Speed Scaling with Explorable Uncertainty

Evripidis Bampis, Konstantinos Dogeas, Alexander Kononov, Giorgio Lucarelli, Fanny Pascual

Speed Scaling with Explorable Uncertainty

Evripidis Bampis, Konstantinos Dogeas, Alexander Kononov, Giorgio Lucarelli, Fanny Pascual

Speed Scaling with Explorable Uncertainty

Evripidis Bampis
evripidis.bampis@lip6.fr
Sorbonne Université, CNRS, LIP6
Paris, France

Konstantinos Dogeas
konstantinos.dogeas@lip6.fr
Sorbonne Université, CNRS, LIP6
Paris, France

Alexander Kononov
alvenko@math.nsc.ru
Novosibirsk State University
Sobolev Institute of Mathematics
Novosibirsk, Russia

Giorgio Lucarelli
giorgio.lucarelli@univ-lorraine.fr
Université de Lorraine, LCOMS
Metz, France

Fanny Pascual
fanny.pascual@lip6.fr
Sorbonne Université, CNRS, LIP6
Paris, France

ABSTRACT

In this paper, we introduce a model for the speed scaling setting in the framework of explorable uncertainty. In the model, each job has a release time, a deadline and an unknown workload that can be revealed to the algorithm only after executing a query that induces a given additional job-dependent load. Alternatively, the job may be executed without any query, but in that case its workload is equal to a given upper bound. This assumption is motivated for instance in applications like code optimization, or file compression. We study the problem of minimizing the overall energy consumption for executing all the jobs in their time windows. We also consider the related problem of minimizing the maximum speed used by the algorithm. We present lower and upper bounds for both the offline case, where all the jobs are known in advance, and the online case, where the jobs arrive over time. We start with the single machine setting and we finally deal with the more general case where multiple identical parallel machines are available.

CCS CONCEPTS

• **Theory of computation** → **Scheduling algorithms**; **Online algorithms**; *Adversary models.*

KEYWORDS

speed scaling, scheduling, explorable uncertainty

ACM Reference Format:
Evripidis Bampis, Konstantinos Dogeas, Alexander Kononov, Giorgio Lucarelli, and Fanny Pascual. 2021. Speed Scaling with Explorable Uncertainty. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '21), July 6–8, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3409964.3461812

1 INTRODUCTION

Speed scaling is a standard and well-known mechanism to handle energy consumption in computing systems. Given that the characteristics of the jobs may not be known in advance, many works in speed scaling adopt the frameworks of online optimization [1], or stochastic optimization [18]. However, in some situations it is possible to obtain the exact job characteristics at some extra cost. The operation that allows to obtain the exact value of some part of the input is called a *query*. Kahan [24] was the first to formalize this notion known as *explorable uncertainty*. He applied this framework in the context of selection problems. Since then, a series of problems have been studied (e.g. see the survey [15]). In most of these works, the aim is the minimization of the number of queries needed to produce the desired solution. In this paper, we introduce a model for speed scaling problems, inspired by the model introduced recently for classical scheduling problem under explorable uncertainty in [14]. In the model of Dürr et al. [14], the uncertain information concerns the processing time of each job for which an upper bound is known in advance. It is possible to learn the exact processing time by querying at the price of a unit cost. If a job is executed without a query, then its execution time is equal to its upper bound. Contrary to the previous approaches, queries are executed directly on the machine running the jobs and so it is important to balance the time spent on queries and the time spent on the execution of jobs. More recently, an extension of this model has been considered in [4], where the querying times are job-dependent.

In this paper, we propose the study of the following natural extension of this model in the speed scaling setting: each job has a release time, a deadline and an unknown workload that can be revealed to the algorithm only after executing a query that induces a given additional job-dependent load. Alternatively, the job may be executed without any query, but in that case its workload is equal to a given upper bound. This assumption is motivated by the fact that a query could correspond to a code optimizer as mentioned in [14]. In that case, the code optimizer needs some workload to process the job and potentially reduces its workload. The upper bound on the workload of a job corresponds to the workload of the job when the code optimizer is not executed. Another possible application for this assumption is file compression. The minimization objective is the overall energy consumption for executing all the jobs in their time windows (between their release dates and deadlines). We also

consider the related problem of minimizing the maximum speed used by the algorithm.

Formulation of the problem. In the speed scaling model [27], the speed of a machine can be modified by the scheduler in order to save energy. Specifically, higher speed corresponds to better performance, but higher energy consumption. To quantify this, we assume that if the machine at a time t runs at speed $s(t)$, then the power needed is $P(s(t))$. In integrated systems produced by the standard CMOS technology, the power can theoretically be described as $P(s(t)) = s(t)^3$, but in practice this exponent varies for different architectures. In this paper, we study the more general case where the power is described by the function $P(s(t)) = s(t)^\alpha$, where $\alpha > 1$ is considered to be constant. Then, the energy consumption is computed as $E = \int P(s(t))dt$.

In the classical speed scaling setting, each job j is characterized by a triple (r_j, d_j, w_j), which represents the *release time*, the *deadline* and the *workload* of the job respectively. The workload of j should be entirely executed in the interval $(r_j, d_j]$ which is called its *active interval*. In this paper, we augment this framework by introducing an *uncertainty* on the workload of the jobs. Here, the workload, w_j, is an upper bound rather than an exact value on the actual work needed for the completion of a job. The *exact load*, $w_j^* \leqslant w_j$, can be revealed to the algorithm only after executing a *query* of additional load $c_j \in (0, w_j]$. Hence, in our setting, each job is characterized by a quintuple $(r_j, d_j, c_j, w_j, w_j^*)$, where w_j^* is not known before the end of the potential execution of the query. Note that, in the case where the query is not executed, the scheduler is obliged to execute the upper bound of the workload w_j.

We call the above enhanced model as *Query-Based Speed-Scaling* model (QBSS). The QBSS model is online by nature, since the value of w_j^* for each job j is revealed only after the potential execution of the query c_j. However, we distinguish between the *offline* and the *online* versions with respect to the classical scheduling setting. In the offline version, the entire input is known in advance, i.e., the total number of jobs to be scheduled, as well as their characteristics, except for the exact loads w_j^*. In the online version, the input becomes available to the algorithm over time: at time $t = r_j$, a new job j and its characteristics are revealed, except again for its exact load w_j^*. In other words, the algorithm does not know in advance how many jobs it has to schedule, at which time they will arrive or what are their characteristics. In both cases, if the exact load of a job j becomes known at the same time as its other characteristics, then the QBSS model reduces to the classical speed scaling setting, since the scheduler can simply decide whether to make the query for j or not based on the value of $\min\{w_j, c_j + w_j^*\}$.

Our contribution and organisation of the paper. In this paper, we study an enhanced speed scaling setting (called QBSS), where queries can be optionally executed in the system in order to reveal a more accurate value of the workload of jobs. The main objective is the energy minimization, while the maximum speed minimization is also studied.

There are two additional questions to answer for each job j in the QBSS model: whether the query will be done or not, and, if yes, how to partition the active interval of the job among the execution of its query and its exact load. Both decisions have a crucial impact on the speeds and on the consumed energy. For the first question, doing always the query leads to constant approximation algorithms, whereas never doing it leads to unbounded ratios (Section 4.1). However, in most cases a better decision can be made by comparing the values of c_j and $\frac{w_j}{\phi}$, where $\phi \approx 1,6180$ is the golden ratio. Note that the optimal algorithm has complete knowledge of the instance, including the exact loads. Hence, it can take this decision by comparing w_j and $c_j + w_j^*$. For the second question, the algorithm has to determine a *splitting point* $\tau_j = r_j + x(d_j - r_j)$, with $0 < x < 1$ so as $\tau_j \in (r_j, d_j)$, indicating the latest time at which the query has to finish execution and the earliest time at which the exact work of j may start its execution.

We introduce the notion of *equal window algorithms* according to which the active interval of a job is split in two equal sub-intervals: the query is executed in the first half, and the exact work in the second half. This is motivated by an instance consisting of a single job, where a different splitting leads to stronger lower bounds (see Lemma 4.3). A further discussion, as well as several lower bounds for the offline version of our model when a single machine is available are given in Section 4.1, where the use of randomization or oracles that answers optimally to one of the questions above are explored.

Subsequently in Section 4, we consider the offline case where all jobs have a common release date and we present a series of results based on different assumptions on the deadlines. Specifically, if all jobs have a common deadline, we propose in Section 4.2 the algorithm CRCD which achieves a 2-approximation ratio with respect to maximum speed and a $\min\{2^{\alpha-1}\phi^\alpha, 2^\alpha\}$-approximation ratio with respect to energy. A better analysis is also given for special values of α. In Section 4.3, we consider the case where all deadlines are powers of two and we propose a $(4\phi)^\alpha$-approximation algorithm (CRP2D) with respect to energy. In Section 4.4, we extend the previous result for arbitrary deadlines and we obtain an approximation ratio of $(8\phi)^\alpha$ (algorithm CRAD) by rounding down the deadlines of the instance to the closest power of two.

In Section 5 we consider the online case, and we adapt the well-known AVR and BKP online algorithms for the classical speed scaling setting to the QBSS model. The competitive ratio of our algorithms (AVRQ and BKPQ) has an additional multiplicative factor with respect to their version in the classical setting: a factor of 2^α for AVRQ in which the query is made for all jobs, and a factor of $(2 + \phi)^\alpha$ for BKPQ in which the query is decided based on the golden ratio. Note that, BKPQ is also $(2 + \phi)e$-competitive with respect to maximum speed. Finally, in Section 6 we study the QBSS model on parallel identical machines and we propose a modification of the algorithm AVR(m), which turns out to be $2^\alpha(2^{\alpha-1}\alpha^\alpha + 1)$-competitive with respect to energy.

In Section 2, we describe the related works with respect to the speed scaling setting as well as the query optimisation model. In Section 3, we present our notation and some preliminary results. We conclude in Section 7. Our results are summarized in Table 1.

