum_t y_j^t$ is assigned to the left parts $I_t^\ell(j)$ with $t \in [r_j, d_j)$, proportionally to their length, i.e., an interval $I_t^\ell(j)$ with $t \in [r_j, d_j)$ receives an $|I_t^\ell(j)|/\ell(j)$-fraction of X_j which implies that the average speed within $I_t^\ell(j)$ must be $X_j/\ell(j)$. To gain some intuition on the values of y_j^t, it is useful to think of the algorithm as waterfilling the volume of j to both the left and the right parts such that no right part receives more than δ_j amount of volume. More formally, the y_j^t, with $0 \le y_j^t \le \delta_j$ and $t \in [r_j, d_j)$ are defined such that they satisfy the following inequalities:

$$\frac{V_t(j)}{\mu} \ge X_j/\ell(j) \qquad \forall t \in [r_j, d_j) \text{ with } y_j^t = 0 \tag{2}$$

$$\frac{V_t(j) + y_j^t}{\mu} = X_j/\ell(j) \qquad \forall t \in [r_j, d_j) \text{ with } 0 < y_j^t < \delta_j \tag{3}$$

$$\frac{V_t(j) + y_j^t}{\mu} \le X_j/\ell(j) \qquad \forall t \in [r_j, d_j) \text{ with } y_j^t = \delta_j. \tag{4}$$

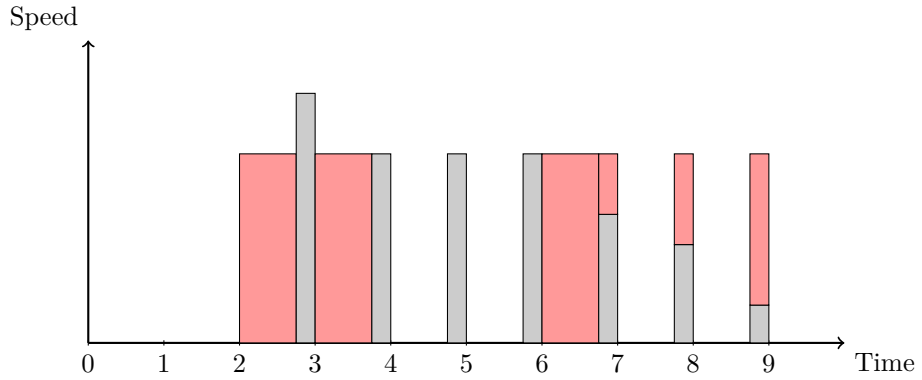

Figure 1 The speed profile corresponds to an instance with $\mu = 0.25$. Job i arrives at $r_i = 2$, with $d_i = 9$, and $w_i = 7$. Hence, $\delta_i = \frac{w_i}{d_i - r_i} = 1$. For this instance, $YDS(P'Q')$ runs job i only in 3 blocks, so we have $\ell(i) = 3 \cdot 0.75 = 2.25$. For the first four blocks we have $y_i^t = 0$ and inequality (2) holds. In the fifth and sixth blocks, $0 < y_i^t < \delta_i$ and equality (3) holds. And in the last block, $y_i^t = \delta_i = 1$ and inequality (4) holds.

Note that the left hand side in each of the above inequalities corresponds exactly to $V_t(j+1)/\mu$ and therefore to the average speed required to process the volume assigned to t before the arrival of job $j+1$ within I_t^r. We prove the existence of such y_j^t and describe how they can be computed in Appendix A.

(ii) Processing: For each $I_t, t = r_j, \dots r_{j+1} - 1$ the algorithm processes job $j' \leq j$ within every $I_t^\ell(j')$ at a speed of $X_{j'}/\ell(j')$, and the assigned volume to I_t^r is processed within I_t^r at a speed of $V_t(j+1)/\mu$, with the order of the jobs within each I_t^r being determined by EDF.

The online step gets repeated upon the arrival of each job. We next show that the resulting schedule is feasible.

▶ **Lemma 14.** *In the schedule output by $SWP(\lambda, \mu)$ a volume of w_j is fully processed for each job j within $[r_j, d_j)$.*

Proof. It is relatively easy to see that a total volume of w_j is assigned to left and right parts of I_t's with $t \in [r_j, d_j)$: Indeed, by the algorithm definition volume of w_j only gets assigned to $I_t^\ell \subset I_t$ or $I_t^r \subset I_t$ with $t \in [r_j, d_j)$. In addition, a volume of $\sum_t y_j^t$ gets assigned to the right parts and $w_j - \sum_t y_j^t$ to the left parts for a total volume assignment of w_j. It therefore remains to show that all the assigned volume is feasibly processed in the processing step.

Consider some I_t with $r_j \leq t < r_{j+1}$ and the corresponding I_t^ℓ and I_t^r. Note that by the above argument, for any such t no job with an index greater than j will assign any volume, and that only job $j' \leq j$ may be assigned to $I_t^\ell(j')$. Therefore a speed of $X_{j'}/\ell(j')$ throughout every such $I_t^\ell(j)$ is sufficient to schedule all volume assigned to it. Finally, since there are no release times or deadlines within each individual interval, the total volume of $V_t(j+1)$ can be feasibly scheduled within I_t^r at a speed of $V_t(j+1)/\mu$. ◀

We next show consistency and robustness of the algorithm.

