Speed Scaling in the Non-clairvoyant Model

[Extended Abstract]

Yossi Azar
Tel Aviv University
Tel Aviv, Israel
azar@tau.ac.il

Nikhil R. Devanur
Microsoft Research
Redmond, WA
nikdev@microsoft.com

Zhiyi Huang
University of Hong Kong
Hong Kong
zhiyi@cs.hku.hk

Debmalya Panigrahi
Duke University
Durham, NC
debmalya@cs.duke.edu

ABSTRACT

In recent years, there has been a growing interest in speed scaling algorithms, where a set of jobs need to be scheduled on a machine with variable speed so as to optimize the flow-times of the jobs and the energy consumed by the machine. A series of results have culminated in constant-competitive algorithms for this problem in the clairvoyant model, i.e., when job parameters are revealed on releasing a job (Bansal, Pruhs, and Stein, SODA 2007; Bansal, Chan, and Pruhs, SODA 2009). Our main contribution in this paper is the first constant-competitive speed scaling algorithm in the non-clairvoyant model, which is typically used in the scheduling literature to model practical settings where job volume is revealed only after the job has been completely processed. Unlike in the clairvoyant model, the speed scaling problem in the non-clairvoyant model is non-trivial *even for a single job*. Our non-clairvoyant algorithm is defined by using the existing clairvoyant algorithm in a novel inductive way, which then leads to an inductive analytical tool that may be of independent interest for other online optimization problems. We also give additional algorithmic results and lower bounds for speed scaling on multiple identical parallel machines.

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problems**]: [Sequencing and scheduling]

General Terms

Theory

Keywords

Scheduling, Energy efficiency, Online algorithms

1. INTRODUCTION

SPAA'15, June 13–15, 2015, Portland, OR, USA.

ACM 978-1-4503-3588-1/15/06 ...$15.00.
DOI: http://dx.doi.org/10.1145/2755573.2755582.

Scheduling jobs released over time on one or more machines is one of the most fundamental optimization problems. A standard objective has been to minimize the average *response time* for a given set of jobs, while the devices ran at their fastest possible speed. (The technical term used for response time is *flow-time*, which for a job is the duration of time between its release and completion.) However, over the years, the *energy* consumed by the processors has become an important consideration. It was observed by Barroso [1] that in data centers, the raw performance and the performance per price of a server have been steadily growing but the performance per power consumption has remained flat. This increasing energy consumption coupled with rising prices of energy has made the cost of energy an important consideration in the design of computing infrastructure such as data centers.

One of the main algorithmic approaches to energy management is via *dynamic speed scaling*,[1] where a processor can be run at different speeds. Higher speeds finish jobs faster (improving response time) but consume more energy. The instantaneous power (which is the rate of energy consumed) is a given function of the speed, typically the cube of the speed, or more generally, speed$^{\alpha}$ for some fixed constant $\alpha \geq 2$. The typical objective used in the literature (introduced by Albers and Fujiwara [2]) is to minimize a linear combination of energy and weighted flow-time.[2] Jobs are released over time, each job comes with a volume and a weight, and the algorithm does not know which jobs will be released in the future. The algorithmic question posed by this problem has two components: which job to schedule next, and what speed to run the machine at. We call this the speed scaling problem. After a considerable amount of research [2, 5, 6, 7, 8, 9], it is now known that the answer to the first question is to schedule the job with the highest density first (HDF, density = weight/volume) and the answer to the second question is to pick a speed so that the power consumption equals the remaining weight of the jobs.

However, most of the results for this problem are in the *clairvoyant* setting where the algorithm knows the volume of a job when it is released. A more difficult, but in many cases more realistic, problem is one where the volume is known only when the job is completed — this is called the *non-clairvoyant* setting (introduced

[1]Other approaches have been considered, for example a power down model where a machine transitions to a low power state when idle, etc.

[2]Other problems such as minimizing energy while finishing jobs within given deadlines [3] and minimizing flow-time with a hard energy budget [4] have also been considered.

by Motwani, Phillips, and Torng [10] in classical scheduling). Our main contribution in this paper is to consider the speed scaling problem in the non-clairvoyant setting.

A particular application that motivated this study comes from cloud computing. Typically, a customer pays at a rate $(\lambda - \rho t_{\text{delay}})$ for each unit volume of a submitted job to the cloud service provider, where λ and ρ are predetermined payment and penalty rates for the job, and t_{delay} is the delay in processing the job, i.e., the difference between the actual and expected duration of the job. Note that the only term in the total payment for a job that is affected by the scheduling algorithm is $\rho F_{int}[j]V[j]$, where $F_{int}[j]$ is the flow-time of the job and $V[j]$ is its volume. This can be interpreted as a *weighted flowtime*, where the weight is $\rho V[j]$. (Since the penalty rate ρ = weight/volume, we call it the *density*.) Since $V[j]$ is unknown to the algorithm when a job is released, but ρ is known, this is the case of known density and unknown weight. We consider two scenarios: one where all jobs have the same density and a more general one where the density is job-dependent and revealed when a job is released.

1.1 Our Results

Before describing our results, let us formally define the notion of flow-time. The *integral* (weighted) flow-time of a job is the difference of its completion and release times multiplied by its weight. The integral flow-time of a processing schedule is the sum of the integral flow-times of the individual jobs in the schedule. In defining the *fractional* flow-time of a job, we imagine breaking up the job into infinitesimal pieces each of which suffers a different flow-time based on when it is completed. We consider both fractional flow-time plus energy, henceforth called *fractional objective*, and integral flow-time plus energy, henceforth called *integral objective*.

Our main results are the first constant-competitive algorithms for non-clairvoyant scheduling with known densities on a single machine. We consider any power function of the form $P(s) = s^\alpha$ and give deterministic algorithms. These results are also summarized in Table 1.

- In the case of *uniform job densities*, we present a $(3 + \frac{1}{\alpha-1})$-competitive algorithm for the integral objective and a $(2 + \frac{1}{\alpha-1})$-competitive deterministic algorithm for the fractional objective.

  Contrast this with a competitive ratio of $\frac{2\alpha^2}{\log \alpha}$ for the integral objective in the non-clairvoyant model with known *weights*, for uniform (unit) weight jobs by Chan *et al* [11]. The best known results for the *clairvoyant* model with uniform density achieve competitive ratios of 4 and 2 respectively for integral and fractional flow-time [5, 8].[3]

- We show an $O(1)$-competitive algorithm for the general problem with non-uniform densities for both the fractional and integral objectives. The constant depends exponentially on α.

  In the non-clairvoyant model with known weights, no results are known for arbitrary weights, except in the special case where all jobs are released at time 0, for which a $(2 - \frac{1}{\alpha})^2$ competitive algorithm is given by Lam *et al* [7]. The best known results for the clairvoyant model for arbitrary weighted flow-time achieve a competitive ratio of 2 for the fractional objective (due to Bansal, Chan, and Pruhs [8]) and $O(\frac{\alpha}{\log \alpha})$ for the integral objective (due to Bansal, Pruhs, and Stein [5] and Bansal, Chan, and Pruhs [8]).[4]

We also consider the problem of scheduling jobs on identical parallel machines. In the literature, two models for dispatching jobs to machines have been considered depending on whether the algorithm is required to do *immediate dispatch*, i.e., a job has to be assigned to a machine immediately on release. In either model, once a job has started processing on a machine, it cannot be *migrated to* any other machine in the future.

- We give an $O(\alpha)$-competitive deterministic algorithm for both the integral and fractional objectives in the case of uniform densities without immediate dispatch.

  No results are known for multiple machines in the non-clairvoyant model with known weights, even for unit weight jobs. Our results almost match the best known competitive ratios for the clairvoyant model (with immediate dispatch), which are $O(\alpha)$ and $O(\alpha^2)$ respectively for the fractional and the integral objective obtained by Anand, Garg, and Kumar [12].[5]

- We give a superconstant lower bound on the competitive ratio of any deterministic algorithm even for fractional flow-time in the case of uniform densities with immediate dispatch.