2 RELATED WORK

Speed scaling. Since the seminal paper of Yao et al. [27], in 1995, which introduced the speed scaling mechanism to reduce the consumption of CPU energy, a series of papers, e.g. [2, 3, 5, 6, 9–13, 21], and surveys, e.g. [1, 8, 19], have been appeared. In [27], each job has

Table 1: Summary of our results

		Energy	
		Lower Bound	Upper Bound
Offline	Oracle	ϕ^α	-
	CRCD	$\max\{\phi^\alpha, 2^{\alpha-1}\}$	$\min\{2^{\alpha-1}\phi^\alpha, 2^\alpha\}$
	CRP2D		$(4\phi)^\alpha$
	CRAD		$(8\phi)^\alpha$
Online	AVRQ	$(2\alpha)^\alpha$	$2^\alpha 2^{\alpha-1}\alpha^\alpha$
	BKPQ	$3^{\alpha-1}$	$(2+\phi)^\alpha 2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$
	AVRQ(m)	$(2\alpha)^\alpha$	$2^\alpha(2^{\alpha-1}\alpha^\alpha+1)$

to be executed preemptively between its arrival time and deadline by a single variable-speed processor. An off-line algorithm (YDS) that is optimal with respect to energy is proposed, while two online algorithms are described for the same problem. Firstly, the Average Rate heuristic (AVR) is shown to have a constant competitive ratio, i.e., $2^{\alpha-1}\alpha^\alpha$, for any power function with $\alpha \geqslant 2$. A lower bound of α^α is stated but not proved in this paper. Bansal et al. [12] showed that this competitive ratio is essentially tight. They provide a nearly matching lower bound of $\frac{((2-\delta)\alpha)^\alpha}{2}$, where δ is a function of α that approaches zero as α approaches infinity. Secondly, the Optimal Available (OA) heuristic is introduced but not analysed in the original work [27]. Bansal et al. [13] gave a tight α^α bound on the competitive ratio of OA with respect to energy. Furthermore, they propose a new online algorithm (BKP) that is e-competitive with respect to maximum speed, and $2\left(\frac{\alpha}{\alpha-1}\right)^\alpha e^\alpha$-competitive with respect to energy, which is lower than the ratio of OA for any $\alpha \geqslant 5$. They also show that no deterministic online algorithm can have a better competitive ratio with respect to maximum speed.

Albers et al. [2] study the same problem of dynamic speed scaling but in multi-processor environments with m parallel variable-speed processors, assuming that job migration is allowed at no cost. They begin by solving optimally the offline version of the problem. Moving to the online version, they extend the two algorithms proposed by Yao et al. [27] into OA(m) and AVR(m) for the multiple machines. They show that OA(m) is α^α-competitive and that AVR(m) achieves a competitive ratio of $2^{\alpha-1}\alpha^\alpha + 1$.

Explorable uncertainty. Kahan [24] was the first to introduce the notion of explorable uncertainty studying some selection problems. Since then, many other problems have been studied in this framework. For instance, in [17, 22, 24], the problem of finding the k-th smallest value in a set of uncertainty intervals has been studied. In [26], caching problems in distributed databases have been studied. Other problems that have been studied include, the shortest path problem [16], the knapsack problem [20] and the minimum spanning tree problem [23, 25]. The goal in most of these works is the minimization of the number of queries to guarantee an exact optimum solution. In [26], the trade-off between the number of queries and the precision of the returned solution has been studied.

More close to our work are the works on scheduling under explorable uncertainty [4, 7, 14]. In [14], the authors consider the problem of scheduling jobs on a single machine when the cost of each query/test is unitary. The authors propose lower and upper bounds on the competitive ratio for deterministic and randomized algorithms. They also consider the problem of minimizing the makespan for which they propose optimal deterministic and randomized online algorithms. In [4], the authors extend the problem to non-uniform testing times and they present new competitive algorithms for different variants of the problem. In [7], a single-machine scheduling problem is considered, where given a set of n unit-time jobs, and a set of k unit-time errors, the objective is to reveal n error-free timeslots with the minimum number of queries. The authors present both lower bounds and asymptotically tight upper bounds for different variants of the problem.

3 NOTATIONS AND PRELIMINARIES

We consider a set of n jobs $\mathcal{J}$ which should be executed on a single machine or on a set of m parallel machines $\mathcal{M}$. Each job $j \in \mathcal{J}$ is characterized by a quintuple $(r_j, d_j, c_j, w_j, w_j^*)$. The scheduler should decide if the initial workload w_j will be executed, or if a query of load $c_j \in (0, w_j]$ will first run in order to reveal the exact (compressed) load $w_j^* \leqslant w_j$, which will be executed afterwards. In any case, the whole execution of the job j should be done during its active interval $(r_j, d_j]$. We assume that the *preemption* of the execution of jobs is permitted, while each machine can execute at most one job at each time. We consider two objectives: the minimization of the maximum speed used, and the minimization of the total energy consumption with respect to the speed scaling mechanism. Then, our goal is to find a feasible preemptive schedule that optimizes one of these objectives.

For a job $j \in \mathcal{J}$, we denote by p_j the amount of work an algorithm chooses to execute, i.e., $p_j = c_j + w_j^*$ if the query is executed, otherwise $p_j = w_j$. Let $p_j^* = \min\{w_j, c_j + w_j^*\}$ be the load executed by the optimal algorithm for j. The following lemma describes the relation between the load p_j^* executed by the optimal solution and the load p_j executed by an algorithm which decides the execution of the query based on the relation of the quantities c_j and $\frac{w_j}{\phi}$, where ϕ is the golden ratio, i.e., $\phi \approx 1,6180$.

Lemma 3.1. *Consider an algorithm which decides to make the query for a job $j \in \mathcal{J}$ only if $c_j \leqslant \frac{w_j}{\phi}$. Then, we have $p_j \leqslant \phi p_j^*$.*

In the classical speed scaling setting without uncertainty, the instance can be described as a set of jobs, each one characterized by the triple (r_j, d_j, w_j). Let $\delta_j = \frac{w_j}{d_j - r_j}$ be the *density* of the job j. The density is an important ingredient in most of the algorithms proposed for this setting as it is related with the speed. Note that the optimal offline solution for the QBSS model coincides with the optimal offline solution in the classical speed scaling setting by using a job (r_j, d_j, p_j^*) for each job $j \in \mathcal{J}$.

Due to space limitations, some proofs are omitted.

4 OFFLINE

4.1 Lower Bounds

In this section, we will compare the performance of an algorithm in the QBSS offline model, i.e an algorithm which does not know the values w_j^*, to an optimal algorithm, which knows these values. Our aim is to give lower bounds on the approximation ratio of any algorithm in our setting, for the two objectives that we consider, the minimization of the maximum speed, and the minimization of the total energy. All our results hold for both the single machine

case and the multiple machines case, since they will only need to consider a single task. Before introducing our results, let us define a new setting, specifically for instances with one job, which we call the *oracle model*.

Recall that x, $0 < x < 1$, is the fraction of the window $(r_j, d_j]$ in which the query is executed. In other words, in the case where we decide to make the query, then this will be executed in $(r_j, r_j + x(d_j - r_j)]$, while the exact work w_j^* will be executed in $(r_j + x(d_j - r_j), d_j]$. In the *oracle model*, we suppose the existence of an oracle that can give us the best value of x for the single job of the instance. Therefore, in this model the algorithm needs to take only one decision, i.e. to make or not the query for the job (if the decision is to make the query, the oracle will dictate where to split the window).

Note that the existence of such an oracle is highly improbable, because it translates to knowing the exact load w_j^* of the job upon its arrival, which conflicts with the setup of our model. The oracle model is however interesting to give lower bounds on the approximation ratio of an algorithm in our setting for two reasons. Firstly, a lower bound on the approximation ratio with the oracle model helps us to better understand the difficulty of our problem. It allows us to see whether the difficulty of a problem is due to the fact that we don't know if it is worthy to do the query or not, or due to the fact that, once we have chosen to do a query, we don't know the exact load w_j^* before the query has been completed. Of course, a lower bound in the oracle model is also valid in the general model. Secondly, in the following lemmas we mainly create instances of a single task. In the oracle model, once it has been decided that the query will be done, the speed to execute this task will be constant during its whole interval, since this choice minimizes both the maximal speed and the energy due to the convexity of the power function.

Lemma 4.1. *Any algorithm which never makes the query, can be arbitrarily bad with respect to maximum speed and to energy.*

Proof. We consider an instance consisting of a single job j for which $r_j = 0$, $d_j = 1$, $c_j = \varepsilon w_j$, and $w_j^* = \varepsilon w_j$, with $\varepsilon < 1$ a small positive constant. If the algorithm does not execute the query, then it uses speed $s = \frac{w_j}{d_j - r_j}$, whereas the speed used by an optimal algorithm is $s^* = \frac{p_j^*}{d_j - r_j} = \frac{c_j + w_j^*}{d_j - r_j}$. Concerning the maximum speed, the ratio of such an algorithm is $\frac{s}{s^*} = \frac{w_j}{c_j + w_j^*} = \frac{1}{2\varepsilon}$, which can be arbitrarily large. Since the speed is constant during the whole interval of size 1, we get that the energy used by the algorithm is $E = s^\alpha$, while the optimal energy is $E^* = (s^*)^\alpha$. The approximation ratio of the algorithm, concerning energy, is thus at least $\frac{E}{E^*} = (\frac{s}{s^*})^\alpha = (\frac{1}{2\varepsilon})^\alpha$, which can be arbitrarily large. □

Lemma 4.2. *For any $\epsilon > 0$, there is no deterministic $(\phi - \epsilon)$-approximate algorithm with respect to maximum speed, even in the* oracle model. *Likewise, there is no $(\phi^\alpha - \varepsilon)$-approximate algorithm with respect to the energy, even in the* oracle model. *Here $\phi \approx 1,6180$ is the golden ratio.*

Lemma 4.3. *For any $\epsilon > 0$, there is no deterministic $(2 - \epsilon)$-approximation algorithm with respect to maximum speed. Moreover, there is no deterministic $(2^{\alpha-1} - \epsilon)$-approximation algorithm with respect to energy.*

Proof. We consider an instance consisting of a single job active in the interval $(r_j, d_j]$, for which $c_j = 1$ and $w_j = 2$. Let $\mathcal{A}$ be a deterministic algorithm. In the case where $\mathcal{A}$ does not make the query, then its speed will be constant during the whole interval and we have that $s = \frac{w_j}{d_j - r_j} = \frac{2}{d_j - r_j}$. In this case the adversary will set $w_j^* = 0$. Therefore, for the speed of an optimal algorithm we have $s^* = \frac{c_j}{d_j - r_j} = \frac{1}{d_j - r_j}$. The approximation ratio of $\mathcal{A}$ with respect to maximum speed is at least 2, while with respect to energy is at least 2^α.