▶ **Lemma 15** (Consistency & Smoothness). *For any $\eta \in (0, \lambda)$ there holds*

$$\mathcal{E}_{SWP} \leq \left(\frac{1}{1-\mu}\right)^{\alpha-1} \left(\frac{2\eta+1}{1-2\lambda}\right)^{\alpha-1} \mathcal{E}_{YDS(RD)}.$$

Proof. We can express $\mathcal{E}_{\text{SwP}}$ as:

$$\mathcal{E}_{\text{SwP}} = \sum_{j=1}^{n} \left(\frac{X_j^\alpha}{\ell(j)^{\alpha-1}} + \sum_{t \in [r_j, d_j)} \left(\frac{V_t(j+1)^\alpha}{\mu^{\alpha-1}} - \frac{V_t(j)^\alpha}{\mu^{\alpha-1}} \right) \right)$$
$$= \sum_{j=1}^{n} \left(\frac{X_j^\alpha}{\ell(j)^{\alpha-1}} + \sum_{t \in [r_j, d_j)} \left(\frac{\left(V_t(j) + y_j^t\right)^\alpha}{\mu^{\alpha-1}} - \frac{V_t(j)^\alpha}{\mu^{\alpha-1}} \right) \right)$$
$$\leq \sum_{j=1}^{n} \frac{w_j^\alpha}{\ell(j)^{\alpha-1}} = \left(\frac{1}{1-\mu} \right)^{\alpha-1} \mathcal{E}^j_{YDS(J_{P'Q'})}.$$

The inequality holds by convexity of the power function and by the fact that $V_t(j+1)/\mu \leq X_j/\ell(j)$ for each t such that $y_j^t > 0$ (Equations 3 and 4). The last equality follows since for $\eta \in (0, \lambda)$, for every job j there holds $[r_j, d_j) \supseteq [p'_j, q'_j)$ (Observation 11), and by construction $\ell(j)$ is $1/(1-\mu)$ times the total processing time reserved for job j under YDS($P'Q'$).

The lemma directly follows, since by Lemma 12,

$$\mathcal{E}_{YDS(J_{P'Q'})} \leq \left(\frac{2\eta + 1}{1 - 2\lambda} \right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_{RD})}. \qquad \blacktriangleleft$$

▶ **Lemma 16** (Robustness). *For any instance, we have*

$$\mathcal{E}_{SwP} \leq 2^{\alpha-1} \alpha^\alpha \left(\frac{1}{\mu} \right)^{\alpha-1} \mathcal{E}_{YDS(J_{RD})}.$$

Proof. Note that by the algorithm definition there holds that $V_t(j) \geq V_t(i)$, for $j > i$ and any t, since upon each release time new volume gets assigned but volume never gets removed. We therefore have

$$\mathcal{E}_{\text{SwP}} \leq \sum_{j=1}^{n} \left(\frac{X_j^\alpha}{\ell(j)^{\alpha-1}} \right) + \sum_{t} \left(\frac{V_t(n+1)^\alpha}{\mu^{\alpha-1}} \right)$$
$$\leq \sum_{t} \left(\frac{\left(\sum_{j : t \in [r_j, d_j)} \delta_j \right)^\alpha}{\mu^{\alpha-1}} \right) = \left(\frac{1}{\mu} \right)^{\alpha-1} \mathcal{E}_{AVR}.$$

The second inequality follows by the convexity of the power function and the fact that $V_t(n+1)/\mu \geq V_t(j+1)/\mu \geq X_j/\ell(j)$ for each t such that $y_j^t < \delta_j$ (Equations 3 and 2). The lemma follows by the competitive ratio of AVR [10]. ◀

Lemmas 15 and 16 together directly imply Theorem 1. Note that Theorem 1 not only implies consistency and robustness, but also smoothness: the competitive ratio gracefully degrades as the error increases.

4 All Jobs Have a Common Deadline

In this section, we present a simpler algorithm that achieves improved consistency and robustness over SwP for the special case in which all jobs have the same deadline, i.e., $d_j = d$ for all $j \in \mathcal{J}$. Since the deadline is the same for all jobs, we only consider predictions on the n release times $R = \{r_1, \ldots, r_n\}$ and denote these by a set $P = \{p_1, \ldots, p_n\}$.

We begin by analyzing a framework for combining different algorithms before presenting an algorithm in Subsection 4.1 that is based on combining two different algorithms; the classic online algorithm qOA that has a worst-case guarantee independent of the prediction error, and a second one, that considers the predictions and has a good performance in the case of small prediction error.

The general idea of combining online algorithms has been repeatedly employed in the past in the areas of online algorithms and online learning, see, for example, the celebrated results of Fiat et al. [17], Blum and Burch [14], Herbster and Warmuth [19], Littlestone and Warmuth [22]. Such a technique has also been used in the learning augmented setting, see Antoniadis et al. [6] for an explicit framework for combining algorithms, and Lykouris and Vassilvtiskii [23] as well as Rohatgi [28] for implicit uses of such algorithm combinations. However, as we will see, the specific problem considered in this paper allows for way more flexibility in such algorithm combinations since it is possible to simulate the parallel execution of different algorithms by increasing the speed. This allows us to obtain a much more tailored result with at most one switch between the different algorithms and more straightforward analysis. We start with the following structural lemma.