An interesting open problem is whether our results can be extended to the case of non-uniform density and multiple machines. Both the algorithm for the uniform density case and the lower bound for the immediate dispatch case do not extend readily to this case.

1.2 Techniques and Intuition

The non-clairvoyant problem with known densities is non trivial even in the case of a *single* job. The optimal speed could vary greatly with the processing volume of the job; therefore, the algorithm has to continuously adapt as it learns more about the volume of the job. Furthermore, in the case of multiple jobs, the choice of the job to process affects the information that the algorithm obtains, which in turn affects the speed of the machine. The natural choice for information gathering turns out to be first-in first-out (FIFO) but the clairvoyant algorithms suggest the HDF rule; most of the difficulty we encounter is due to the conflict between the FIFO and the HDF rules. A more detailed discussion of this conflict and how we deal with it is presented later.

In order to give a glimpse of the techniques used, we describe the simplest scenario of a single job here. On any instance I, we denote the optimal objective by $\mathsf{opt}(I)$, and that produced by our algorithm by $\mathsf{algo}^{(NC)}(I)$. The online problem is a game between the adversary and the algorithm with strategies that are functions of (continuous) time, where at every moment of time the adversary must declare whether the job has been completely processed or not and the algorithm responds to this declaration by setting the current speed of the machine. (The instance ends when the adversary declares that the job is completely processed.) Therefore, the algorithm must *continuously* ensure that the objective value of its schedule, $\mathsf{algo}^{(NC)}(I(t))$, is competitive against the optimal solution, $\mathsf{opt}(I(t))$, for the *current instance* $I(t)$ defined by the volume

[3] Our competitive ratio of $3 + \frac{1}{\alpha-1}$ for the integral objective in the non-clairvoyant model improves the best-known clairvoyant competitive ratio of 4 in uniform density case for $\alpha > 2$.

[4] To achieve the $O(\frac{\alpha}{\log \alpha})$ competitive ratio, one needs to combine the 2-competitive algorithm of [8] for the fractional objective with the reduction from the integral objective to the fractional objective in [5]. See, e.g., [12] for the relevant discussions.

[5] This has been improved to $O(\frac{\alpha}{\log \alpha})$ for both objectives in [13]. Otherwise, we seem to be improving [12] for integral flow-time, although the results of [12] are in a much stronger model, with unrelated machines, arbitrary weights and immediate dispatch.

Table 1: Summary of Results

	Clairvoyant	Non-clairvoyant known weight	Non-clairvoyant know density (this paper)
Integral unit density/weight	4 (unit density) [5] 3 (unit weight) [8]	$\frac{2\alpha^2}{\ln \alpha}$ [11] (unit weight)	$3 + \frac{1}{\alpha - 1}$ (unit density)
Fractional unit density/weight	2 [8]		$2 + \frac{1}{\alpha - 1}$ (unit density)
Integral arbitrary density/weight	$O\left(\frac{\alpha}{\log \alpha}\right)$ [8, 5]	$\left(2 - \frac{1}{\alpha}\right)^2$ [7] (jobs at time 0)	$2^{O(\alpha)}$
Fractional arbitrary density/weight	2 [8]		$2^{O(\alpha)}$

processed till the current time t. This naturally leads to an inductive analytical framework where we bound the rate of change (w.r.t. time) of the algorithmic solution against the rate of change of the optimal solution. If we can show that for any time t,

$$\frac{d\mathsf{algo}^{(NC)}(I(t))}{dt} \leq \Gamma \cdot \frac{d\mathsf{opt}(I(t))}{dt}, \tag{1}$$

then we have a competitive ratio of Γ. Unfortunately we do not have a good handle on how the optimal solution evolves with time (in the general case). We therefore use a *surrogate* of the optimal solution, the solution produced by the clairvoyant algorithm $\mathsf{algo}^{(C)}$ defined by the speed setting rule instantaneous power = remaining weight, or $P = \overline{W}$, *on the current instance*. Since this algorithm is known to be 2-competitive [8], we show that for any time t,

$$\frac{d\mathsf{algo}^{(NC)}(I(t))}{dt} \leq \Gamma' \cdot \frac{d\mathsf{algo}^{(C)}(I(t))}{dt}, \tag{2}$$

and obtain a competitive ratio of $2\Gamma'$.

Let us now take a closer look at Eqn. (1). If $s(t)$ is the speed of the machine at time t, then the incremental volume of the job processed in the interval $[t, t + dt]$ is $s(t)dt$.[6] The corresponding change in the algorithmic objective, $d\mathsf{algo}^{(NC)}(I(t))$ comprises two parts: the additional energy $P(s(t))dt$ consumed in the interval $[t, t+dt]$, and the flow-time $t \cdot s(t)dt$ of the infinitesimal part of the job processed in this interval. The sum of these quantities must be bounded in terms of the increase in the clairvoyant objective.

In order to understand how the clairvoyant schedule changes in response to an increase in weight of the job, say from W to $W + dW$, consider the "power curve" of $\mathsf{algo}^{(C)}$, i.e., the power consumed by the algorithm as a function of time. Since the algorithm sets instantaneous speed s such that power = remaining weight, this curve also gives the remaining weight $\overline{W}$ as a function of time. The remaining weight and speed satisfy $s = -d\overline{W}/dt$; therefore the power curve is defined by the differential equation $P(-d\overline{W}/dt) = \overline{W}$. The "shape" of the power curve is independent of the actual weight of the job in the current instance; the weight simply determines the point on this curve where we start from. This means that in response to an increase in weight of the job, the starting point shifts higher and the entire power curve shifts to the right (as illustrated in Fig. 1a). Let dt' denote the increase in the total processing time of the job by $\mathsf{algo}^{(C)}$ due to the increase in weight; this is how much the power curve shifts to the right by. It is now easy to see that dt' and dW are related by $dW = P^{-1}(W)dt'$. The corresponding increase in energy is $Wdt' = \frac{WdW}{P^{-1}(W)}$.

[6]Since we are in the single job case, we assume w.l.o.g. (without loss of generality) that the density of the job is 1.

We design the non-clairvoyant algorithm such that it matches the clairvoyant algorithm in terms of the increase in energy. In other words, we want $P(s(t))dt = \frac{WdW}{P^{-1}(W)}$, i.e.,

$$P^{-1}(W)P(s(t))dt = WdW = Ws(t)dt. \tag{3}$$

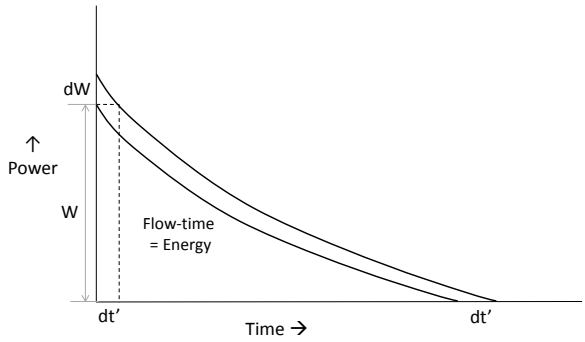
This is achieved by setting $P(s(t)) = W$, i.e., the instantaneous power in the non-clairvoyant algorithm equals the *processed weight* of the job. This ensures that the energy consumption of $\mathsf{algo}^{(NC)}$ and $\mathsf{algo}^{(C)}$ match exactly. It is interesting to note that the power curve of $\mathsf{algo}^{(NC)}$ is exactly identical to the power curve of $\mathsf{algo}^{(C)}$ in reverse (illustrated in Fig. 1b).

But what about the flow-times? The clairvoyant algorithm, by virtue of the $P = \overline{W}$ rule, has the property that its (fractional) flow-time and energy are exactly equal. However, this is not true for the non-clairvoyant algorithm. In fact, the flow-time of the $\mathsf{algo}^{(NC)}$ is given by the area *above* the power curve since remaining weight at any instant is the total weight minus the processed weight, the latter being equal to the instantaneous power. This is illustrated in Fig. 1b. Our crucial observation, which makes this analysis work for a single job, is that if the power function is $P = s^\alpha$, then the ratio of the two areas corresponding to flow-time and energy depends only on the value of α and is independent of the actual weight of the job.