Let us now consider the case where $\mathcal{A}$ makes the query. Recall that the query is executed in $(r_j, r_j + x(d_j - r_j)]$ and the exact work in $(r_j + x(d_j - r_j), d_j]$. Thus, the speed of $\mathcal{A}$ during the whole first interval is $s_1 = \frac{c_j}{x(d_j - r_j)}$, while during the whole second interval is $s_2 = \frac{w_j^*}{(1-x)(d_j - r_j)}$. We have two sub-cases with respect to x. If $x \in \left(0, \frac{1}{2}\right]$, then the adversary will set $w_j^* = 0$, and hence the speed of an optimal algorithm $s^* = \frac{c_j}{d_j - r_j}$ will be constant for the whole interval, while $s_1 \geqslant \frac{2c_j}{(d_j - r_j)}$. In this case, the approximation ratio of $\mathcal{A}$ with respect to maximum speed is at least $\frac{s_1}{s^*} \geqslant 2$, while with respect to energy is at least $\frac{E}{E^*} = \frac{x(d_j - r_j)s_1^\alpha}{(d_j - r_j)(s^*)^\alpha} = \frac{x\left(\frac{c_j}{x(d_j - r_j)}\right)^\alpha}{(\frac{c_j}{d_j - r_j})^\alpha} = x^{1-\alpha} \geqslant 2^{\alpha-1}$. If $x \in \left[\frac{1}{2}, 1\right)$, then the adversary will set $w_j^* = w_j$ having $s^* = \frac{w_j}{d_j - r_j}$. Then the maximum speed used by $\mathcal{A}$ is $s \geqslant \max\{s_1, s_2\} \geqslant s_2 \geqslant \frac{w_j}{(1-x)(d_j - r_j)} \geqslant \frac{w_j}{\frac{d_j - r_j}{2}} = \frac{2w_j}{d_j - r_j}$. In this case, the approximation ratio of $\mathcal{A}$ with respect to maximum speed is at least $\frac{s_2}{s^*} = \frac{1}{1-x} \geqslant 2$, while with respect to energy is at least $\frac{E}{E^*} = \frac{(1-x)(d_j - r_j)s_2^\alpha}{(d_j - r_j)(s^*)^\alpha} = \frac{(1-x)\left(\frac{w_j}{(1-x)(d_j - r_j)}\right)^\alpha}{\left(\frac{w_j}{d_j - r_j}\right)^\alpha} = (1-x)^{1-\alpha} \geqslant 2^{\alpha-1}$. □

The next lemma deals with randomized algorithms. We consider that for a given instance I, a randomized algorithm makes the query with a probability ρ_I, and does not make it with probability $1 - \rho_I$. The approximation ratio of a randomized algorithm is the maximum value, over all instances, of the expected value of the objective function (energy or maximum speed) of the algorithm over the value of an optimal solution. We focus in this paper on deterministic algorithms, but it is worth noticing that the problem is also difficult, even with a randomized algorithm, and even in the oracle model. As in the previous proofs, we will use a proof with a single task: the algorithm will only have to choose with which probability it will do the query (if the query is done then the window is divided into two parts optimally, so that the speed is constant during the whole interval).

Lemma 4.4. *For any $\epsilon > 0$, there is no $(4/3 - \epsilon)$-approximate randomized algorithm with respect to maximum speed, even in the* oracle model. *Likewise, there is no $\left(\frac{1}{2}(1 + \phi^\alpha) - \varepsilon\right)$-approximate randomized algorithm with respect to energy, even in the* oracle model.

Algorithm 1: CRCD

for *each job* $j \in \mathcal{J}$ **do**
  if $j \in B$, *i.e.,* $c_j \leqslant \frac{w_j}{\phi}$ **then**
    Add $(0, \frac{D}{2}, c_j)$ in set $\mathcal{Q}$;
  if $j \in A$, *i.e.,* $c_j > \frac{w_j}{\phi}$ **then**
    Add $(0, \frac{D}{2}, \frac{w_j}{2})$ in set $\mathcal{W}_1$;
Schedule the jobs in $\mathcal{Q} \cup \mathcal{W}_1$ in an arbitrary order during the interval $\left(0, \frac{D}{2}\right]$ using speed $s(t) = \sum_{j \in \mathcal{Q} \cup \mathcal{W}_1} \delta_j$;
// At time $\frac{D}{2}$ all queries are done;
for *each job* $j \in \mathcal{J}$ **do**
  if $j \in B$ **then**
    Add $(\frac{D}{2}, D, w_j^*)$ in set $\mathcal{W}^*$;
  if $j \in A$ **then**
    Add $(\frac{D}{2}, D, \frac{w_j}{2})$ in set $\mathcal{W}_2$;
Schedule the jobs in $\mathcal{W}^* \cup \mathcal{W}_2$ in an arbitrary order during the interval $\left(\frac{D}{2}, D\right]$ using speed $s(t) = \sum_{j \in \mathcal{W}^* \cup \mathcal{W}_2} \delta_j$;

Lemma 4.5. *The competitive ratio of an equal window algorithm is at least* 3 *with respect to the maximum speed, and at least* $3^{\alpha-1}$ *with respect to energy.*

Note that this last result holds even if we restrict to instances where the optimal algorithm always does the query (since in the example of the proof above both the equal window algorithm and the optimal algorithm always do the query). This shows that, even if an oracle would tell us whether the query should be done or not, the difficulty of splitting the window for each job (query, real work) is significant.

4.2 Common Release, Common Deadline

In this section, we consider that all jobs are released at time 0, and they have to finish execution at time D. We present Algorithm 1 which is an approximation algorithm with respect to both maximum speed and energy. For each job of our instance, the algorithm creates two jobs of the classical speed scaling setting. In order to do this, it first partitions the jobs into two subsets A and B, where A and B are defined as follows: $A = \{j \in \mathcal{J} : c_j > \frac{w_j}{\phi}\}$ and $B = \{j \in \mathcal{J} : c_j \leqslant \frac{w_j}{\phi}\}$. By construction, we have that $A \cup B = \mathcal{J}$ and $A \cap B = \emptyset$.

For the jobs in A the algorithm chooses to execute their initial workload without doing a query. Specifically, for each job $j \in A$, it creates two jobs j_1 and j_2 with half the initial workload to be scheduled in the first half and the second half of the initial interval respectively: $(r_{j_1}, d_{j_1}, w_{j_1}) = (0, \frac{D}{2}, \frac{w_j}{2})$ and $(r_{j_2}, d_{j_2}, w_{j_2}) = (\frac{D}{2}, D, \frac{w_j}{2})$. On the other hand, for the jobs in B the algorithm chooses to make the query and hence the exact load of these jobs is revealed once the execution of their query is finished. Specifically, for each job $j \in B$, it creates at time 0 the job $(0, \frac{D}{2}, c_j)$ to be scheduled in the first half of the initial interval. At the end of this first half-interval, the exact load w_j^* of j is known, and hence the algorithm creates the job $(\frac{D}{2}, D, w_j^*)$ to be scheduled in the second half-interval.

Theorem 4.6. *Algorithm 1 achieves an approximation ratio of 2 with respect to maximum speed and of* $\min\{2^{\alpha-1}\phi^\alpha, 2^\alpha\}$ *with respect to energy.*

Proof. The optimal solution for this problem is computed by using the offline optimal YDS algorithm [27]. Since all jobs are active during the same interval $(0, D]$, the speed during the whole interval is constant and equal to the sum of densities of all jobs. In an optimal solution, the load for each job $j \in \mathcal{J}$ is $p_j^* = \min\{w_j, c_j + w_j^*\}$, and hence its density is $\delta_j^* = \frac{\min\{w_j, c_j + w_j^*\}}{D} = \frac{p_j^*}{D}$. Then, the speed at each time t is $s^* = s^*(t) = \sum_{j \in \mathcal{J}} \frac{p_j^*}{D}$ and the total energy consumed by the optimal solution is

$$E^* = \int_0^D \left(s^*(t)\right)^\alpha dt = D\left(\sum_j \frac{p_j^*}{D}\right)^\alpha$$

Algorithm 1 produces a schedule which uses two distinct speeds s_1 and s_2 in the time intervals $(0, \frac{D}{2}]$ and $(\frac{D}{2}, D]$ respectively. For these speeds we have

$$\begin{aligned} s_1 &= \sum_{j \in \mathcal{Q} \cup \mathcal{W}_1} \delta_j = \sum_{j \in \mathcal{W}_1} \frac{\frac{w_j}{2}}{\frac{D}{2} - 0} + \sum_{j \in \mathcal{Q}} \frac{c_j}{\frac{D}{2} - 0} \\ &= \sum_{j \in A} \frac{w_j}{D} + \sum_{j \in B} \frac{2c_j}{D} \leqslant \sum_{j \in A} \frac{\phi p_j^*}{D} + \sum_{j \in B} \frac{2p_j^*}{D} \leqslant 2 \sum_{j \in \mathcal{J}} \frac{p_j^*}{D} = 2s^* \end{aligned}$$

and

$$\begin{aligned} s_2 &= \sum_{j \in \mathcal{W}^* \cup \mathcal{W}_2} \delta_j = \sum_{j \in \mathcal{W}_2} \frac{\frac{w_j}{2}}{D - \frac{D}{2}} + \sum_{j \in \mathcal{W}^*} \frac{w_j^*}{D - \frac{D}{2}} \\ &= \sum_{j \in A} \frac{w_j}{D} + \sum_{j \in B} \frac{2w_j^*}{D} \leqslant \sum_{j \in A} \frac{\phi p_j^*}{D} + \sum_{j \in B} \frac{2p_j^*}{D} \leqslant 2 \sum_{j \in \mathcal{J}} \frac{p_j^*}{D} = 2s^* \end{aligned}$$

where the first inequality in both cases holds by Lemma 3.1, $c_j \leqslant \min\{c_j + w_j^*, w_j\} = p_j^*$ and $w_j^* \leqslant \min\{c_j + w_j^*, w_j\} = p_j^*$. Hence, Algorithm 1 is 2-approximation with respect to maximum speed.