▶ **Lemma 17.** *Consider a partition of the job set of instance J into m job sets $J_1, J_2, \dots J_m$, and furthermore consider m schedules $C_1, C_2, \dots C_m$ with speed functions $s_1(t), s_2(t), \dots s_m(t)$ respectively, such that C_i is a feasible schedule for J_i for all $i = 1, \dots m$. Then there exists a schedule C with speed function $s_C(t) = \sum_i s_i(t)$ that is feasible for the complete job set J and has an energy consumption of $\mathcal{E}_C \leq m^{\alpha-1} \sum_i \mathcal{E}_i$, where for each i, $\mathcal{E}_i = \int_t s_i(t)^\alpha dt$ is the energy consumption of the respective schedule.*

4.1 Algorithm CommonDeadlineScheduleWithPredictions (CDSwP)

At a high level CDSwP(λ) (almost) follows the optimal schedule for the predicted instance as long as the prediction error is not higher than λ and switches to a classical online algorithm (i.e., one without predictions) in case the prediction error becomes higher than λ.

More formally, the algorithm can reside in one of two modes: *follow the prediction* (FtP) mode, and *recovery* mode. Initially, before the release time r_1 of the first job the algorithm is in the FtP-mode and has an associated speed-profile given by $s(FtP(0), t) = 0$ for all $t \in [0, d]$. Upon each release time r_i, $i = 1, \dots n$, and while in the FtP-mode, CDSwP(λ) does the following:

- If $\eta_i \leq \lambda$, CDSwP remains in the FtP-mode and updates the speed profile from $s(FtP(i-1), t)$ to $s(FtP(i), t)$ for $[r_i, d]$ with the help of a job instance J^i. Instance J^i consists of:
  - One job i' with release time $r_{i'} = r_i$, workload $w_{i'}$ equal to the total amount of unfinished workload at r_i workload that was released at any timepoint $t \leq r_i$, and deadline d.
  - For each job j not yet released at r_j, include job j with a release time of p'_j, a deadline of d and a volume of w_j in J^i.

  The new speed-profile $s(FtP(i), t)$ is given for any $t \in [r_i, d]$ by

$$s(FtP(i), t) := \begin{cases} s(YDS(J'), t), & \text{if } YDS(J') \text{ runs job } i' \text{ at } t, \\ 0, & \text{otherwise.} \end{cases}$$

  Algorithm CDSwP now runs at $s(FtP(i), t)$ for any $t \in [r_i, r_{i+1})$, and remains in the FtP-mode.
- Otherwise, if $\eta_i > \lambda$ then CDSwP switches to the recovery-mode, and sets $k := i$.

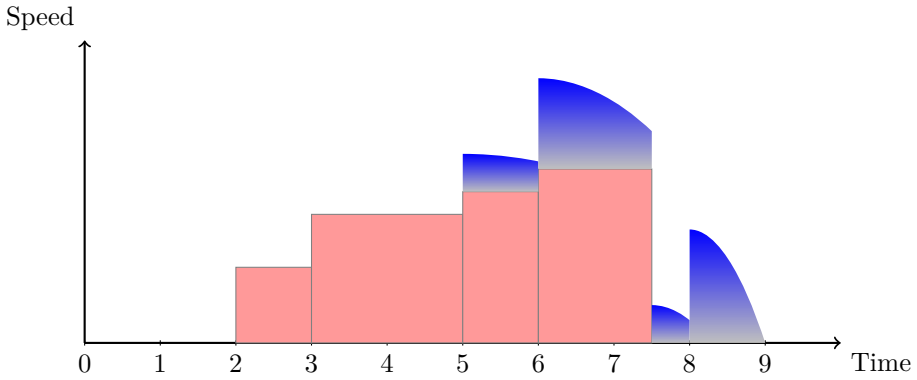

Figure 2 A common deadline instance with $\eta > \lambda$. The first time point with $\eta_i > \lambda$ is time 5 in which we start to run qOA for the rest of the jobs (blue part) while we continue running YDS for the jobs released before 5 (red part). At time point 7.5, the workload of the first set of jobs is finished.

When in recovery-mode, the algorithm runs at speed $s(t) = s(FtP(k-1), t) + s(qOA(k), t)$ at each timepoint t until d, where $s(FtP(k-1), t)$ is the last speed-profile generated in the FtP-mode, and $s(qOA(k), t)$, is the speed that the online algorithm qOA would have at timepoint t when presented (in an online fashion) with (the actual) jobs $k, \dots n$.