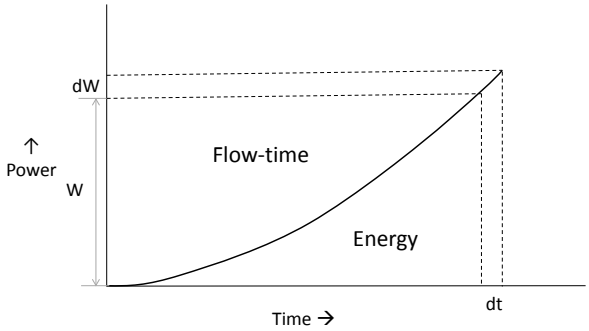
1.3 Related Work

In the last few years, there has been tremendous interest in the design of energy efficient algorithms. In the dynamic speed scaling approach, the problem of minimizing the sum of energy and flow-time was introduced by Albers and Fujiwara [2]. They also proposed what has been the thematic approach for this problem (in the clairvoyant setting): to run at a speed such that the power consumed is equal to the number of remaining jobs. This was later generalized to weighted flow-time. This approach has been analyzed in many papers (e.g., [5, 6, 7]) and the best result for this problem is by Bansal et. al. [8] who gave a $2 + \epsilon$ competitive algorithm for fractional weighted flow-time and $3 + \epsilon$ competitive algorithm for unweighted integral flow-time. For unweighted integral flow-time, Andrew et al. [9] improved the competitive ratio to 2. For weighted integral flow-time the combined results of [8] and [5] imply an $O\left(\frac{\alpha}{\log \alpha}\right)$ competitive ratio. Note that the problem we consider is harder since the volumes are not known in advance. We relate the performance of our algorithm to the algorithm of Bansal et al. [8] for fractional weighted flow-time. For scheduling on multiple machines, Gupta et al. [14] gave an $O(\alpha^2)$-competitive algorithm for the related machines case[7] which was extended to unrelated machines (with the same competitive ratio) by Anand et al. [12].

[7]and an $O(\alpha)$-competitive algorithm for the unweighted version



(a) The clairvoyant power curve



(b) The non-clairvoyant power curve

Figure 1: Analysis of a single job

The non-clairvoyant version where the weight is known at the time of release has also been considered [7, 11]. Lam *et al* [7] gave a $(2 - \frac{1}{\alpha})^2$-competitive algorithm for weighted flow-time plus energy when all jobs are released together, in the non-clairvoyant setting with known weights. Chan et al. [11] gave a $\frac{2\alpha^2}{\log \alpha}$-competitive algorithm for unweighted flow-time plus energy, once again for the non-clairvoyant setting with known (unit) weights. We consider the non-clairvoyant problem where the densities are known instead of the weights. The two problems are very different. In the case of known weights and unknown densities, the main challenge is to determine the order in which competing jobs are to be processed (recall that the clairvoyant algorithms suggest the HDF rule, but the algorithm does not have knowledge of job densities in advance), and once this is determined, the speed at which the jobs are processed essentially follows from the clairvoyant strategy of equating remaining job weights with instantaneous power. For example, the known weights model is identical to the clairvoyant setting in the uniform densities scenario. On the other hand, in our setting of known densitites and unknown weights, the order in which jobs are processed is given by the HDF rule, but determining the speed of the machine at any given time turns out to be the main challenge (note that the clairvoyant strategy of equating the remaining job weights with instantaneous power cannot be implemented since the job weights are not known in advance). Moreover our most general result is for the weighted case of (integral/fractional) flow-time plus energy. Many other variants of these problems have been considered, such as when the maximum speed is bounded [6], minimizing energy for jobs with given deadlines (offline and online) [15, 3], and minimizing flow-time under an energy budget [4]. The issues tackled in these papers are quite different from the ones in this paper. The reader is referred to a survey by Albers [16] of the different approaches to energy management and corresponding results.

Roadmap. We establish preliminary notation and terminology that we use throughout the paper in section 2. In section 3, we present the algorithm for the uniform case and give its detailed analysis for both fractional and integral flow-times. Our analysis for the case of non-uniform densities on a single machine is substantially more complicated. In section 4, we give a description of the algorithm and an overview of the analytical tools that we use, but defer a detailed analysis to the full version of the paper. We give a black box extension for non-uniform densities from fractional to integral flow-times in section 5. In section 6, we give the algorithm for scheduling on identical parallel machines without immediate dispatch and the lower bound with immediate dispatch. We discuss open problems in Section 7.

2. PRELIMINARIES

The (offline) problem of scheduling to minimize flow-time plus energy is as follows. There is a single machine that can run at any non-negative speed. Running the machine at a higher speed processes jobs faster but consumes higher energy, as given by a *power* function $P : \mathbb{R}_+ \to \mathbb{R}_+$ that is monotonically non-decreasing and convex. $P(0) = 0$. There are jobs that need certain amounts of processing power. The problem is to process all the jobs in a way that minimizes the sum of the total energy consumed and the total weighted *flow-time* of all the jobs. We think of the power function as something pre-defined and not as part of an *instance* of the problem.

Input: A set of jobs J. For each job $j \in J$, its release time $r[j]$, volume $V[j]$ and density $\rho[j]$. Let the *weight* of job j be $W[j] = \rho[j] \cdot V[j]$.

Output: For each time $t \in [0, \infty)$ the job to be scheduled at time t, denoted by $j(t)$, and the speed of the machine, denoted by $s(t)$. We write just s when the dependence on t is clear from the context.

Constraints: A job can only be scheduled after its release time. For each job j, the total computational power allocated to the job must be equal to its volume, that is,

$$\int_{t \in [r[j], \infty): j(t) = j} s(t)\, dt = V[j].$$

Objectives: The total energy consumed is the integral of the power function over time: $E = \int_{t=0}^{\infty} P(s(t))\, dt$. The integral flow-time of a given job j is

$$F_{int}[j] = W[j] \cdot (c[j] - r[j]),$$

where $c[j]$ denotes the completion time of j. The fractional flow-time of j is

$$\begin{aligned} F[j] &= \rho[j] \cdot \int_{t \in [r[j], \infty]: j(t) = j} (t - r[j]) s(t) dt \\ &= \rho[j] \cdot \int_{t=r[j]}^{\infty} \overline{V}[j](t)\, dt, \end{aligned}$$

where $\overline{V}[j](t)$ is the remaining volume of job j at time t, i.e.,

$$\overline{V}[j](t) = V[j] - \int_{t' \in [r[j], t]: j(t') = j} s(t')\, dt'.$$

The problem is to minimize the sum of the flow-times of all jobs plus energy, which is (the integral objective) $G_{int} = E + \sum_j F_{int}[j]$, or (the fractional objective) $G_{frac} = E + \sum_j F[j]$.

The *online clairvoyant* problem is when the details of job j (density, volume) are given at time $r[j]$. The algorithm makes its decisions at any time without knowing which jobs will be released in the future. In the *online non-clairvoyant* problem, only the density $\rho[j]$ is given upon the release of job j at time $r[j]$. The volume $V[j]$ is not given. At any future point of time, the algorithm only knows if $\overline{V}[j](t) > 0$ or not. If a job j is such that $\overline{V}[j](t) > 0$, we say the job is *active*, otherwise it is *inactive* (i.e., completed or not released yet).

We now describe a 2-competitive algorithm for the online clairvoyant problem. We call it **Algorithm** C. The job to be scheduled is determined by highest density first (HDF): schedule the active job with the highest density. The speed at time t is set based on the total remaining weight of active jobs at time t, denoted by $\overline{W}(t) = \sum_j \rho[j] \cdot \overline{V}[j](t)$. The speed at time t is such that $P(s(t)) = \overline{W}(t)$. For Algorithm C, the total energy always equals the total flow-time. This is because the total flow time is

$$\sum_j F[j] = \int_{t=0}^{\infty} \sum_j \rho[j] \cdot \overline{V}[j](t)\, dt = \int_{t=0}^{\infty} \overline{W}(t)\, dt.$$

THEOREM 1 ([8]). **Algorithm** C *is 2-competitive.*

We need the following properties of Algorithm C that follow from elementary calculus (recall that α is the parameter that relates the instantaneous power P to speed s through $P = s^\alpha$). Note that this lemma can be invoked at any intermediate stage of processing a job since Algorithm C is memoryless.