For the energy consumption of our algorithm we have:

$$\begin{aligned} E_I &= \int_0^{\frac{D}{2}} s_1^\alpha dt + \int_{\frac{D}{2}}^D s_2^\alpha dt \\ &= \frac{D}{2}\left(\sum_{j \in A} \frac{w_j}{D} + \sum_{j \in B} \frac{2c_j}{D}\right)^\alpha + \frac{D}{2}\left(\sum_{j \in A} \frac{w_j}{D} + \sum_{j \in B} \frac{2w_j^*}{D}\right)^\alpha \end{aligned}$$

We can now bound the total energy consumption of Algorithm 1. We use two different approaches. In the first approach, we apply the property $x^\alpha + y^\alpha \leqslant (x+y)^\alpha$. Specifically, we have

$$\begin{aligned} E &\leqslant \frac{D}{2}\left(\sum_{j \in A} \frac{w_j}{D} + \sum_{j \in B} \frac{2c_j}{D} + \sum_{j \in A} \frac{w_j}{D} + \sum_{j \in B} \frac{2w_j^*}{D}\right)^\alpha \\ &= \frac{D}{2}\left(\sum_{j \in A} \frac{2w_j}{D} + \sum_{j \in B} \frac{2c_j + 2w_j^*}{D}\right)^\alpha \\ &= 2^{\alpha-1} D\left(\sum_{j \in A} \frac{p_j}{D} + \sum_{j \in B} \frac{p_j}{D}\right)^\alpha \\ &\leqslant 2^{\alpha-1} D\left(\sum_{j \in A} \frac{\phi p_j^*}{D} + \sum_{j \in B} \frac{\phi p_j^*}{D}\right)^\alpha \\ &= 2^{\alpha-1}\phi^\alpha D\left(\sum_j \frac{p_j^*}{D}\right)^\alpha = 2^{\alpha-1}\phi^\alpha E^* \end{aligned}$$

where the second inequality holds by Lemma 3.1.

In the second approach, we bound the energy of the entire interval by twice the maximum energy consumed in one of the two half-intervals. Hence, we have

$$E \leqslant 2 \cdot \max\left\{\frac{D}{2}\left(\sum_{j\in A}\frac{w_j}{D}+\sum_{j\in B}\frac{2c_j}{D}\right)^{\alpha}, \frac{D}{2}\left(\sum_{j\in A}\frac{w_j}{D}+\sum_{j\in B}\frac{2w_j^*}{D}\right)^{\alpha}\right\}$$

$$= D\max\left\{\left(\sum_{j\in A}\frac{p_j}{D}+\sum_{j\in B}\frac{2c_j}{D}\right)^{\alpha}, \left(\sum_{j\in A}\frac{p_j}{D}+\sum_{j\in B}\frac{2w_j^*}{D}\right)^{\alpha}\right\}$$

$$\leqslant D\max\left\{\left(\sum_{j\in A}\frac{\phi p_j^*}{D}+\sum_{j\in B}\frac{2p_j^*}{D}\right)^{\alpha}, \left(\sum_{j\in A}\frac{\phi p_j^*}{D}+\sum_{j\in B}\frac{2p_j^*}{D}\right)^{\alpha}\right\}$$

$$= D\left(\sum_{j\in A}\frac{\phi p_j^*}{D}+\sum_{j\in B}\frac{2p_j^*}{D}\right)^{\alpha} \leqslant 2^{\alpha}D\left(\sum_{j}\frac{p_j^*}{D}\right)^{\alpha} = 2^{\alpha}E^*$$

where the second inequality holds using Lemma 3.1 and the facts that $c_j \leqslant \min\{w_j, c_j + w_j^*\} = p_j^*$ and $w_j^* \leqslant \min\{w_j, c_j + w_j^*\} = p_j^*$. The third inequality holds since $\phi < 2$. □

In what follows in this section, we give a more tight analysis of Algorithm 1 for special values of α based on the following lemma.

Lemma 4.7. *Let $\alpha \geqslant 2$ and $x \geqslant y$. Then $(x+y)^{\alpha} \geqslant x^{\alpha} + y^{\alpha} + \alpha x^{\alpha-1}y$.*

Theorem 4.8. *If $\alpha \geqslant 2$, then Algorithm 1 achieves a competitive ratio of $\max_{r\geqslant 1}\{\min\{f_1(r), f_2(r)\}\}$ with respect to energy, where $f_1(r) = 2^{\alpha-1}\left(1+\frac{1}{r^{\alpha}}\right)$, $f_2(r) = 2^{\alpha-1}\phi^{\alpha}\left[1-\frac{\alpha r^{\alpha-1}}{(r+1)^{\alpha}}\right]$ and $r = \frac{x}{y}$, where $x = \sum_{j\in A}\frac{w_j}{D}+\sum_{j\in B}\frac{2c_j}{D}$ and $y = \sum_{j\in A}\frac{w_j}{D}+\sum_{j\in B}\frac{2w_j^*}{D}$.*

In general, comparing the three ratios $\rho_1 = 2^{\alpha-1}\phi^{\alpha}$, $\rho_2 = 2^{\alpha}$ and $\rho_3 = \max_{r\geqslant 1}\left\{\min\left\{2^{\alpha-1}\left(1+\frac{1}{r^{\alpha}}\right), 2^{\alpha-1}\phi^{\alpha}\left[1-\frac{\alpha r^{\alpha-1}}{(r+1)^{\alpha}}\right]\right\}\right\}$ for different values of α, we get that ρ_1 is better for $1 < \alpha \leqslant 1.44$, ρ_2 is better for $1.44 < \alpha < 2$ and ρ_3 is better for $\alpha \geqslant 2$.

α	1.25	1.5	1.75	2	2.25	2.5	2.75	3
ρ_1	**2.17**	2.91	3.90	5.23	7.02	9.41	12.63	16.94
ρ_2	2.37	**2.82**	**3.36**	4	4.75	5.65	**6.72**	**8**
ρ_3	0	0	0	**2.76**	**3.70**	**5.25**	**6.72**	**8**

4.3 Common Release, Power of 2 Deadlines

In this section, we consider that all jobs are released at time zero, but they have a different deadline. We assume that the deadlines are powers of 2 and that 2^k is the biggest deadline of our instance.

We present here Algorithm 2 which is an approximation algorithm with respect to energy. We split again the set of jobs $\mathcal{J}$ into two subsets: $A = \{j \in \mathcal{J} : c_j > \frac{w_j}{\phi}\}$ and $B = \{j \in \mathcal{J} : c_j \leqslant \frac{w_j}{\phi}\}$. We further split B into the subsets $B_\ell = \{j \in B : d_j = 2^\ell\}, 0 \leqslant \ell \leqslant k$, with respect to the deadline of the jobs. As in the previous section, for each job in our instance, Algorithm 2 creates one or two jobs of an instance of the classical speed scaling setting.

In order to analyze our algorithm, we define the three following instances of the classical speed scaling setting:

Algorithm 2: CRP2D

```
for each job j ∈ J do
    if j ∈ B, i.e., c_j ⩽ w_j/φ then
        Add (0, d_j/2, c_j) in set Q;
    if j ∈ A, i.e., c_j > w_j/φ then
        Add (0, d_j, w_j) in set W;
Run YDS algorithm to determine the speed YDS^S(t) for
  each time t ∈ (0, 2^k] for the jobs in Q ∪ W;
In the interval (0, 1/2], execute the (parts of) jobs in Q ∪ W
  scheduled by YDS during this interval, using speed
  s(t) = s^YDS(t);
for each discrete time 2^ℓ/2, ℓ = 0, 1, ..., k do
    // the queries for the jobs in B_ℓ are finished;
    for j ∈ B_ℓ do
        Add (d_j/2, d_j, w_j^*) in set W_ℓ^*
    In the interval (2^ℓ/2, 2^ℓ], execute the (parts of) jobs in
      Q ∪ W scheduled by YDS during this interval as well
      as the jobs in W_ℓ^*, using speed
      s(t) = s^YDS(t) + Σ_{j∈W_ℓ^*} δ_j;
```

- I^*: $(0, d_j, p_j^*)\forall j \in \mathcal{J} = A \cup B$
- I': $(0, d_j, c_j)$ and $(0, d_j, w_j^*)\forall j \in B$ and $(0, d_j, w_j)\forall j \in A$
- $I'_{1/2}$: $(0, \frac{d_j}{2}, c_j)$ and $(\frac{d_j}{2}, d_j, w_j^*)\forall j \in B$ and $(0, d_j, w_j)\forall j \in A$

Lemma 4.9. *Let E^* and E' be the energy consumption in an optimal schedule for the instance I^* and I' respectively. Then, $E' \leqslant \phi^{\alpha}E^*$.*

Proof. Given an optimal solution for the instance I^*, we are going to create a feasible schedule, $\mathcal{S}$, for the instance I'.

Consider an arbitrary job $j \in \mathcal{J}$ and its corresponding job $(0, d_j, p_j^*)$ of the instance I^* which is executed in q intervals in the optimal schedule for this instance: $(t_1, t'_1]$, $(t_2, t'_2]$,..., $(t_q, t'_q]$. Let s_p, $1 \leqslant p \leqslant q$, be the speed used in the interval $(t_p, t'_p]$. By definition, we have that

$$p_j^* = \sum_{p=1}^{q}\int_{t_p}^{t'_p} s_p dt = \sum_{p=1}^{q}(t'_p - t_p)s_p$$

In the schedule $\mathcal{S}$, we use in the interval $(t_p, t'_p]$, $1 \leqslant p \leqslant q$, the speed ϕs_p. Hence the work that can be executed in this interval is

$$\sum_{p=1}^{q}(t'_p - t_p)\phi s_p = \phi\sum_{p=1}^{q}(t'_p - t_p)s_p = \phi p_j^* \geqslant p_j$$

where the inequality follows from Lemma 3.1. Thus, in these intervals we can execute the jobs $(0, d_j, c_j)$ and $(0, d_j, w_j^*)$ or the job $(0, d_j, w_j)$ of the instance I'. By doing this for each job, we get a feasible schedule for the instance I' which at each time t uses speed ϕ times bigger than the speed of the optimal schedule for the instance I^*, and hence the energy consumption $E(\mathcal{S})$ in the schedule $\mathcal{S}$ is at most ϕE^*. Therefore, an optimal schedule for I' will use even smaller energy, i.e., $E' \leqslant E(\mathcal{S}) \leqslant \phi^{\alpha}E^*$, and the lemma follows. □

Lemma 4.10. *Let E' and $E'_{1/2}$ be the energy consumption in an optimal schedule for the instance I' and $I'_{1/2}$ respectively. Then $E'_{1/2} \leqslant 2^{\alpha} E'$.*

Proof. We consider that both optimal solutions for the instances I' and $I'_{1/2}$ are created using the YDS algorithm. Let $s^*_{I'}(t)$ be the speed at each time t in an optimal schedule for the instance I'. Due to the YDS algorithm and the fact that the jobs have a common release date, this speed is non-increasing with respect to the time, i.e., $s^*_{I'}(t_1) \geqslant s^*_{I'}(t_2)$ for each $t_1 < t_2$. Moreover, the speed can change only at a deadline.