Note that defining the speed at any timepoint t is sufficient in order to fully describe the algorithm. Indeed, since all jobs have a common deadline of d, it is irrelevant which job (among the active jobs) is being processed at any timepoint t. Nevertheless, to simplify the presentation we will implicitly assume in the following that at timepoint t the currently active and unfinished job with the earliest release time is the one being processed – and ties are broken arbitrarily. We first prove that the algorithm produces feasible schedules:

▶ **Observation 18.** *Algorithm CDSwP fully processes the whole processing volume of each job w_j, within $[r_j, d]$.*

Proof. Note that by the algorithm definition, no job starts being processed before its arrival in any mode. So it suffices to show that the complete processing volume of each job is completed before its deadline. Assume first that the algorithm remains in the FtP-mode until d. By the definition of the job instances J^i, any still unfinished processing volume $w_{n'}$ will be assigned to job n' at timepoint r_n and YDS will schedule it within $[r_n, d)$ according to YDS at a speed of $w_{n'}/(d - r_n)$. So the resulting schedule is feasible in that case. If the algorithm switches to the recovery mode at some r_k, then by the above argument the speed profile $s(FtP(k-1), t)$ is sufficient to finish jobs $1, \dots k-1$, and furthermore speed profile $s(qOA(k), t)$ is feasible for for jobs $k, \dots n$, by the feasibility of algorithm qOA. So the overall speed profile $s(FtP(k-1), t) + s(qOA(j), t)$ is sufficient for processing the whole volume. ◀

We begin by showing the following theorem which will imply consistency and smoothness.

▶ **Lemma 19** (Consistency & Smoothness). *Under the assumption that $\eta \in (0, \lambda)$, there holds*

$$\mathcal{E}_{CDSwP} \leq \left(\frac{1+\eta}{1-\lambda}\right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_R)}.$$

Before proving Lemma 19 we show the following intermediate result.

▶ **Lemma 20.** *Assuming that $\eta \in (0, \lambda)$, there holds*

$$\mathcal{E}_{CDSwP} \leq \mathcal{E}_{YDS(J_{P'})}$$

Proof. Consider job instance $J^{i'}$ which consists of:

- A job j^{i-1} with release time r_i, deadline d and volume $w_{j^{i-1}} := w'_i - w_i$ equal to the total volume of jobs $1, \dots i-1$ that is still unfinished at r_i,
- Job i with release time at p'_i (and still deadline d and processing volume w_i),
- For each job j not yet released at r_j, include job j with a release time of p'_j, a deadline of d and a volume of w_i in $J^{i'}$.

Note that, instance $J^{i'}$ differs from J^i only in that job i is considered separately, and not together with all previously released jobs that are still not finished. By Observation 11 a YDS schedule for the former is a feasible schedule for the later, and therefore by optimality of YDS,

$$\mathcal{E}^{[r_i,\infty)}_{CDSwP(J^i)} \leq \mathcal{E}^{[r_i,\infty)}_{CDSwP(J^{i'})}, \tag{5}$$

where $\mathcal{E}^{[a,b]}_{A(J)}$, refers to the energy consumption that the schedule produced by algorithm A on instance J has within interval $[a, b]$.

Using this notation, we can express the total energy-consumption of the $CDSwP$ as

$$\begin{aligned}
\mathcal{E}_{CDSwP} &= \sum_{i=1}^{n} \mathcal{E}^{[r_i,r_{i+1})}_{CDSwP(J^i)} \\
&= \sum_{i=1}^{n-2} \mathcal{E}^{[r_i,r_{i+1})}_{CDSwP(J^i)} + \mathcal{E}^{[r_{n-1},r_n)}_{CDSwP(J^{n-1})} + \mathcal{E}^{[r_n,d)}_{CDSwP(J^n)} \\
&\leq \sum_{i=1}^{n-2} \mathcal{E}^{[r_i,r_{i+1})}_{CDSwP(J^i)} + \mathcal{E}^{[r_{n-1},r_n)}_{CDSwP(J^{n-1})} + \mathcal{E}^{[r_n,d)}_{CDSwP(J^{n\prime})} \\
&= \sum_{i=1}^{n-2} \mathcal{E}^{[r_i,r_{i+1})}_{CDSwP(J^i)} + \mathcal{E}^{[r_{n-1},d)}_{CDSwP(J^{n-1})} \\
&\vdots \\
&\leq \mathcal{E}^{[r_1,d)}_{CDSwP(J^1)} \leq \mathcal{E}_{YDS(J_{P'})},
\end{aligned}$$

where the inequalities follow by applying Equation (5). ◀

Proof of Lemma 19. By combining Lemmas 20 and 13 we have,

$$\mathcal{E}_{CDSwP} \leq \mathcal{E}_{YDS(J_{P'})} \leq \left(\frac{1+\eta}{1-\lambda}\right)^{\alpha-1} \cdot \mathcal{E}_{YDS(J_R)},$$

and the lemma directly follows. ◀

We note that the above proof also works in exactly the same way when only a subset A of the job set is processed.

▶ **Corollary 21.** *Consider a set of jobs $A \subseteq \mathcal{J}$ and assume that $\eta_i \in (0, \lambda)$ holds for every job $i \in A$. Then*

$$\mathcal{E}_{CDSwP(A)} \leq \mathcal{E}_{YDS(J_{P'}(A))}$$

We next analyze the case of inadequate predictions.