LEMMA 2. *Consider a run of* **Algorithm** C *on a single job of weight W and density ρ, which takes time t to complete. Then they satisfy the following relations:*

1. $\frac{dW}{dt} = \rho W^{\frac{1}{\alpha}}$,
2. $\rho(1 - \frac{1}{\alpha})t = W^{1-\frac{1}{\alpha}}$, *and*
3. $\frac{W}{t} = (1 - \frac{1}{\alpha})\frac{dW}{dt}$.

3. UNIFORM DENSITY

In this section we consider the uniform density case, i.e., $\rho[j] = 1$ for all j. First, we give an algorithm for the online non-clairvoyant version of the problem for minimizing fractional flow-time plus energy on a single machine, which we call **Algorithm** NC. Assume w.l.o.g that the release times are all distinct. The job to be scheduled is determined according to the first-in first-out (FIFO) rule: schedule active job j, if one exists, with the smallest $r[j]$. The speed is set by considering a run of algorithm C on the same instance. Notice that by the time Algorithm NC schedules a job j, it knows the volumes/weights of all the jobs that are released earlier than $r[j]$. Thus one can simulate Algorithm C upto time $r[j]$. Let $\overline{W}^{(\mathrm{C})}(r[j]^-) = \lim_{t \to r[j]-} \overline{W}^{(\mathrm{C})}(t)$ be the remaining weight of the active jobs in algorithm C at time $r[j]$ (not including the weight of job j). Further, let $\breve{W}[j](t)$ be the weight of job j processed by Algorithm NC till time t. At time t, Algorithm NC sets a speed s such that $P(s) = \overline{W}^{(\mathrm{C})}(r[j]^-) + \breve{W}[j](t)$.

We now restrict our attention to power functions of the form $P(s) = s^\alpha$ for some $\alpha > 1$. We obtain a competitive ratio for Algorithm NC by showing that it is almost as good as Algorithm C, which is surprising since the former is in the non-clairvoyant setting. In particular we show that the energy consumed in the two algorithms is the same (Lemma 3) and that the flow-times are within a factor of $1 - \frac{1}{\alpha}$ (Lemma 4).

LEMMA 3. *Algorithms* C *and* NC *consume the same amount of energy.*

LEMMA 4. *For all $\alpha > 1$ and power functions $P(s) = s^\alpha$, the total flow-time of algorithm* NC *= the total flow-time of Algorithm* C *$/(1 - \frac{1}{\alpha})$.*

The competitive ratio of Algorithm NC follows immediately from Lemmas 3 and 4 since we noted earlier that Algorithm C is 2-competitive (Theorem 1) and that the flow-time equals the total energy for Algorithm C. Therefore we get the following theorem.

THEOREM 5. *For all $\alpha > 1$ and power functions $P(s) = s^\alpha$, Algorithm* NC *is $1 + 1/(1 - \frac{1}{\alpha}) = 2 + \frac{1}{\alpha - 1}$-competitive for the objective of fractional flow-time plus energy.*

In the remainder of this section we give the proofs of Lemma 3. and Lemma 4, and the extension to the integral objective.

3.1 Energy Comparison (Proof of Lemma 3)

We show a very close structural similarity between the two algorithms, by showing that their "speed profiles" are essentially the same, upto a re-mapping of time (Lemma 6). This almost immediately implies Lemma 3. In fact, Lemmas 6 and 3 are actually true for all power functions, not just ones of the form s^α. The form of the power function is needed for flow-time comparison (Lemma 4).

LEMMA 6. *There exists a measure preserving $1 - 1$ mapping $t \to t'$ from $\mathbb{R}_+$ to itself such that for all times t, the speed in Algorithm* NC *at time t is equal to the speed in Algorithm* C *at time t'.*

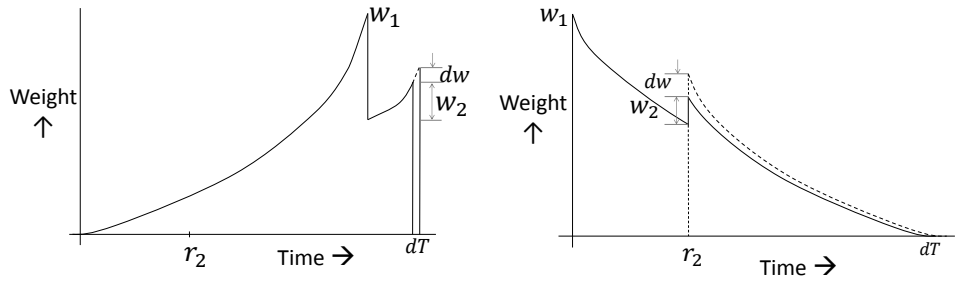
This lemma is proved by induction on time, with a stronger inductive hypothesis, which is stated in Lemma 7. First, we define the notion of an instantaneous instance. The stronger inductive hypothesis constructs a mapping between the two algorithms for every such instance and shows how to modify this mapping over time. The instance at time ∞ is just the original instance and hence we obtain a mapping for it as well.

For any time T, let the *clairvoyant instance at time T*, denoted by $I(T)$, be an instance where the job release times are as in the original instance, with weight of job j being $\breve{W}[j](t)$, which is the weight of job j processed by Algorithm NC till time T. This would be the instance if for algorithm NC, the instance ended at time T. (Clearly $I(T)$ depends on Algorithm NC).

For any two times $T_1 < T_2$, the run of Algorithm NC for instance $I(T_1)$ is a prefix of its run for instance $I(T_2)$. However, the run of Algorithm C is different for each instance. The proofs use how the run of Algorithm C evolves with these changing instances. We need to consider various quantities of Algorithm C that correspond to different instances. Therefore, we will denote a quantity w.r.t a particular instance by representing the time of that instance in the subscript. For example, the remaining weight at time t in instance $I(T)$ is denoted $\overline{W}_T(t)$.

We denote the speed of Algorithm C by $s^{(\mathrm{C})}$ and that of Algorithm NC by $s^{(\mathrm{NC})}$.

LEMMA 7. *$\forall\, T \in \mathbb{R}_+$, there exists a measure preserving $1 - 1$ mapping $t \to t'$ from $[0, T]$ to itself such that for all $t \in [0, T]$, the speed of algorithm* NC *at time t, which is $s^{(\mathrm{NC})}(t)$, equals the speed of algorithm* C *at time t' on instance $I(T)$, which is $s_T^{(\mathrm{C})}(t')$.*



(a) The change in the non-clairvoyant algorithm upon processing an extra dw weight of job 2 which takes an extra time of dT. Job 2 is released at r_2 and has weight w_2 currently. Job 1 is released at time 0 and has weight w_1, all of which has been processed.

(b) The change in the run of the clairvoyant algorithm due to an extra dw weight of job 2. Here the speed of the algorithm changes all the way from time r_2 to the end. The extra time taken dT is however the same as in the non-clairvoyant case.

Figure 2: Analysis for the uniform density case

PROOF. The proof is by induction on T (refer to Figure 2 for a pictorial depiction). The statement is vacuously true at time $T = 0$. Suppose the statement is true for T. We will argue that it is also true for $T + dT$, for an infinitesimal dT.