Note that the optimal schedule for I' is not feasible for $I'_{1/2}$. In order to make it feasible, we first transform the optimal schedule for I' into an intermediate schedule $\mathcal{S}$ which at each time t uses speed $s(t) = 2s^*_{I'}(t)$. Specifically, for any ℓ, $0 \leqslant \ell \leqslant k$, consider the work executed during the time interval $(2^{\ell-1}, 2^{\ell}]$. By doubling the speed during this interval, we can execute all this work during the first half, i.e., $(2^{\ell-1}, 2^{\ell} - 2^{\ell-2}]$, while the second half, i.e., $(2^{\ell} - 2^{\ell-2}, 2^{\ell}]$, remains idle. In a similar way we double the speed during $(0, \frac{1}{2}]$, and we are able to execute all of its work during $(0, \frac{1}{4}]$ while $(\frac{1}{4}, \frac{1}{2}]$ remains idle. By slightly abusing the definitions, we assume that the speed of the machine for any time t satisfies $s(t) = 2s^*_{I'}(t)$, even during the idle intervals of $\mathcal{S}$ where no work is executed. Note that, for each time interval $(0, 2^{\ell}]$, $-1 \leqslant \ell \leqslant k$, the half of it is idle in the constructed schedule $\mathcal{S}$. However, $\mathcal{S}$ is still not feasible for the instance $I'_{1/2}$. In what follows, we make $\mathcal{S}$ feasible by shifting some jobs in time.

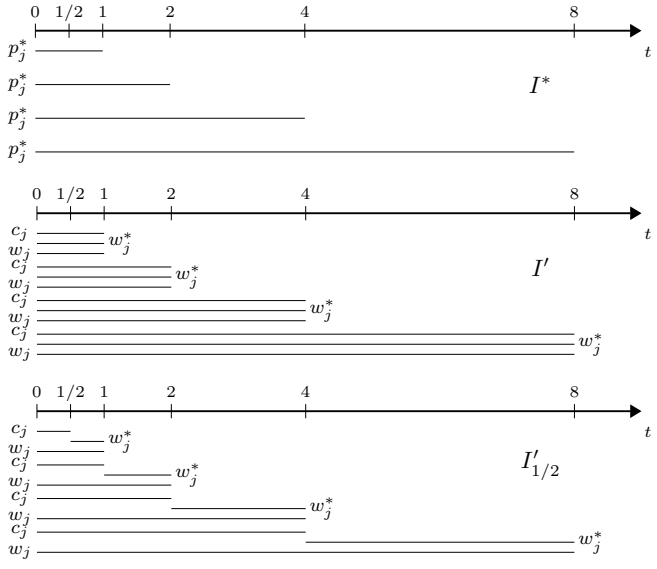

Figure 1: The figure shows the intervals of the three different instances. On the top, there are the intervals of instance I^*, in the middle, the intervals of instance I' and on the bottom, the intervals of the instance $I'_{1/2}$.

For each job $j \in A$, there is a job $(0, d_j, w_j)$ which is added both in $I'_{1/2}$ and in I'. For these jobs, their allocation in $\mathcal{S}$ is already feasible since they have the same active interval in I' and $I'_{1/2}$.

For each job $j \in B_{\ell}$, $0 \leqslant \ell \leqslant k$, the instance I' contains two jobs $(0, d_j, c_j)$ and $(0, d_j, w^*_j)$, while the instance $I'_{1/2}$ contains the jobs $(0, \frac{d_j}{2}, c_j)$ and $(\frac{d_j}{2}, d_j, w^*_j)$. Hence, in order to guarantee the feasibility of $\mathcal{S}$, we shift in $(\frac{d_j}{2}, d_j]$ the (parts of) jobs $(\frac{d_j}{2}, d_j, w^*_j)$ of $I'_{1/2}$ allocated in $(0, \frac{d_j}{2}]$. Similarly, we shift in $(0, \frac{d_j}{2}]$ the (parts of) jobs $(0, \frac{d_j}{2}, c_j)$ of $I'_{1/2}$ allocated in $(\frac{d_j}{2}, d_j]$. We make these shifts starting with the jobs having deadline 2^0, we continue with those having deadline 2^1, and so on.

We will prove by induction the following statement: "For each ℓ, $0 \leqslant \ell \leqslant k$, in the ℓ-th iteration of our shifting procedure, there is enough idle space in order to allocate all jobs $(0, 2^{\ell-1}, c_j)$ of $I'_{1/2}$ to the interval $(0, 2^{\ell-1}]$ and all jobs $(2^{\ell-1}, 2^{\ell}, w^*_j)$ of $I'_{1/2}$ to the interval $(2^{\ell-1}, 2^{\ell}]$".

- Basis: As explained before, the intervals $(\frac{1}{4}, \frac{1}{2}]$ and $(\frac{3}{4}, 1]$ in the schedule $\mathcal{S}$ are idle before any shifting due to the doubling of the speed. At the same time, the (parts of the) jobs $(0, \frac{1}{2}, c_j)$ which were infeasibly allocated after the doubling in $\mathcal{S}$ appear only in the interval $(\frac{1}{2}, \frac{3}{4}]$. Similarly, the (parts of the) jobs $(\frac{1}{2}, 1, w^*_j)$ which were infeasibly allocated after the doubling appear only in the interval $(0, \frac{1}{4}]$. Moreover, the speed during the whole interval $(0, 1]$ is constant, due to the YDS algorithm. Hence we can shift all the infeasible (parts of) jobs $(0, \frac{1}{2}, c_j)$ to the interval $(\frac{1}{4}, \frac{1}{2}]$ and all the infeasible (parts of) jobs $(\frac{1}{2}, 1, w^*_j)$ to the interval $(\frac{3}{4}, 1]$.
- Induction: Assume now that the statement is true for $\ell - 1$. Consider first the jobs $(0, 2^{\ell-1}, c_j)$ of $I'_{1/2}$. Some parts of these jobs may have been allocated in the interval $(2^{\ell-1}, 2^{\ell} - 2^{\ell-2}]$, making their execution infeasible. However, these parts are executed for at most $2^{\ell-2}$ time which corresponds exactly to the idle time during the interval $(0, 2^{\ell-1}]$. Thus, we can safely shift their execution to the left, since the speed used in $(0, 2^{\ell-1}]$ is at least the speed used in $(2^{\ell-1}, 2^{\ell} - 2^{\ell-2}]$, by the definition of the YDS algorithm, getting a feasible schedule for these jobs.
  Consider now the the jobs $(2^{\ell-1}, 2^{\ell}, w^*_j)$ of $I'_{1/2}$. Some parts of these jobs may have been allocated in the interval $(0, 2^{\ell-1}]$, making their execution infeasible. However, these parts are executed for at most $\frac{1}{4} + 2^{-2} + 2^{-1} + \cdots + 2^{\ell-3} = 2^{\ell-2}$ time which corresponds exactly to the idle time during the interval $(2^{\ell} - 2^{\ell-2}, 2^{\ell}]$. If the speed used in $(0, 2^{\ell-1}]$ is equal to the speed used in $(2^{\ell-1}, 2^{\ell}]$, then we can safely shift their execution to the right, getting a feasible schedule for these jobs. If the speed used in $(0, 2^{\ell-1}]$ is bigger than the speed used in $(2^{\ell-1}, 2^{\ell}]$, then the jobs $(0, 2^{\ell}, w^*_j)$ of I' are not executed at all during $(2^{\ell-1}, 2^{\ell}]$ in the optimal schedule for the instance I' obtained by the YDS algorithm, since they belong on a different critical interval. Hence, the jobs $(2^{\ell-1}, 2^{\ell}, w^*_j)$ of $I'_{1/2}$ are also not executed in $(2^{\ell-1}, 2^{\ell}]$. They are already feasible.

As a result, the schedule $\mathcal{S}$ is feasible for $I'_{1/2}$ after all the shifts, and it uses speed $s(t) = 2s^*_{I'}(t)$, for any time t. Then, the energy consumption $E(\mathcal{S})$ of $\mathcal{S}$ is at most two time the energy consumption

of the optimal solution for I'. Therefore, an optimal schedule for $I'_{1/2}$ will use even smaller energy, i.e., $E'_{1/2} \leqslant E(\mathcal{S}) \leqslant 2^\alpha E'$, and the lemma follows. □

Lemma 4.11. *Given an optimal schedule for the instance $I'_{1/2}$ and a schedule given by Algorithm 2, we have that $s(t) \leqslant 2s^*_{I'_{1/2}}(t)$ for each time instant t.*

Proof. By the construction of $I'_{1/2}$ and the definition of $\mathcal{Q}$ and $\mathcal{W}$ in Lines 1-5 of the algorithm, we have that $\mathcal{Q} \cup \mathcal{W} \subseteq I'_{1/2}$. In Line 6, an optimal schedule is created for the jobs in $\mathcal{Q} \cup \mathcal{W}$. Since both optimal solutions for $I'_{1/2}$ and for the jobs in $\mathcal{Q} \cup \mathcal{W}$ are computed by the YDS algorithm, and due to the properties of this algorithm, we have that $s^{YDS}(t) \leqslant s^*_{I'_{1/2}}(t)$, for each $t \in (0, 2^k]$.

Similarly, by the construction of $I'_{1/2}$ and the definition of $\mathcal{W}^*_\ell$'s in Lines 8-11 of the algorithm, we have that $\bigcup_{\ell=0}^{k} \mathcal{W}^*_\ell \subseteq I'_{1/2}$. Moreover, the jobs in $\mathcal{W}^*_\ell$, $0 \leqslant \ell \leqslant k$, are of the form $(\frac{2^\ell}{2}, 2^\ell, w^*_j)$, and hence the active intervals of any two jobs belonging to two different sets $\mathcal{W}^*_\ell$ and $\mathcal{W}^*_{\ell'}$ are time-disjoint. Thus, in an optimal solution for the jobs in $\bigcup_{\ell=0}^{k} \mathcal{W}^*_\ell$, the speed used during the interval $(\frac{2^\ell}{2}, 2^\ell]$ is $\sum_{j \in \mathcal{W}^*_\ell} \delta_j$. Therefore, using the same arguments as before, for each $t \in (\frac{2^\ell}{2}, 2^\ell]$, we have that $\sum_{j \in \mathcal{W}^*_\ell} \delta_j \leqslant s^*_{I'_{1/2}}(t)$.