▶ **Lemma 22.** *(Robustness) With a parameter $\eta \notin (0, \lambda)$, we have*

$$\mathcal{E}_{CDSwP} \leq 2^{\alpha} \left(\frac{1+\lambda}{1-\lambda}\right)^{\alpha-1} \cdot \mathcal{E}_{qOA}.$$

Proof. As in the definition of CDSwP, let k be the smallest index, such that $\eta_k > \lambda$. Hence, the algorithm switches to the recovery mode at r_k. We partition the job set into two subsets $A = \{1, \cdots, k-1\}$ and $B = \{k, \cdots, n\}$. By Lemma 17, and by the fact that by Corollary 21 the energy consumption for set B is at most the energy consumption of qOA for the whole job instance, it suffices to upper bound the energy consumption required for set A by the total energy that $qOA(k)$ uses.

We transform the schedule obtained by CDSwP for job set A through three intermediate steps to the schedule produced by $YDS^J(R)$. Since $\mathcal{E}_{YDS^J(R)} \leq \mathcal{E}_{qOA^J(R)}$ this will imply the theorem.

Step 1. Let J^A be the job instance that contains all jobs in A, along with jobs $j = k, k+1, \ldots n$ with respective release time p'_j, deadline d and processing volume w_j.

Let $\mathcal{E}^A_{CDSwP}$, and $\mathcal{E}^A_{YDS(J^A)}$ be the energy consumptions incured while scheduling the subset of jobs A for CDSwP(J) and $YDS(J^A)$ respectively. By Corollary 21,

$$\mathcal{E}^A_{CDSwP} \leq \mathcal{E}^A_{YDS(J_{P'})}.$$

Let J^A_P be the job instance, consisting of the predicted release times (p_i) of jobs in set A and the "shrunk" predicted release times (p'_i) for the remaining jobs. Note that J^A_P differs from J^A only in the release-times of jobs in set A. Since $\eta_i \leq \lambda$ for any $i \in A$, there holds for any such i that $d - p'_i = 1/(1-\lambda)(d - p_i)$. By Lemma 6, there therefore holds

$$\mathcal{E}^A_{YDS(J_{P'})} \leq \left(\frac{1}{1-\lambda}\right)^{\alpha-1} \cdot \mathcal{E}^A_{YDS(J^A_P)}.$$

Consider set $P^* = \{p^*_1, \ldots p^*_{k-1}, p'_k, \ldots, p'_n\}$ with $p^*_i = p_i - \eta_i(q_i - p_i)$ for all $j \in [k-1]$. There holds

$$\mathcal{E}_{YDS(J^A_P)} \leq (1+\lambda)^{\alpha-1} \mathcal{E}_{YDS(J_{P^*})} \leq (1+\lambda)^{\alpha-1} \mathcal{E}_{YDS(J^A)}. \tag{6}$$

By having $c = \frac{1}{(1+\lambda)}$, and $J = J_{P^*}$ $(r_i = p^*_i)$ in Lemma 6, we obtain $J' = J_P$. Since we have $\eta_j < \lambda$ for all $j < k$, the first inequality in (6) holds. For every job $i \in A$ there holds $(r_i, d) \subseteq (p^*_i, d)$. More specifically, a feasible schedule for J^A is feasible for J_{P^*} as well. The second inequality in (6) then directly follows by the optimality of YDS.

Putting things together we therefore have

$$\mathcal{E}^A_{CDSwP} \leq \left(\frac{1+\lambda}{1-\lambda}\right)^{\alpha-1} \cdot \mathcal{E}^A_{YDS(J^A)}, \tag{7}$$

Step 2. In this step, we want to compare $\mathcal{E}^A_{YDS(J^A)}$ with the energy of YDS algorithm for a new job instance in which we consider the real release times for some jobs in set B that their shrinking predictions are after their real release times.

A job instance J^l is defined, consisting of the real release times of jobs in set A, the real release times of job j in set B for which $r_j \leq p'_j$, and the shrunk prediction (p'_j) for the rest. Since moving the release times of the future jobs to the left could increase the speed (and hence increases energy) in the first part,

$$\mathcal{E}^A_{YDS(J^A)} \leq \mathcal{E}^A_{YDS(J^l)}.$$

Step 3. In the last step, we want to compare $\mathcal{E}^A_{YDS(J^l)}$ with the optimum offline algorithm (YDS) for the complete job instance J and their real release time J_R. We want to show

$$\mathcal{E}^A_{YDS(J^l)} \leq \mathcal{E}^J_{YDS(J^l)} \leq \mathcal{E}^J_{YDS(J_R)}.$$

The first inequality holds because $A \subseteq J$. Consider the difference between two job instances J_R and J^l. Since for each job i, its available time in J_R is a subset of its available time in J^l, $YDS(J_R)$ is a feasible algorithm for job instance J^l. Therefore, the second inequality holds.