Clearly, as one goes from T to $T + dT$, the speed of algorithm NC only changes in the interval $[T, T + dT]$. Suppose Algorithm NC schedules job j during time $[T, T + dT]$. The speed $s^{(\text{NC})}(T)$ is such that $P(s^{(\text{NC})}(T)) = \overline{W}^{(\text{C})}(r[j]^-) + \breve{W}[j](T)$. Since dT is infinitesimal, we may assume that the speed does not change during this interval. The increase in the completed weight of job j during this interval is

$$dW = \breve{W}[j](T + dT) - \breve{W}[j](T) = s^{(\text{NC})}(T)dT.$$

The change in Algorithm C due to the change in the instance from $I(T)$ to $I(T + dT)$ is more complicated. This change does not affect Algorithm C before time $r[j]$. At $r[j]$, the remaining weight increases by dW, i.e.,

$$\overline{W}^{(\text{C})}_{T+dT}(r[j]) = \overline{W}^{(\text{C})}_{T}(r[j]) + dW,$$

which increases the speed of Algorithm C at time $r[j]$. Then as we move forward in time from $r[j]$ the remaining weight decreases.

$$\begin{aligned}\overline{W}^{(\text{C})}_{T+dT}(r[j] + dT) &= \overline{W}^{(\text{C})}_{T+dT}(r[j]) - s^{(\text{C})}(r[j])dT \\ &= \overline{W}^{(\text{C})}_{T}(r[j]) + (s^{(\text{NC})}(T) - s^{(\text{C})}(r[j]))dT.\end{aligned}$$

In fact, the drop in remaining weight after time dT is the same as the initial increase since $s^{(\text{NC})}(T) = s^{(\text{C})}(r[j])$, which follows from

$$\begin{aligned}\overline{W}^{(\text{C})}(r[j]) &= \overline{W}^{(\text{C})}(r[j]^-) + \breve{W}[j](T), \text{ and} \\ P(s^{(\text{C})}(r[j])) &= \overline{W}^{(\text{C})}(r[j]).\end{aligned}$$

Therefore, the remaining weight at time $r[j] + dT$ in the instance $I(T + dT)$ is the same as the remaining weight at time $r[j]$ in the instance $I(T)$. This continues for all $t \geq r[j]$, i.e.,

$$\overline{W}^{(\text{C})}_{T+dT}(t + dT) = \overline{W}^{(\text{C})}_{T}(t).$$

We now specify the measure preserving $1 - 1$ mapping for $T + dT$. We map $T + dT \to r[j]$. Recall that $s^{(\text{NC})}(T + dT) = s^{(\text{C})}(r[j])$. For $t \leq T$, we map $t \to t' + dT$, where $t \to t'$ in $I(T)$. The speed $s^{(\text{NC})}$ does not change for $t \leq T$, due to the change in T. The speed $s^{(\text{C})}$ does not change for $t < r[j]$. For $t \geq r[j]$, the above argument shows that $s^{(\text{C})}_{T+dT}(t' + dT) = s^{(\text{C})}_{T}(t') = s^{(\text{NC})}(t)$. □

3.2 Flow-time Comparison (Proof of Lemma 4)

We prove Lemma 4 by once again considering the evolution of Algorithm C with $I(T)$. We show that the rate of change of flow-time of Algorithm C on instance $I(T)$, w.r.t. T, is $1 - \frac{1}{\alpha}$ times the rate of change of flow-time of Algorithm NC w.r.t. T. The rest of this section assumes $P(s) = s^\alpha$ for some $\alpha > 1$.

As in the proof of Lemma 7, when we go from T to $T + dT$, Algorithm C processes an extra dW amount of weight at time $r[j]$ at speed $s^{(\text{C})}(r[j])$, which takes time dT. This incurs an extra $P(s^{(\text{C})}(r[j]))\, dT$ units of energy consumption. Therefore, the rate of change of flow-time/energy[8] of Algorithm C on instance $I(T)$ w.r.t. T is

$$\frac{dE^{(\text{C})}(I(T))}{dT} = P(s^{(\text{C})}(r[j])) = \overline{W}^{(\text{C})}(r[j]^-) + \breve{W}[j]. \quad (4)$$

At the same time, Algorithm NC incurs an extra flow-time due to the extra dW weight which is processed in the interval $[T, T+dT]$. This weight dW waits for time $T - r[j]$, giving an extra flow-time of $(T - r[j])dW$. Therefore the rate of change of flow-time of Algorithm NC w.r.t. time T is

$$\frac{dF^{(\text{NC})}}{dT} = (T - r[j])\frac{d\breve{W}[j]}{dT}. \quad (5)$$

Let $W' = \overline{W}^{(\text{C})}(r[j]^-) + \breve{W}[j]$ and $T' = T - r[j]$. W' is the total weight processed by Algorithm C in time T' and hence

$$\begin{aligned}W' &= (1 - \tfrac{1}{\alpha})T'\frac{dW'}{dT'}, \quad \text{by Lemma 2} \\ \Rightarrow \frac{dE^{(\text{C})}(I(T))}{dT} &= (1 - \tfrac{1}{\alpha})\frac{dF^{(\text{NC})}}{dT}, \quad \text{by Eqns. 4 and 5.}\end{aligned}$$

[8]Recall that the total flow-time and energy are equal for Algorithm C.

3.3 Integral Objective

We will now show that the integral flow-time of any schedule produced by Algorithm NC can be bounded in terms of its fractional flow-time.

LEMMA 8. *The integral flow-time of any schedule produced by Algorithm* NC *is at most* $2 - \frac{1}{\alpha-1}$ *times its fractional flow-time.*

PROOF. When we go from T to $T + dT$, Algorithm NC incurs an extra fractional flow-time of $(T - r[j])dW$, i.e.,

$$\frac{dF^{(\text{NC})}}{dT} = (T - r[j])dW. \tag{6}$$

At the same time, Algorithm NC incurs an extra integral flow-time both due to the extra dW weight which is processed between T and $T + dT$ and due to the processed weight $\breve{W}[j]$ of job j that now contributes for the extra duration dt. The first part corresponds exactly to the increase in fractional flow-time, i.e., $(T-r[j])dW$. So, we bound the second part, i.e., $\breve{W}[j]dt$. Let $W' = \overline{W}^{(\text{C})}(r[j]^-) + \breve{W}[j]$ and $T' = T - r[j]$. W' is the total weight processed by Algorithm C in time T', and hence by Lemma 2,

$$\begin{aligned} W' &= (1 - \tfrac{1}{\alpha})T'\frac{dW'}{dT'} \\ \Rightarrow \breve{W}[j]dt &\leq (1 - \tfrac{1}{\alpha})(T - r[j])dW \\ \Rightarrow \breve{W}[j]dt &\leq (1 - \tfrac{1}{\alpha})\frac{dF^{(\text{NC})}}{dT}, \end{aligned}$$

where the first step follows from $W' \geq \breve{W}[j]$ and the second step follows from Eqn. 6. Therefore, $\frac{dF^{(\text{NC})}_{int}}{dT} \leq (1+(1 - \frac{1}{\alpha}))\frac{dF^{(\text{NC})}}{dT}$. □

The competitive ratio of Algorithm NC follows immediately from Lemmas 3, 8 and 4 since we noted earlier that Algorithm C is 2-competitive (Theorem 1) and that the total flow-time is equal to the total energy for Algorithm C. Therefore we get the following theorem.

THEOREM 9. *For all* $\alpha > 1$ *and power functions* $P(s) = s^\alpha$, *Algorithm* NC *is* $2 + 1/(1 - \frac{1}{\alpha}) = 3 + \frac{1}{\alpha-1}$*-competitive for the objective of integral flow-time plus energy.*

4. NON-UNIFORM DENSITY

In this section, we will significantly generalize the results in the previous section, and give a non-clairvoyant algorithm for jobs of non-uniform density. First, we consider the fractional objective in this section, and then extend it to the integral objective in the next section.

To define an algorithm for our problem, we need to specify, for every time t, the job selected for processing at time t, and the speed at which the selected job is processed. Recall that in the non-clairvoyant version, the algorithm only has the following information at its disposal: the densities of all the jobs that have been released till time t, the volume/weight of all jobs that have been completed till time t, the set of active jobs, and a lower bound on the volume/weight of every active job given by the volume/weight of the job processed by the non-clairvoyant algorithm till time t. As in the case of uniform densities, the non-clairvoyant algorithm is closely related to the clairvoyant algorithm (algorithm C) for the current instance $I(t)$. Recall that algorithm C uses the HDF rule to determine the processing order among jobs of different densities that are waiting at any given time. If there are multiple jobs of the highest density, then the algorithm is agnostic to which of these jobs is chosen, but for the purpose of our analysis, it will be convenient to assume that algorithm C uses the FIFO rule, i.e. it selects the job with the highest density that was released the earliest.