For the speed of the algorithm, for any time $t \in (0, \frac{1}{2}]$, we have that $s(t) = s^{YDS}(t) \leqslant s^*_{I'_{1/2}}$ (see Line 7). Moreover, for any ℓ, $0 \leqslant \ell \leqslant k$, and any time $t \in (\frac{2^\ell}{2}, 2^\ell]$, we have that $s(t) = s^{YDS}(t) + \sum_{j \in \mathcal{W}^*_\ell} \delta_j \leqslant s^*_{I'_{1/2}}(t) + s^*_{I'_{1/2}}(t) \leqslant 2s^*_{I'_{1/2}}(t)$ (see Line 12), and the lemma follows. □

Corollary 4.12. *Let E and $E'_{1/2}$ be the energy consumption of the schedule created by Algorithm 2 and of an optimal schedule for the instance $I'_{1/2}$ respectively. Then, $E \leqslant 2^\alpha E'_{1/2}$.*

Theorem 4.13. *Algorithm 2 achieves a competitive ratio of $(4\phi)^\alpha$ with respect to energy.*

Proof. Note that the energy consumption of an optimal schedule for our original instance and of an optimal schedule for the instance I^* is exactly the same, as they contain exactly the same set of jobs with the same characteristics. Then, the proof of the theorem is an immediate consequence of Lemmas 4.9 and 4.10, and Corollary 4.12. Specifically, we have: $E \leqslant 2^\alpha E'_{1/2} \leqslant 4^\alpha E' \leqslant (4\phi)^\alpha E^*$. □

4.4 Common Release, Arbitrary Deadlines

In this section we adapt the previous result for jobs with arbitrary deadlines. Given an instance I of our original problem, we create an instance $\check{I}$ by rounding down the deadline of all jobs to a power of two: for each job $(r_j, d_j, c_j, w_j, w^*_j) \in \mathcal{J}$, add a job $(r_j, d'_j, c_j, w_j, w^*_j)$ in $\check{I}$, where $d'_j = \max\{2^i | 2^i \leqslant d_j\}$. Then, run Algorithm 2 using instance $\check{I}$ as input. We call this algorithm CRAD.

Lemma 4.14. *Let E and $\check{E}$ be the energy consumption of an optimal schedule for the instance I and $\check{I}$ respectively. Then, $\check{E} \leqslant 2^\alpha E$.*

Corollary 4.15. *CRAD achieves a competitive ratio of $(8\phi)^\alpha$ with respect to energy.*

5 ONLINE

In this section, we consider the QBSS model when the jobs arrive online and they should be executed on a single machine.

5.1 AVR with Queries

The online AVR algorithm for the classical speed scaling setting works as follows: at each time t, the machine runs at speed $s^{AVR}(t) = \sum_{j:t \in (r_j, d_j]} \delta_j$ and it executes the unfinished job with the smaller deadline which is released before t. Yao et al. [27] proved that AVR is $2^{\alpha-1}\alpha^\alpha$-competitive with respect to energy.

In this section we propose the online algorithm AVRQ, an adaptation of AVR to the QBSS model. AVRQ does the query for all the jobs by selecting as a splitting point the half of their interval. Specifically, for each job $(r_j, d_j, c_j, w_j, w^*_j)$ in $\mathcal{J}$, two jobs of the classical speed scaling setting are created and added to the set $\mathcal{J}'$ (in an online manner): the job $(r_j, \frac{r_j+d_j}{2}, c_j)$ at time r_j, and the job $(\frac{r_j+d_j}{2}, d_j, w^*_j)$ at time $\frac{r_j+d_j}{2}$. The AVR algorithm runs using as input the set of jobs $\mathcal{J}'$ which is created online.

The following lemma extends the lower bound for AVR proposed in [13] and gives a lower bound to the competitive ratio of AVRQ with respect to energy.

Lemma 5.1. *The competitive ratio of algorithm AVRQ is at least $(2\alpha)^\alpha$ with respect to energy.*

Let AVR* be the original AVR algorithm when executed using the set of jobs $\mathcal{J}^*$ created as follows: for each $j \in \mathcal{J}$, add the job (r_j, d_j, p^*_j) to $\mathcal{J}^*$. The following theorem compares, for each time t, the speed used by the algorithm AVRQ with the speed of AVR*.

Theorem 5.2. *For any time instant t, we have $s^{AVRQ}(t) \leqslant 2s^{AVR^*}(t)$.*

Proof. At any time t, the speed of AVRQ is

$$\begin{aligned} s^{AVRQ}(t) &\leqslant \sum_{j \in \mathcal{J}: t \in (r_j, d_j]} \max\left\{ \frac{c_j}{(d_j - r_j)/2}, \frac{w^*_j}{(d_j - r_j)/2} \right\} \\ &= 2 \sum_{j \in \mathcal{J}: t \in (r_j, d_j]} \frac{\max\{c_j, w^*_j\}}{d_j - r_j} \\ &\leqslant 2 \sum_{j \in \mathcal{J}: t \in (r_j, d_j]} \frac{\min\{w_j, c_j + w^*_j\}}{d_j - r_j} \\ &= 2 \sum_{j \in \mathcal{J}^*: t \in (r_j, d_j]} \frac{p^*_j}{d_j - r_j} = 2s^{\text{AVR}^*}(t) \end{aligned}$$ □

Corollary 5.3. *AVRQ is $2^{2\alpha-1}\alpha^\alpha$-competitive with respect to energy.*

5.2 BKP with Queries

The online BKP algorithm for the classical speed scaling setting works as follows: for the time instants t, t_1 and t_2 with $t_1 < t \leqslant t_2$, let $w(t, t_1, t_2)$ be the total work of jobs that have arrived by time t, have a release time of at least t_1 and a deadline of at most t_2. At any time t, the machine runs at speed $s^{BKP}(t) = e \max_{t_1, t_2} \frac{w(t, t_1, t_2)}{(t_2 - t_1)}$ and it executes the unfinished job with the smallest deadline which is released before t. Bansal et al. proved that BKP achieves a competitive

ratio of $2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$ with respect to energy, while it is e-competitive with respect to maximum speed.

In this section, we propose the online algorithm BKPQ, an adaptation of BKP to the QBSS model. For each job $(r_j, d_j, c_j, w_j, w_j^*)$ in $\mathcal{J}$, BKPQ decides to make the query only if $c_j \leqslant \frac{w_j}{\phi}$ using as splitting point $\tau_j = \frac{r_j+d_j}{2}$. Hence, in the case of a query, two jobs of the classical speed scaling setting, corresponding to $(r_j, d_j, c_j, w_j, w_j^*)$, are created and added to the set of jobs $\mathcal{J}'$ (in an online manner): the job $(r_j, \frac{r_j+d_j}{2}, c_j)$ at time r_j, and the job $(\frac{r_j+d_j}{2}, d_j, w_j^*)$ at time $\frac{r_j+d_j}{2}$. In the case where no query is made, then a single job, corresponding to $(r_j, d_j, c_j, w_j, w_j^*)$, is added to the set $\mathcal{J}'$: the job (r_j, d_j, w_j) at time r_j. The BKP algorithm runs using as input the set of jobs $\mathcal{J}'$ which is created online.

Let BKP* be the original BKP algorithm when executed using the set of jobs $\mathcal{J}^*$ created as follows: for each $j \in \mathcal{J}$, add the job (r_j, d_j, p_j^*) to $\mathcal{J}^*$. The following theorem compares, for each time t, the speed used by the algorithm BKPQ with the speed of BKP*.

Theorem 5.4. *For any time instant t, we have $s^{BKPQ}(t) \leqslant (2+\phi)s^{BKP^*}(t)$.*

Proof. Let t_1 and t_2 be time instants such that

$$\frac{w(t,t_1,t_2)}{(t_2-t_1)} = \max_{t_1',t_2'} \frac{w(t,t_1',t_2')}{(t_2'-t_1')}$$

for the jobs in $\mathcal{J}'$. We define three disjoint subsets of $\mathcal{J}$ and we explain how the corresponding jobs in $\mathcal{J}'$ contribute to $w(t,t_1,t_2)$:

- Let $\mathcal{L}$ be the set of queried jobs whose queries are entirely processed in the interval $(t_1,t_2]$, but not its exact loads themselves, and start before time t.
- Let $\mathcal{R}$ be the set of queried jobs whose exact load is entirely processed in the interval $(t_1,t_2]$, but not its queries themselves, and start before time t.
- Let C be the set of jobs (corresponding to queries, exact loads or initial workloads) that are entirely processed in the interval $(t_1,t_2]$ and start before time t.

Let $W(\mathcal{L}) = \sum_{j\in\mathcal{L}} c_j$, $W(\mathcal{R}) = \sum_{j\in\mathcal{R}} w_j^*$ and $W(C) = \sum_{j\in C} p_j$ be the total work of jobs in $\mathcal{J}'$ which belong to $\mathcal{L}$, $\mathcal{R}$ and C, respectively. By definition, $W(\mathcal{L})+W(\mathcal{R})+W(C)$ describes the total work that is executed in $(t_1,t_2]$ by BKPQ. So, for any time $t \in (t_1,t_2]$, we have that $s^{BKPQ}(t) = \frac{W(\mathcal{L})+W(\mathcal{R})+W(C)}{(t_2-t_1)}$.

In a similar way, let $W^*(\mathcal{L}) = \sum_{j\in\mathcal{L}} p_j^*$, $W^*(\mathcal{R}) = \sum_{j\in\mathcal{R}} p_j^*$ and $W(C) = \sum_{j\in C} p_j^*$. We consider the following three bounds:

(1) Consider a job $j \in \mathcal{L}$. If $p_j^* = w_j$ then $c_j \leqslant \frac{w_j}{\phi} \leqslant w_j = p_j^*$. If $p_j^* = c_j + w_j^*$ then $c_j \leqslant c_j + w_j^* = p_j^*$. Thus, for each $j \in \mathcal{L}$ we have $c_j \leqslant p_j^*$ and $W(\mathcal{L}) \leqslant W^*(\mathcal{L})$.
(2) Consider a job $j \in \mathcal{R}$. If $p_j^* = w_j$ then $w_j^* \leqslant w_j = p_j^*$. If $p_j^* = c_j + w_j^*$ then $w_j^* \leqslant c_j + w_j^* = p_j^*$. Thus, for each $j \in \mathcal{R}$ we have $w_j^* \leqslant p_j^*$ and $W(\mathcal{R}) \leqslant W^*(\mathcal{R})$.
(3) For each $j \in C$ we have $p_j \leqslant \phi p_j^*$ by using Lemma 3.1. Thus, $W(C) \leqslant \phi W^*(C)$.