So far we proved that

$$\mathcal{E}^A_{CDSwP} \leq \left(\frac{1+\lambda}{1-\lambda}\right)^{\alpha-1} \cdot \mathcal{E}^J_{YDS(J_R)}.$$

Since we run qOA for the job set B,

$$\mathcal{E}^B_{CDSwP} = \mathcal{E}^B_{qOA(J_R)} \leq \mathcal{E}^J_{qOA(J_R)}.$$

And by Lemma 17,

$$\mathcal{E}_{CDSwP} \leq 2^{\alpha-1} \cdot \left(\left(\frac{1+\lambda}{1-\lambda}\right)^{\alpha-1} \cdot \mathcal{E}^J_{YDS(J_R)} + \mathcal{E}^J_{qOA(J_R)}\right).$$

Since $\mathcal{E}^J_{YDS(J_R)} \leq \mathcal{E}^J_{qOA(J_R)}$,

$$\mathcal{E}_{CDSwP} \leq 2^{\alpha-1} \cdot (\mathcal{E}_{qOA})\left(\left(\frac{1+\lambda}{1-\lambda}\right)^{\alpha-1} + 1\right) \leq 2^{\alpha}\left(\frac{1+\lambda}{1-\lambda}\right)^{\alpha-1} \cdot (\mathcal{E}_{qOA}). \quad \blacktriangleleft$$

Lemmas 19 and 22 together imply Theorem 2.

5 Discussion on Confidence Parameters λ and μ

In order to give some intuition on how the confidence parameters μ and λ affect the obtained performance guarantees of SwP, we perform some numerical experiments for different settings. Moreover, we compare our algorithm with the currently best-known online algorithm qOA and the optimum offline algorithm YDS using real-world data. All experiments were run on a typical laptop computer.

We only consider $\alpha = 3$ for the experiments, as this is the typical value of α for real-world processors, see for example [15, 31]. Furthermore for qOA, we only consider $q = 2 - \frac{1}{\alpha} \approx 1.667$ since this is the value that minimizes the competitive ratio [11].

The input data for our experiments is the same as in [1]. There, jobs are generated from http requests received on EPAs web-server. For practical reasons, we limit our input instances to the first 1000 jobs of their sample. In order to generate predictions for the input, we use a normal distribution with a mean of 0, and a standard deviation of $0.01, 0.05$, or 0.1. For each job, two samples from this distribution are taken and each of them is scaled by the real interval length of the job. The result is then added to each job's actual release time and deadline to obtain predictions for them.

In order to illustrate the effect of parameters λ and μ, we run SwP for different combinations of these values. In particular we consider $\lambda = 0, 0.1, 0.2, 0.3$ and $\mu = 0.1, 0.2, \cdots, 1$. Our results with standard deviation 0.05 can be found in Figure 3. Our results with standard deviations 0.01, and 0.1 can be found in the supplementary material.

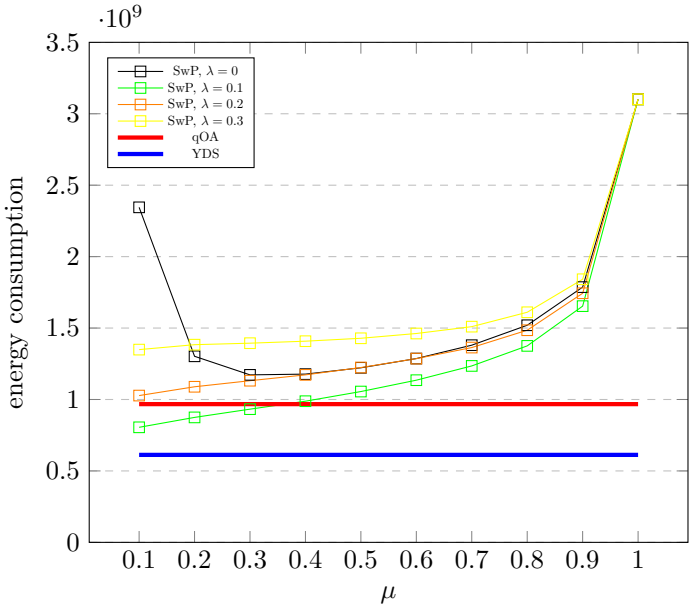

Figure 3 Prediction set 2, stddev=0.05.

To gain some intuition on the results, recall that μ denotes the portion of each block for which AVR is run. In particular, for $\mu = 1$ the SwP algorithm becomes identical to the AVR algorithm and disregards the predictions, whereas the smaller μ's value the more the predictions are trusted. This explains why the competitive ratio increases with μ. Similarly, recall that λ defines how much the predicted interval will be shrunk and that the improved competitive ratio is only proven for $\eta \leq \lambda$ but on the other hand the bigger λ gets the smaller that improvement in the competitive ratio will be. Although the best choices for λ and μ depend on the quality and/or structure of the predictions, our experiments highlight that for appropriate such choices, one can significantly improve upon the energy-consumption of qOA. To summarize, in practice the most sensible settings of λ and μ will depend on the quality as well as structure of the predictions and it may be worthwhile experimenting with different such settings.

6 Conclusion

In this paper, we have presented a consistent, smooth, and robust algorithm for the general classical, deadline-based, online speed-scaling problem using ML predictions for release times and deadlines.

We can remove the assumption of knowing the number of jobs n, by slightly adapting the error definition, so that the prediction is considered to be inadequate if the predicted number of jobs is wrong.

It remains an interesting open question on whether a similar robust, consistent and smooth algorithm exists for the more general setup in which the workloads of the jobs are not known in advance but predicted along with their release times and deadlines. Although we were able to extend SwP under the assumption that it satisfies a natural monotonicity property, it is unclear if that property holds in general.