As mentioned earlier, a key step in algorithm NC is to round all densities down to powers of some constant β. Similar to algorithm C, algorithm NC also processes the job with the highest density among the active jobs at any given time, and uses the FIFO rule to decide the processing order of jobs of the same density. (Note that, in effect, jobs in the same density bracket are processed in FIFO rather than HDF order since their densities are rounded to the same value.) The speed of algorithm NC at time t is η times the speed of algorithm C for the instance $I(t)$ (we call $I(t)$ the *current instance*), i.e. $s^{(\text{NC})}(t) = \eta \cdot s^{(\text{C})}_t(t)$, where η is a constant that we will determine later. (Again, the rounding of densities affects the speed of algorithm NC via algorithm C since $I(t)$ is now defined to be the rounded instance at time t.)

Our analysis depends crucially on the fact that the current instance will eventually evolve to the real problem instance. But this need not be the case if algorithm NC always runs at zero speed. Indeed, initially all jobs in the current instance have zero weight; so the speed given by the above definition will be zero. We fix this issue by setting the speed of algorithm NC to be ϵ more than that given by the above definition, for some arbitrarily small but fixed ϵ. We will ignore this excess speed in the analysis.

In analyzing algorithm NC, we will compare its energy and flow-time to the energy and flow-time (which are equal) of algorithm C respectively. The energy comparison is relatively straightforward and is deferred to the full version of the paper. Intuitively, Algorithm NC uses η^α times the energy of Algorithm C. In terms of flow-times, we will show the following lemma.

LEMMA 10. *For any instance* $I(t)$*, the total flow-time in algorithm* NC *is at most a constant times that in algorithm* C*, where the constant depends only on* α*.*

4.1 Flow-time Comparison (Sketch of Proof of Lemma 10)

As in the case of uniform densities, we will establish Lemma 10 by induction over the evolving instances. The main objective of using the multiplicative factor of η in setting the speed of algorithm NC is to ensure that for every active job j, a constant fraction of j is waiting at time t in algorithm Cin the current instance. Recall that in contrast, the entire weight of job j in the current instance has already been processed by algorithm NC before time t.

LEMMA 11 (PROPERTY (A)). *For any active job* j*,*

$$\overline{W}^{(\text{C})}_t(t)[j] \geq \zeta \cdot W_t[j] \text{ for some constant } \zeta.$$

Another consequence of the higher speed of algorithm NC compared to algorithm C is that the total volume of jobs processed by algorithm NC dominates that processed by algorithm C for any time interval ending at the current time t.

LEMMA 12 (PROPERTY (B)). *For any time* $t_1 < t$*,*

$$V^{(\text{NC})}(t_1, t) \geq \gamma \cdot V^{(\text{C})}_t(t_1, t) \text{ for some constant } \gamma.$$

We prove the above two properties jointly using induction over the evolving instances. Once these properties have been established, we use them to show that for any active job j, the completion time of j in algorithm C is significantly greater than its completion time in algorithm NC if the job get completed right now. Let $c^{(C)}_t[j]$ denote the completion time of job j in Algorithm C in instance $I(t)$.

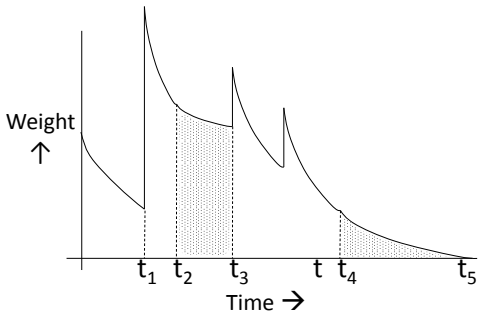


Figure 3: The structure of preemption intervals in algorithm C for the current instance. Job j^* is released at time t_1, i.e., $r_{j^*} = t_1$ and is processed in the dotted intervals. There are two preemption intervals $[t_1, t_2]$ and $[t_3, t_4]$, i.e., $\widehat{R}_1 = t_1$ and $\widehat{R}_2 = t_3$. Therefore, $i^* = 2$. The last preemption interval completes after the current time t.

LEMMA 13. *For any active job j,*

$$c_t^{(C)}[j] - t \geq \psi \cdot (t - r[j]) \text{ for some constant } \psi.$$

The proofs of Lemma 11, Lemma 12, and Lemma 13 are deferred to the full version of the paper.

We now explain the difficulties of proving Lemma 10 via Lemma 13, and sketch the key ideas. Note that when we transform the instance from $I(t)$ to $I(t+dt)$ by adding a weight of $dW = dW_t[j^*]$ to job j^*, the increase in flow-time of algorithm NC is

$$dF^{(\text{NC})} = (t - r[j^*])\, dW \leq \frac{1}{\psi} \cdot \Big(c_t^{(C)}[j] - t\Big)\, dW, \quad \text{by Lemma 13.} \tag{7}$$

We can show that if the increase in the remaining weight at time t for algorithm C, i.e., $d\overline{W}_t^{(\text{C})}(t)$, is at least a constant factor of dW, then Eqn. (7) is sufficient to prove Lemma 10. However, while this property holds in the case $j_t^{(\text{C})}(t) = j^*$, it may not hold in general. This necessitates a more complicated proof, the first step of which is to show a somewhat weaker property that we describe below.

To describe this property, we first need to describe and establish some notation about the structure of the time interval $[r[j^*], t]$ in algorithm C for the current instance. This time interval alternates between intervals where j^* is being processed by algorithm C and intervals where jobs of densities higher than $\rho[j^*]$ are being processed (see Figure 3 for an example). We call the latter *preemption intervals*, and the jobs that are processed by algorithm C in the preemption intervals are called *preempting jobs*. We index the preemption intervals in chronological order. Let $\widehat{R}_i$ and $\widehat{V}_i$ denote the starting time and the volume of all preempting jobs in the i-th preemption interval respectively. Also, let $\overline{W}_i = \overline{W}_t^{(\text{C})}(\widehat{R}_i^-)$. (Note that $\overline{W}_i$ does not include the weight of jobs released at time $\widehat{R}_i$.) Let i^* be the index of the last preemption interval.

The next lemma states that the increment in the remaining weight at time $\widehat{R}_{i^*}$ is a constant fraction of the added weight dW, if 1) the ratio of remaining weight at $\widehat{R}_{i^*}$ is a constant fraction of that at $\widehat{R}_1$, and 2) the total preempting volume except the last preemption interval is a constant fraction of the volume of the current job. Note that properties (A) and (B) informally correspond to these two conditions, though a formal proof will need to use induction over the properties since the following lemma is used to prove the properties themselves. Also, note that the property established by the lemma is weaker than what we would have ideally liked: the increment in the remaining weight at time t (instead of time $\widehat{R}_{i^*}$ as given by the lemma) is at least a constant factor of dW. (The proof of this lemma is deferred to the full version of the paper.)

LEMMA 14. *If $\overline{W}_{i^*} \geq \zeta \cdot W_t[j^*]$, then*

$$\frac{d\overline{W}_t^{(\text{C})}(\widehat{R}_{i^*})[j^*]}{dW} \geq \left(1 - \frac{1}{4\zeta} \frac{\sum_{k=1}^{i^*-1} \widehat{V}_k}{V_t[j^*]}\right) \left(\frac{\overline{W}_{i^*}}{\overline{W}_1}\right)^{\frac{1}{\alpha}}.$$

Recall that the increase in the flow-time of Algorithm C equals the integral of $d\overline{W}_t^{(\text{C})}(t')$ as over time t' starting from $r[j^*]$. Up to the last preemption interval, $d\overline{W}_t^{(\text{C})}(t')$ is at least some constant factor of $dW_t[j^*]$ by Lemma 14. Therefore if the last preemption interval is short enough, i.e., if it is at most a constant factor of $(t - r[j^*])$, then we will be done. However, in general, the last preemption interval could be long. Therefore a local charging argument will not work and we resort to an amortized analysis.