Consider now the time interval (t_0,t_3) such as $t_0 = \max\{0, 2t_1 - t_2\}$ and $t_3 = 2t_2 - t_1$. For each $j \in \mathcal{L}$ we have $t_1 \leqslant r_j \leqslant t < \tau_j \leqslant t_2$, and hence $d_j = 2\tau_j - r_j \leqslant 2t_2 - t_1 = t_3$. For each $j \in \mathcal{R}$ we have $t_1 \leqslant \tau_j$ and $d_j \leqslant t_2$, and hence $r_j = 2\tau_j - d_j \geqslant \max\{0, 2t_1 - t_2\} = t_0$. Therefore, each job $j \in \mathcal{L}\cup\mathcal{R}\cup C$ should be executed in the interval $[t_0,t_3]$ by any algorithm and also by BKP*. As a result, we have

$$\begin{aligned} s^{\text{BKP}^*}(t) &\geqslant \frac{W^*(\mathcal{L}) + W^*(\mathcal{R}) + W^*(C)}{(t_3-t_0)} \\ &= \frac{W^*(\mathcal{L}) + W^*(\mathcal{R}) + W^*(C)}{3(t_2-t_1)} \end{aligned} \tag{1}$$

As explained before, for the speed of BKPQ at any time $t \in (t_1,t_2]$ we have

$$\begin{aligned} s^{BKPQ}(t) &= \frac{W(\mathcal{L}) + W(\mathcal{R}) + W(C)}{(t_2-t_1)} \\ &\leqslant \frac{W^*(\mathcal{L}) + W^*(\mathcal{R}) + \phi W^*(C)}{(t_2-t_1)} \\ &= \frac{W^*(\mathcal{L}) + W^*(\mathcal{R}) + W^*(C) + (\phi-1)W^*(C)}{(t_2-t_1)} \\ &\leqslant 3s^{\text{BKP}^*}(t) + (\phi-1)s^{\text{BKP}^*}(t) = (2+\phi)s^{\text{BKP}^*}(t) \end{aligned}$$

The first inequality follows by the three bounds presented above. In the next line we just add and subtract $W^*(C)$. For the last inequality we use Inequality 1, as well as the fact that $s^{\text{BKP}^*}(t) \geqslant \frac{W^*(C)}{(t_2-t_1)}$ since all jobs in the set C are entirely executed in the interval $(t_1,t_2]$ by any algorithm and also by BKP*. □

Corollary 5.5. *BKPQ is $(2+\phi)^\alpha 2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$-competitive with respect to energy, and $(2+\phi)e$-competitive with respect to maximum speed.*

6 MULTIPLE MACHINES

In this section we adapt to the QBSS model the online AVR(m) algorithm proposed by Albers et al. [2] for the classical speed scaling setting on a set of m parallel identical machines $\mathcal{M}$. AVR(m) is $(2^{\alpha-1}\alpha^\alpha+1)$-competitive with respect to energy consumption.

For completeness, we briefly present AVR(m). The algorithm works online per each unit time slot $(t,t+1]$ and it schedules δ_j amount of work from each active job during $(t,t+1]$. Let $\mathcal{J}_t$ be the set of active jobs in $(t,t+1]$. Moreover, let $U \subseteq \mathcal{J}_t$ be the unscheduled jobs of $\mathcal{J}_t$ and $R \subseteq \mathcal{M}$ be the remaining unused machines at each step of the algorithm. In the beginning, we set $U = \mathcal{J}_t$ and $R = \mathcal{M}$. We denote by $\Delta = \sum_{j\in U}\delta_j$ the total work of the jobs in U. The jobs in U will be characterized as *big* or *small* depending on their densities. Intuitively, each *big* job will occupy one machine during the whole slot $(t,t+1]$, while *small* jobs will share the remaining machines in $(t,t+1]$. The algorithm searches in an iterative way the job $\hat{j} = \text{argmax}\{\delta_j : j \in U\}$ with the maximum density in U and if $\delta_{\hat{j}} > \frac{\Delta}{|R|}$ then it is characterized as a *big* one. In this case, the algorithm schedules $\hat{j}$ with speed $\delta_{\hat{j}}$ in the machine of the lower index in R which is then removed from R. Moreover, the algorithm updates $\mathcal{J}_t = \mathcal{J}_t \setminus \{\hat{j}\}$ and it searches for the next *big* job. If no big job exists, then all the remaining jobs are *small* and they are allocated to the remaining machines using speed $\frac{\Delta}{|R|}$. Note that, at each time moment the speed of a machine with lower index is not less than the speed of a machine with larger index.

In this section we propose the online algorithm AVRQ(m) which makes the query for all jobs by selecting as a splitting point the half of their interval. Specifically, for each job $(r_j, d_j, c_j, w_j, w_j^*)$ in $\mathcal{J}$,

two jobs of the classical speed scaling setting are created and added to the set $\mathcal{J}'$ (in an online manner): the job $\zeta(j) = (r_j, \frac{r_j+d_j}{2}, c_j)$ at time r_j, and the job $\zeta'(j) = (\frac{r_j+d_j}{2}, d_j, w_j^*)$ at time $\frac{r_j+d_j}{2}$. The AVR(m) algorithm runs using as input the set of jobs $\mathcal{J}'$ which is created online.

We start our analysis with two technical lemmas.

Lemma 6.1. *Let two sets of non-negative rational numbers $A = \{a_1, ..., a_n\}$ and $B = \{b_1, ..., b_n\}$ be given such that $b_j \leqslant 2a_j$ for all $j = 1, \ldots, n$. Let π_A and π_B be permutations of numbers from A and B, respectively, in which the numbers are ordered in non-increasing order. Then $\pi_B(i) \leqslant 2\pi_A(i)$ for all $i = 1, \ldots, n$.*

Lemma 6.2. *Let a sequence of non-negative rational numbers $a_1, ..., a_n$ and an integer $m \geqslant 2$ be given. If $a_1 > \frac{\sum_{i=1}^n a_i}{m}$, then $\frac{\sum_{i=1}^n a_i}{m} > \frac{\sum_{i=2}^n a_i}{m-1}$, otherwise, $\frac{\sum_{i=1}^n a_i}{m} \leqslant \frac{\sum_{i=2}^n a_i}{m-1}$.*

Let AVR$^*(m)$ be the original AVR(m) algorithm when executed using the set of jobs $\mathcal{J}^*$ created as follows: for each $j \in \mathcal{J}$, add the job (r_j, d_j, p_j^*) to $\mathcal{J}^*$. The following theorem compares, for each time t, the speed used by the algorithm AVRQ(m) with the speed of AVR$^*(m)$.

Theorem 6.3. *For any time instant t, and any machine i, we have $s_i^{AVRQ(m)}(t) \leqslant 2s_i^{AVR^*(m)}(t)$.*

Proof. We consider the set of jobs $\mathcal{J}''$ which is produced from $\mathcal{J}^*$ by replacing each job (r_j, d_j, p_j^*) with two jobs, $\psi(t) = (r_j, \frac{r_j+d_j}{2}, \frac{p_j^*}{2})$ and $\psi'(t) = (\frac{r_j+d_j}{2}, d_j, \frac{p_j^*}{2})$. Since the number of jobs and their densities in each unit time slot $(t, t+1]$ do not change, then the speed of the machines in the schedules obtained by AVR(m) when applied to the sets of jobs $\mathcal{J}^*$ and $\mathcal{J}''$ does not change either.

Algorithm AVRQ(m) also creates two jobs, let $\zeta(j)$ and $\zeta'(j)$ for each original job $j \in \mathcal{J}$ using the same intervals as in $\mathcal{J}''$. Hence, the number of active jobs in $\mathcal{J}'$ and $\mathcal{J}''$ is the same at each unit time slot, and by definition we have

$$\delta_{\zeta(j)} \leqslant 2\delta_{\psi(j)} \text{ and } \delta_{\zeta'(j)} \leqslant 2\delta_{\psi'(j)} \text{ for all } j \in \mathcal{J} \qquad (2)$$

since $c_j \leqslant p_j^*$ and $w_j^* \leqslant p_j^*$.

Denote by $\mathcal{J}_t'' \subseteq \mathcal{J}''$ and $\mathcal{J}_t' \subseteq \mathcal{J}'$ the set of active jobs in the unit slot $(t, t+1]$. We order the jobs in each set in non-increasing densities. Note that $|\mathcal{J}_t''| = |\mathcal{J}_t'| = r$. Let a_j be the density of the j-th job in $\mathcal{J}''$ and b_j be the density of the j-th job in $\mathcal{J}'$. Lemma 6.1 and Inequalities (2) imply that $b_j \leqslant 2a_j$.

Let k be the number of *big* jobs in the set $\mathcal{J}_t''$ and ℓ be the number of *big* jobs in the set $\mathcal{J}_t'$. We consider three cases.