References

1 Ahmed Abousamra, David P. Bunde, and Kirk Pruhs. An experimental comparison of speed scaling algorithms with deadline feasibility constraints. *CoRR*, abs/1307.0531, 2013. arXiv:1307.0531.

2 Susanne Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, 2010.

3 Susanne Albers, Antonios Antoniadis, and Gero Greiner. On multi-processor speed scaling with migration. *J. Comput. Syst. Sci.*, 81(7):1194–1209, 2015.

4 Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Trans. Algorithms*, 3(4):49, 2007.

5 Eric Angel, Evripidis Bampis, Fadi Kacem, and Dimitrios Letsios. Speed scaling on parallel processors with migration. *J. Comb. Optim.*, 37(4):1266–1282, 2019.

6 Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *International Conference on Machine Learning*, pages 345–355. PMLR, 2020.

7 Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In *NeurIPS*, 2020.

8 Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In *NeurIPS*, 2020.

9 Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *NeurIPS*, 2020.

10 Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. Average rate speed scaling. *Algorithmica*, 60(4):877–889, 2011.

11 Nikhil Bansal, Ho-Leung Chan, Dmitriy Katz, and Kirk Pruhs. Improved bounds for speed scaling in devices obeying the cube-root rule. *Theory Comput.*, 8(1):209–229, 2012.

12 Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), March 2007.

13 Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM J. Comput.*, 39(4):1294–1308, 2009.

14 Avrim Blum and Carl Burch. On-line learning and the metrical task system problem. *Machine Learning*, 39(1):35–58, 2000.

15 Dale L. Critchlow, Robert H. Dennard, and Stanley Schuster. Design and characteristics of n-channel insulated-gate field-effect transistors. *IBM J. Res. Dev.*, 44(1):70–83, 2000.

16 Paul Dütting, Silvio Lattanzi, Renato Paes Leme, and Sergei Vassilvitskii. Secretaries with advice. In *EC*, pages 409–429. ACM, 2021.

17 Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48(3):410–428, 1994.

18 Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *ICML*, 2019.

19 Mark Herbster and Manfred K Warmuth. Tracking the best expert. *Machine learning*, 32(2):151–178, 1998.

20 Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

21 Nicola Jones. How to stop data centers from gobbling up the world's electricity, 2018. [Online; accessed 02-August-2021].

22 N. Littlestone and M.K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994. doi:10.1006/inco.1994.1009.

23 Thodoris Lykouris and Sergei Vassilvtiskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3296–3305. PMLR, 2018.

24 Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *ITCS*, volume 151 of *LIPIcs*, pages 14:1–14:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

25 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In *Beyond the Worst-Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020.

26 Benjamin Moseley, Sergei Vassilvitskii, Silvio Lattanzi, and Thomas Lavastida. Online scheduling via learned weights. In *SODA*, 2020.

27 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

28 Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *SODA*, 2020.

29 Shufan Wang and Jian Li. Online algorithms for multi-shop ski rental with machine learned predictions. In *AAMAS*, pages 2035–2037. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

30 Alexander Wei. Better and simpler learning-augmented online caching. In *APPROX/RANDOM*, volume 176 of *LIPIcs*, pages 60:1–60:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

31 Adam Wierman, Lachlan L. H. Andrew, and Ao Tang. Power-aware speed scaling in processor sharing systems: Optimality and robustness. *Perform. Evaluation*, 69(12):601–622, 2012.

32 F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382. IEEE Computer Society, 1995.

A Calculating the y_i^t's

First we show the following lemma.

▶ **Lemma 23.** *For any given $0 \leq X \leq \ell(j) \max_{t\in[r_j,d_j)}(V_t(j)+\delta_j)/\mu$ there exist values y_j^t, with $0 \leq y_j^t \leq \delta_j$ so that equations (3),(4) and (2) are satisfied for all $t \in [r_j, d_j)$, with X in place of X_j. Furthermore for any t, t' with $y_j^t \leq y_j^{t'}$ there holds $V_{t'}(j) \leq V_t(j)$, and $\sum y_j^t$ is a continuous and non-decreasing function in X.*

Proof. If $X/\ell(j) < \min_{t\in[r_j,d_j)} V_t(j)/\mu$, then it is easy to verify that $y_j^t = \delta_j$ for all $t \in [r_j, d_j)$ satisfies all equations. So we assume for the remainder of this proof that $\min_{t\in[r_j,d_j)} V_t(j)/\mu \leq X/\ell(j) \leq \max_{t\in[r_j,d_j)}(V_t(j)+\delta_j)/\mu$.

For any $t \in [r_j, d_j)$, let

$$y_j^t := \begin{cases} 0, & \text{if } V_t(j)/\mu \geq X/\ell(j), \\ \delta_j, & \text{if } (V_t(j)+\delta_j)/\mu \leq X/\ell(j), \\ \mu X/\ell(j) - V_t(j), & \text{otherwise.} \end{cases} \tag{8}$$

It is easy to verify that for the above definition of y_j^t, equations (3), (4) and (2) are satisfied with X in place of X_j, and that for any t, t' with $y_j^t \leq y_j^{t'}$ there holds $V_{t'}(j) \leq V_t(j)$. Finally, $\sum y_j^t$ is a continuous function as a sum of a finite number of continuous functions, and non-decreasing in X (as each y_j^t is by definition a non-increasing function of X). ◀

▶ **Lemma 24.** *For any set of values $V_t(j)$, there exist values y_j^t, with $0 \leq y_j^t \leq \delta_j$ so that equations (3),(4) and (2) are satisfied for all $t \in [r_j, d_j)$.*

Proof. Note that it suffices to show that there exists $X_j = w_j - \sum_t y_j^t$ where the y_j^t are as defined in the proof of Lemma 23, since then by Lemma 23 the equations (3),(4), and (2) would hold for $X_j = w_j - \sum_t y_j^t$.