The intuition for why an amortized analysis works is as follows. If a preempting job in the last preemption interval is small, then we can ignore it since the time taken by this job is small compared to the total time. On the other hand if the preempting job is large, then we can charge some of $dF^{(\text{NC})}(t)$ to $dF^{(\text{C})}(t)$ at *an earlier time when we were processing this large preempting job*. The amortized analysis uses a potential function to accomplish this. When we have an extra amount in the $dF^{(\text{C})}(t)$ in the charging argument, we will store this in the potential function. Later we may need help in the charging argument in which case we will draw from the stored potential function to pay for $dF^{(\text{NC})}(t)$.

More precisely, we will have one bin for each pair of density levels $k > k'$, into which we will store the extra clairvoyant flow-time when we are processing jobs with density β^k and from which we will draw when we are processing jobs with density $\beta^{k'}$. In particular, consider a bin (k, k'). Suppose $j^* = j$ with density β^k. We will show that $dF^{(\text{C})}(t)$ is at least a constant fraction of the change in processing time of the current job times its weight. We will store $2^{k'-k}$ fraction of $dF^{(\text{C})}(t)$ in the bin for (k, k'). By doing so, the total amount that we store in all bins is a constant factor of $dF^{(\text{C})}(t)$. Further, when Algorithm NC finishes processing job j, the total potential stored in bin (k, k') due to j is at least the processing time of j times a $2^{k'-k}$ fraction of its weight. Later, when Algorithm NC is processing dW weight of a job j' with density $\beta^{k'}$, we can withdraw potential equal to the processing time of job j times dW from the bin (k, k'). We can keep doing this until the weight of the job j' become at least a $2^{k'-k}$ fraction of the weight of job j. Now recall that the density of the jobs are rounded to be powers of β. Choosing $\beta > 4$, we get that if the weight of the job j' becomes at least a $2^{k'-k}$ fraction of the weight of job j, then the volume of job j' is at least $(2/\beta)^{k'-k} = (\beta/2)^{k-k'} > 2^{k-k'}$ times the volume of j. So the processing time of job j is now negligible compared to that of j'. The details of the amortized analysis, which ultimately yields Lemma 10, is deferred to the full version of the paper.

5. INTEGRAL OBJECTIVE

In this section, we will give a black box reduction of any schedule optimizing fractional flow-time plus energy to one optimizing integral flow-time plus energy. Our goal is to show the following lemma.

LEMMA 15. *If a non-clairvoyant algorithm $\mathcal{A}_{frac}$ has the guarantee that the fractional flow-time plus energy of any schedule produced by the algorithm is at most Γ_{frac} times the optimum, where*

Γ_{frac} is a constant that depends only on α, then there is a non-clairvoyant algorithm $\mathcal{A}_{int}$ with the guarantee that the integral flow-time plus energy of any schedule produced by the algorithm is at most Γ_{int} times the optimum, where Γ_{int} is some constant that also depends only on α.

PROOF. Let $j(t)$ be the job processed by algorithm $\mathcal{A}_{frac}$ at time t, and let $s(t)$ be the corresponding speed of the machine. Then, algorithm $\mathcal{A}_{int}$ is defined as follows: if job $j(t)$ is active at time t (i.e., it has not been completed yet), then algorithm $\mathcal{A}_{int}$ processed job $j(t)$ at speed $(1+\epsilon)s(t)$ for some constant $\epsilon > 0$; otherwise, if job $j(t)$ has already been completed by algorithm $\mathcal{A}_{int}$ before time t, then it keeps the machine idle at time t. Note that for every job j and every time t, the weight of job j processed by algorithm $\mathcal{A}_{int}$ till time t is exactly $1+\epsilon$ times the weight of job j processed by algorithm $\mathcal{A}_{frac}$ till time t. In other words, when $\mathcal{A}_{int}$ finishes job j, $\frac{1}{1+\epsilon}$ fraction of the job is left in algorithm $\mathcal{A}_{frac}$. Let $T[j]$ be the difference between the completion time of job j in algorithm $\mathcal{A}_{int}$ and its release time $r[j]$. Then, the integral flow-time due to job j in algorithm $\mathcal{A}_{int}$ is exactly $W[j]\cdot T$, whereas the fractional flow-time of job j in algorithm $\mathcal{A}_{frac}$ is at least

$$(1-\frac{1}{1+\epsilon})W[j]\cdot T = \frac{\epsilon}{1+\epsilon}\cdot W[j]\cdot T.$$

Therefore, the total integral flow-time in algorithm $\mathcal{A}_{int}$ is at most $1+\frac{1}{\epsilon}$ times the total fractional flow-time in algorithm $\mathcal{A}_{frac}$. On the other hand, the total energy consumed in algorithm $\mathcal{A}_{int}$ is at most $(1+\epsilon)^\alpha$ times that in algorithm $\mathcal{A}_{frac}$. To complete the proof, we set $\Gamma_{int} = \max\left\{(1+\epsilon)^\alpha, 1+\frac{1}{\epsilon}\right\}\Gamma_{frac}$. □

The next theorem follows from the above lemma and our result for minimizing fractional flow-time plus energy.

THEOREM 16. *There is a deterministic algorithm for the non-clairvoyant scheduling problem with non-uniform density that has a constant competitive ratio, where the constant only depends on α, for the objective of integral flowtime plus energy.*

6. IDENTICAL PARALLEL MACHINES

Let there be a set of k identical machines. The goal is to minimize the sum of the energy and (fractional/integral) flow-time on all the machines. In the immediate dispatch model, on the release of a job, the algorithm must specify which machine it will be processed on. The actual processing might be done later but the selection of the machine has to be immediate. We show a lower bound of $\Omega(k^\beta)$ where $\beta = 1-1/\alpha$ on deterministic algorithms in the immediate dispatch model, even for uniform densities and fractional flow-time. The main property we use is that the algorithm has no way of distinguishing jobs of different volumes. Due to this, the adversary can pick the job volumes in such a way that the algorithm cannot do any load balancing. Release k^2 jobs at time 0, of which k jobs will have high volumes and the rest will have low volumes. The algorithm dispatches all the jobs right away and there exists a machine which has been assigned at least k jobs. The adversary chooses these k jobs to be the ones with the high volumes. The optimum schedule is the one that assigns all the high volume jobs to different machines. The cost of the optimum is (dominated by) the cost of processing a single high volume job whereas the cost of the algorithm is (dominated by) the cost of processing k high volume jobs. The ratio of these two costs gives a lower bound of $\Omega(k^\beta)$.

Now, we focus on the model where jobs do not need to be assigned immediately on arrival. However, once a job has started processing on a machine, it cannot be moved to another machine. In this model, we give an algorithm with constant competitive ratio. Our algorithm (we call it NC-PAR) is defined as follows. It maintains a global queue of unassigned jobs $\mathcal{Q}$ in FIFO order. Whenever a machine is available, i.e., all jobs previously assigned to the machine have been completed, we assign the first job in NC-PAR to the available machine. The instantaneous speed on a machine is determined exactly as in Algorithm NC, where the current instance is defined by the processed volume of all jobs assigned by Algorithm NC-PAR to this machine.

THEOREM 17. *For all $\alpha > 1$ and power functions $P(s) = s^\alpha$, Algorithm* NC-PAR *is $O(\alpha + \frac{1}{\alpha-1})$-competitive for the objective of integral/fractional flow-time plus energy on identical parallel machines.*

Similar to the single machine case, the analysis of Algorithm NC-PAR relies on comparing its flow-time and energy consumption with those of a known competitive (immediate dispatch) clairvoyant algorithm that we call C-PAR. Algorithm C-PAR behaves identical to Algorithm C on each individual machine, i.e., the instantaneous speed is set such that power equals fractional remaining weight. In assigning jobs to machines, Algorithm C-PAR embraces a greedy policy and assigns each arriving job immediately to a machine that minimizes the increase in the fractional objective.