(1) Case $k = \ell$. For each machine $i \leqslant k$ we have:

$$s_i^{\text{AVRQ}(m)}(t) = b_i \leqslant 2a_i = 2s_i^{\text{AVR}^*(m)}(t)$$

For each machine $i > k$ we have:

$$s_i^{\text{AVRQ}(m)}(t) = \frac{\sum_{j=\ell+1}^r b_j}{m-\ell} \leqslant 2\frac{\sum_{j=k+1}^r a_j}{m-k} \leqslant 2s_i^{\text{AVR}^*(m)}(t)$$

(2) Case $k > \ell$. For each machine $i \leqslant \ell$ we have:

$$s_i^{\text{AVRQ}(m)}(t) = b_i \leqslant 2a_i = 2s_i^{\text{AVR}^*(m)}(t)$$

For each machine $i > \ell$ we have $b_h \leqslant \frac{\sum_{j=h}^r b_j}{m-h+1}$ for $\ell+1 \leqslant h \leqslant k$. Successively applying Lemma 6.2 we get:

$$\frac{\sum_{j=\ell+1}^r b_j}{m-\ell} \leqslant \frac{\sum_{j=\ell+2}^r b_j}{m-\ell-1} \leqslant \cdots \leqslant \frac{\sum_{j=k+1}^r b_j}{m-k}$$

Hence, we obtain:

$$s_i^{\text{AVRQ}(m)}(t) = \frac{\sum_{j=\ell+1}^r b_j}{m-\ell} \leqslant \frac{\sum_{j=k+1}^r b_j}{m-k} \leqslant 2\frac{\sum_{j=k+1}^r a_j}{m-k} \leqslant 2s_i^{\text{AVR}^*(m)}(t)$$

(3) Case $k < \ell$. For each machine $i \leqslant k$ we have

$$s_i^{\text{AVRQ}(m)}(t) = b_i \leqslant 2a_i = 2s_i^{\text{AVR}^*(m)}(t)$$

For each machine i, $k < i \leqslant \ell$, we have

$$s_i^{\text{AVRQ}(m)}(t) = b_i \leqslant 2a_i \leqslant 2\frac{\sum_{j=k+1}^r a_j}{m-k} = 2s_i^{\text{AVR}^*(m)}(t)$$

where the last inequality follows by the definition of AVR$^*(m)$.

For each machine $i > \ell$ we have $b_h > \frac{\sum_{j=h}^r b_j}{m-h+1}$ for $k+1 \leqslant h \leqslant \ell$. Successively applying Lemma 6.2 we get:

$$\frac{\sum_{j=k+1}^r b_j}{m-k} > \frac{\sum_{j=\ell+2}^r b_j}{m-k-1} > \cdots > \frac{\sum_{j=\ell+1}^r b_j}{m-\ell}$$

Hence, we obtain:

$$s_i^{\text{AVRQ}(m)}(t) \leqslant \frac{\sum_{j=\ell+1}^r b_j}{m-\ell} < \frac{\sum_{j=k+1}^r b_j}{m-k} \leqslant 2\frac{\sum_{j=k+1}^r a_j}{m-k} \leqslant 2s_i^{\text{AVR}^*(m)}(t)$$

and the theorem follows. □

Corollary 6.4. *AVRQ(m) is $2^\alpha(2^{\alpha-1}\alpha^\alpha + 1)$-competitive with respect to energy.*

7 CONCLUSION

In this paper, we studied an enhanced speed scaling setting, where queries can be additionally executed in the system in order to reveal a more accurate value of the workload of jobs. The main minimization objective was the energy consumption, while the maximum speed minimization was also studied. We proposed various lower bounds for the offline and the online settings. In particular, we showed how to use known online algorithms (AVR and BKP) of the classical speed scaling context in the speed scaling with explorable uncertainty setting. Notice also that our approach can directly be applied to the preemptive-non-migratory variant of the problem [21]. An interesting open question is whether the Optimal Available (OA) algorithm of [27] could be extended in this new framework.

ACKNOWLEDGMENTS

This work was partially supported by the French National Research Agency (Energumen ANR-18-CE25-0008 and Algoridam ANR-19-CE48-0016). The research of the third author was supported by the Russian Science Foundation RSF-ANR 21-41-09017.

REFERENCES

[1] Susanne Albers. 2010. Energy-efficient algorithms. *Commun. ACM* 53, 5 (2010), 86–96. https://doi.org/10.1145/1735223.1735245

[2] Susanne Albers, Antonios Antoniadis, and Gero Greiner. 2015. On multi-processor speed scaling with migration. *J. Comput. Syst. Sci.* 81, 7 (2015), 1194–1209. https://doi.org/10.1016/j.jcss.2015.03.001

[3] Susanne Albers, Evripidis Bampis, Dimitrios Letsios, Giorgio Lucarelli, and Richard Stotz. 2017. Scheduling on power-heterogeneous processors. *Inf. Comput.* 257 (2017), 22–33. https://doi.org/10.1016/j.ic.2017.09.013

[4] Susanne Albers and Alexander Eckl. 2020. Explorable Uncertainty in Scheduling with Non-Uniform Testing Times. *CoRR* abs/2009.13316 (2020). arXiv:2009.13316 https://arxiv.org/abs/2009.13316

[5] Susanne Albers, Fabian Müller, and Swen Schmelzer. 2014. Speed Scaling on Parallel Processors. *Algorithmica* 68, 2 (2014), 404–425. https://doi.org/10.1007/s00453-012-9678-7

[6] Eric Angel, Evripidis Bampis, Fadi Kacem, and Dimitrios Letsios. 2019. Speed scaling on parallel processors with migration. *J. Comb. Optim.* 37, 4 (2019), 1266–1282. https://doi.org/10.1007/s10878-018-0352-0

[7] Luciana Arantes, Evripidis Bampis, Alexander V. Kononov, Manthos Letsios, Giorgio Lucarelli, and Pierre Sens. 2018. Scheduling under Uncertainty: A Query-based Approach. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, Jérôme Lang (Ed.). ijcai.org, 4646–4652. https://doi.org/10.24963/ijcai.2018/646

[8] Evripidis Bampis. 2016. Algorithmic Issues in Energy-Efficient Computation. In *Discrete Optimization and Operations Research - 9th International Conference, DOOR 2016, Vladivostok, Russia, September 19-23, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9869)*, Yury Kochetov, Michael Khachay, Vladimir L. Beresnev, Evgeni A. Nurminski, and Panos M. Pardalos (Eds.). Springer, 3–14. https://doi.org/10.1007/978-3-319-44914-2_1

[9] Evripidis Bampis, Alexander V. Kononov, Dimitrios Letsios, Giorgio Lucarelli, and Ioannis Nemparis. 2015. From preemptive to non-preemptive speed-scaling scheduling. *Discret. Appl. Math.* 181 (2015), 11–20. https://doi.org/10.1016/j.dam.2014.10.007

[10] Evripidis Bampis, Alexander V. Kononov, Dimitrios Letsios, Giorgio Lucarelli, and Maxim Sviridenko. 2018. Energy-efficient scheduling and routing via randomized rounding. *J. Sched.* 21, 1 (2018), 35–51. https://doi.org/10.1007/s10951-016-0500-2

[11] Evripidis Bampis, Dimitrios Letsios, and Giorgio Lucarelli. 2015. Green scheduling, flows and matchings. *Theor. Comput. Sci.* 579 (2015), 126–136. https://doi.org/10.1016/j.tcs.2015.02.020

[12] Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. 2011. Average Rate Speed Scaling. *Algorithmica* 60, 4 (2011), 877–889. https://doi.org/10.1007/s00453-009-9379-z

[13] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. 2007. Speed scaling to manage energy and temperature. *J. ACM* 54, 1 (2007), 3:1–3:39. https://doi.org/10.1145/1206035.1206038

[14] Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. 2018. Scheduling with Explorable Uncertainty. In *ITCS 2018*. 30:1–30:14. https://doi.org/10.4230/LIPIcs.ITCS.2018.30

[15] Thomas Erlebach and Michael Hoffmann. 2015. Query-Competitive Algorithms for Computing with Uncertainty. *Bulletin of the EATCS* 116 (2015). http://eatcs.org/beatcs/index.php/beatcs/article/view/335

[16] Tomás Feder, Rajeev Motwani, Liadan O'Callaghan, Chris Olston, and Rina Panigrahy. 2007. Computing shortest paths with uncertainty. *J. Algorithms* 62, 1 (2007), 1–18. https://doi.org/10.1016/j.jalgor.2004.07.005

[17] Tomás Feder, Rajeev Motwani, Rina Panigrahy, Chris Olston, and Jennifer Widom. 2003. Computing the Median with Uncertainty. *SIAM J. Comput.* 32, 2 (2003), 538–547. https://doi.org/10.1137/S0097539701395668

[18] Bruno Gaujal, Alain Girault, and Stéphan Plassart. 2020. Dynamic speed scaling minimizing expected energy consumption for real-time tasks. *J. Sched.* 23, 5 (2020), 555–574. https://doi.org/10.1007/s10951-020-00660-9

[19] Marco E. T. Gerards, Johann L. Hurink, and Philip K. F. Hölzenspies. 2016. A survey of offline algorithms for energy minimization under deadline constraints. *J. Sched.* 19, 1 (2016), 3–19. https://doi.org/10.1007/s10951-015-0463-8

[20] Marc Goerigk, Manoj Gupta, Jonas Ide, Anita Schöbel, and Sandeep Sen. 2015. The robust knapsack problem with queries. *Computers & OR* 55 (2015), 12–22. https://doi.org/10.1016/j.cor.2014.09.010

[21] Gero Greiner, Tim Nonner, and Alexander Souza. 2014. The Bell Is Ringing in Speed-Scaled Multiprocessor Scheduling. *Theory Comput. Syst.* 54, 1 (2014), 24–44. https://doi.org/10.1007/s00224-013-9477-9

[22] Manoj Gupta, Yogish Sabharwal, and Sandeep Sen. 2011. The update complexity of selection and related problems. In *IARCS FSTTCS 2011*. 325–338. https://doi.org/10.4230/LIPIcs.FSTTCS.2011.325

[23] Michael Hoffmann, Thomas Erlebach, Danny Krizanc, Matús Mihalák, and Rajeev Raman. 2008. Computing Minimum Spanning Trees with Uncertainty. In *STACS 2008*. 277–288. https://doi.org/10.4230/LIPIcs.STACS.2008.1358

[24] Simon Kahan. 1991. A Model for Data in Motion. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, Cris Koutsougeras and Jeffrey Scott Vitter (Eds.). ACM, 267–277. https://doi.org/10.1145/103418.103449

[25] Nicole Megow, Julie Meißner, and Martin Skutella. 2015. Randomization Helps Computing a Minimum Spanning Tree under Uncertainty. In *Algorithms - ESA 2015*. 878–890. https://doi.org/10.1007/978-3-662-48350-3_73

[26] Chris Olston and Jennifer Widom. 2000. Offering a Precision-Performance Tradeoff for Aggregation Queries over Replicated Data. In *VLDB 2000*. 144–155. http://www.vldb.org/conf/2000/P144.pdf

[27] F. Frances Yao, Alan J. Demers, and Scott Shenker. 1995. A Scheduling Model for Reduced CPU Energy. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*. IEEE Computer Society, 374–382. https://doi.org/10.1109/SFCS.1995.492493