First, let $X = w_j$ and compute the values of y_j^t via (8). If $\sum_t y_j^t = 0$, then we have found the desired X and are done. Assume therefore, that $0 < \sum_t y_j^t \leq w_j$. By Lemma 23, $\sum_t y_j^t$ is

a non-decreasing and continuous function of X within $[0, w_j]$ that obtains value 0 for $X = 0$, and a value $\leq w_j$ for $X = w_j$. Equivalently the function $w_j - \sum_t y_j^t$ is non-increasing and continuous in X within $[0, w_j]$ and obtains value w_j for $X = 0$ and a value ≥ 0 for $X = w_j$. Therefore, by the intermediate value theorem there must exist an $X_j \in [0, w_j]$, such that $w_j - \sum_t y_j^t$ obtains a value of X_j, which concludes the proof of the lemma. ◀

Algorithm

Lemmas 23 and 24 directly imply an algorithm for identifying such values of y_j^t. In particular, since for any t, t' with $y_j^t \leq y_j^{t'}$ there holds $V_{t'}(j) \leq V_t(j)$, we can order all relevant t's by $V_t(j)$ and find (through enumeration) t', t'' such that for any $V_t(j) \geq V_{t'}(j)$ we have $y_j^t = 0$, for any $V_t(j) \leq V_{t''}(j)$, $y_j^t = \delta_j$ and for all other t there holds $0 < y_j^t < \delta_j$. Let N be the number of t's such that $y_j^t = \delta_j$, and $Z = w_j - N\delta_j$ be the remaining processing volume that needs to be assigned through the y_j^t's for t's with $V_{t''}(j) < V_t(j) < V_{t'}(j)$. In other words we need to find $0 < y_j^t < \delta_j$ so that $Z - \sum_t y_j^t = X_j$, and for each individual such y_j^t, we have $y_j^t = \mu X_j / \ell(j) - V_t(j)$. This implies a system of $k + 1$ equations (for some k) with $k + 1$ unknowns, that by Lemma 24 has a solution assuming that t', t'' were chosen correctly.

B Missing Plots of Section 5

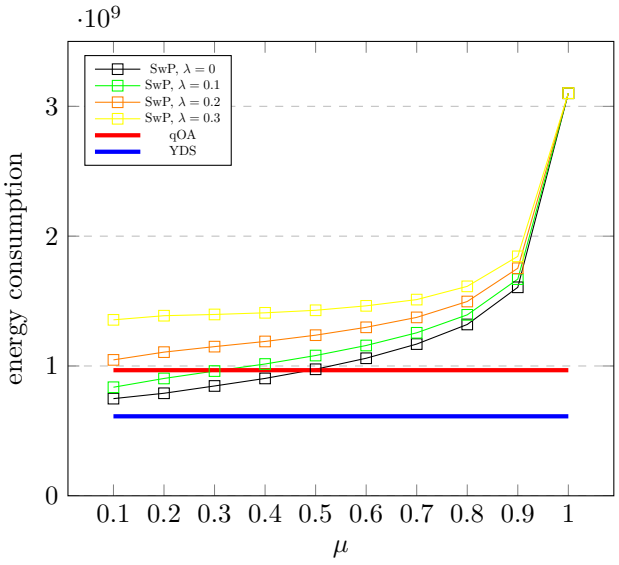

Figure 4 Prediction set 1, stddev=0.01.

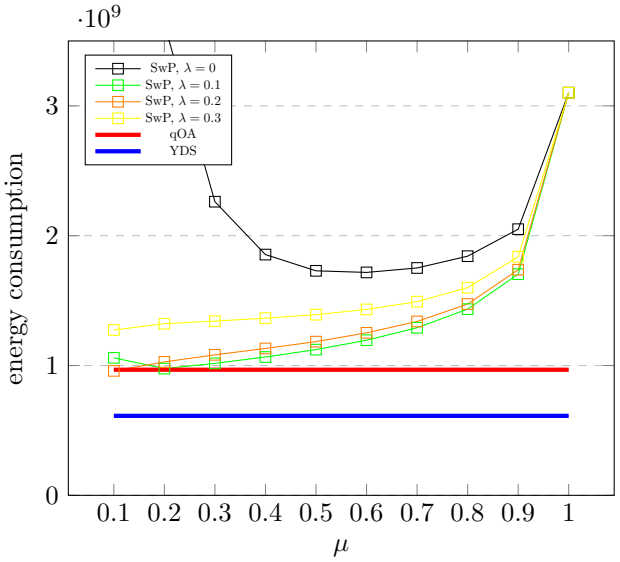

Figure 5 Prediction set 3, stddev=0.1.