THEOREM 18 ([12]). *Algorithm* C-PAR *is $O(\alpha)$-competitive for the objective of fractional flow-time plus energy.*

Next, we will show that the job assignment of algorithm NC-PAR is identical to that of algorithm C-PAR. This suffices since on an individual machine, Algorithms NC-PAR and C-PAR are identical to Algorithms NC and C respectively. The next lemma characterizes the job assignment rule of Algorithm C-PAR.

LEMMA 19. *For every job j, Algorithm* C-PAR *assigns j to a machine i that has the least remaining (fractional) weight when j is released.*

PROOF. Suppose the fractional remaining weight of jobs on some machine i is W when job j is released, i.e., at time $r[j]$. Then, using Lemma 2, the total energy for processing the remaining jobs is given by

$$\int_0^W \frac{\alpha}{\alpha-1} w^{1-\frac{1}{\alpha}} dw = \frac{\alpha}{\alpha-1}\cdot\frac{\alpha}{2\alpha-1} W^{2-\frac{1}{\alpha}}.$$

So, assigning job j to machine i would increase the energy by

$$\frac{\alpha}{\alpha-1}\cdot\frac{\alpha}{2\alpha-1}\left((W+W[j])^{2-\frac{1}{\alpha}} - W^{2-\frac{1}{\alpha}}\right),$$

which is monotonically increasing in W by the convexity of $x^{2-\frac{1}{\alpha}}$. To conclude the proof, we note that since the fractional flow-time equals energy for Algorithm C-PAR, it assigns job j to the machine that minimizes the increase in the energy consumption after time $r[j]$. □

Next, for convenience of presentation, we will assume that there is an arbitrary total order of all machines and both algorithms C-PAR and NC-PAR break ties according to this total order. This assumption can be removed with more careful analysis using the same idea. We omit the details for the sake of brevity.

LEMMA 20. *Algorithm* NC-PAR *assigns job j to machine i if and only if algorithm* C-PAR *assigns job j to machine i.*

PROOF. We will use induction on the order of arrival of jobs. All machines have zero remaining weight when the first job arrives. So the first job will be assigned to the first machine with respect to the total order in both algorithms C-PAR and NC-PAR.

Now, suppose that the assignments of all jobs before job j are identical in the two algorithm. We will show that this is also the case for job j. By Lemma 19, Algorithm C-PAR assigns job j to the machine i that has the least remaining weight at time $r[j]$. Further, by property 2 of Lemma 2, the completion time of all jobs on an individual machine is a monotonically increasing function of its fractional remaining weight at time $r[j]$. Finally, given that the job assignment has been the same in algorithms C-PAR and NC-PAR so far, it follows from Lemma 6 that for any individual machine, the completion time for all jobs released before job j on the machine is identical for algorithms C-PAR and NC-PAR. It follows that Algorithm C-PAR assigns job j to the machine with the earliest completion time in Algorithm NC-PAR for all jobs released before job j (breaking ties according to the total order of the machines). By definition of Algorithm NC-PAR, this is the same machine that Algorithm NC-PAR would assign job j to. □

Combining Lemma 20 with Lemma 3 and 4 respectively, we relate the energy and fractional flow-time of C-PAR with those of NC-PAR.

LEMMA 21. *Both algorithms* C-PAR *and* NC-PAR *consume the same amount of energy.*

LEMMA 22. *The fractional flow-time of Algorithm* NC-PAR *equals the fractional flow-time of Algorithm* C-PAR *divided by* $(1 - \frac{1}{\alpha})$.

Lemmas 21 and 22, and Theorem 18 immediately yield Theorem 17 for the fractional objective. Extending our proof to the integral objective is almost identical to the the analysis in Section 3.3; we omit the details for brevity.

7. OPEN PROBLEMS

The most natural open problem is to extend our results to the case of non-uniform density and identical parallel machines.

In terms of getting a positive result, it is no longer feasible for the non-clairvoyant algorithm to mimic the job assignment rule of the clairvoyant algorithm. When the clairvoyant algorithm dispatches a higher density job, it takes into account the lower density jobs to compute the increase in the cost for each machine. The non-clairvoyant algorithm, however, has to dispatch the higher density job without exploring the lower density jobs thoroughly. A natural policy for the non-calirvoyant algorithm is to follow HDF (probably with rounded densities) and dispatch only as needed to follow this rule. A candidate clairvoyant algorithm to compare this with is the one that considers only jobs of equal or higher density to calculate the increase in the cost. However the job assignments could still be different: for instance, jobs released later could affect the machine a job is assigned to in the non-clairvoyant algorithm whereas they do not in the clairvoyant algorithm.

In terms of showing a hardness result, one natural attempt is to use different densities to force the algorithm to do immediate dispatch and then apply the hardness result for immediate dispatch in Section 6. However, the following somewhat surprising fact rules this approach out. Suppose there are l jobs with densities $1, \rho, \rho^2, \ldots, \rho^{l-1}$ such that the cost of processing any one of them by itself on a single machine is c. Then the cost of processing all of them on l machines is lc. However, the cost of processing them all on a single machine is at most $4lc$ as long as $\rho \geq 4$. Thus, not load balancing jobs of different densities costs only a constant factor unlike the case of uniform densities where it costs a super-constant factor.

These issues indicate that this is an interesting open problem that needs some new ideas.

Acknowledgements

Part of this work was done when Yossi Azar, Zhiyi Huang, and Debmalya Panigrahi were visiting Microsoft Research, Redmond, WA. Yossi Azar is supported in part by the Israel Science Foundation (grant No. 1404/10) and by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11). Debmalya Panigrahi is suppported in part by a Duke University startup grant and a Google Faculty Research Award.

8. REFERENCES

[1] Barroso, L.A.: The price of performance. ACM Queue **3**(7) (2005) 48–53

[2] Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. ACM Transactions on Algorithms **3**(4) (2007)

[3] Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced cpu energy. In: FOCS. (1995) 374–382

[4] Pruhs, K., Uthaisombut, P., Woeginger, G.J.: Getting the best response for your erg. ACM Transactions on Algorithms **4**(3) (2008)

[5] Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. SIAM J. Comput. **39**(4) (2009) 1294–1308

[6] Bansal, N., Chan, H.L., Lam, T.W., Lee, L.K.: Scheduling for speed bounded processors. In: ICALP (1). (2008) 409–420

[7] Lam, T.W., Lee, L.K., To, I.K.K., Wong, P.W.H.: Speed scaling functions for flow time scheduling based on active job count. In: ESA. (2008) 647–659

[8] Bansal, N., Chan, H.L., Pruhs, K.: Speed scaling with an arbitrary power function. In: SODA. (2009) 693–701

[9] Andrew, L.L.H., Wierman, A., Tang, A.: Optimal speed scaling under arbitrary power functions. SIGMETRICS Performance Evaluation Review **37**(2) (2009) 39–41

[10] Motwani, R., Phillips, S., Torng, E.: Nonclairvoyant scheduling. Theoretical computer science **130**(1) (1994) 17–47

[11] Chan, H.L., Edmonds, J., Lam, T.W., Lee, L.K., Marchetti-Spaccamela, A., Pruhs, K.: Nonclairvoyant speed scaling for flow and energy. Algorithmica **61**(3) (2011) 507–517

[12] Anand, S., Garg, N., Kumar, A.: Resource augmentation for weighted flow-time explained by dual fitting. In: SODA. (2012) 1228–1241

[13] Devanur, N.R., Huang, Z.: Primal dual gives almost optimal energy efficient online algorithms. In: SODA. (2014)

[14] Gupta, A., Krishnaswamy, R., Pruhs, K.: Scalably scheduling power-heterogeneous processors. In: ICALP (1). (2010) 312–323

[15] Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. J. ACM **54**(1) (2007)

[16] Albers, S.: Energy-efficient algorithms. Commun. ACM **53**(5) (2010) 86–96