# Algorithms for Cost-Aware Scheduling

Janardhan Kulkarni* and Kamesh Munagala**

Department of Computer Science, Duke University
{kulkarni,kamesh}@cs.duke.edu

## 1 Introduction

In this paper, we generalize classical machine scheduling problems by introducing a cost involved in processing jobs, which varies as a function of time. Before defining the problems formally and discussing the technical novelty, we present a few technological motivations for introducing this model.

***Demand Response Models:*** Modern data centers are big consumers of electricity, and large providers see huge cost savings by even modest savings in electricity consumption. The conventional approach taken by system designers has been to build systems which are energy efficient, via technologies such as speed scalable processors, dynamic power-down and power-up mechanisms, new cooling technologies, and multi-core servers. These approaches have been investigated widely both in practice and theory; see [21,1,20] for more details.

Another relatively recent and less explored approach to reducing the energy costs is to exploit the *variable pricing of electricity*. The electricity markets in large parts of United States are moving towards variable pricing. As noted in [24], in those parts of the U.S. with wholesale electricity markets, prices vary on an hourly basis and are often not well correlated at different locations. Moreover, these variations are substantial, as much as a factor of ten from one hour to the next. Several suppliers offer "Time Of Use" plans [25], where they charge higher price for *peak* hours and considerably lower price during non-peak hours. Electricity markets are too complex to give a simple thumb rule on which this variation in cost depends but loosely speaking, the price depends on the resource used for generation: As the demand peaks, the cost goes up disproportionately as the suppliers have to rely on expensive and nonrenewable resources like coal to meet the demand [27]. In this sense, the cost of electricity is also an indicator of how "green" its generation is and its impact on environment.

This variation in prices of electricity offers opportunities for large scale system designers to cut down their electricity expenses by moving their workload both in space and time. Note that in contrast to energy efficient computing, the purpose of this line of work is not to reduce the amount of energy consumed per unit of work, but to reduce the cost for doing the work. In [24] the authors exploit the spatial nature of variation in electricity cost for scheduling, while

* Supported by NSF awards CCF-1008065 and IIS-0964560.
** Supported by an Alfred P. Sloan Research Fellowship, a gift from Cisco, and by NSF via grants CCF-0745761, CCF-1008065, and IIS-0964560.

T. Erlebach and G. Persiano (Eds.): WAOA 2012, LNCS 7846, pp. 201–214, 2013.
© Springer-Verlag Berlin Heidelberg 2013

in [10], the authors analyse a simple model to exploit the temporal nature of variation in electricity prices. The latter work is particularly relevant to our paper: They consider a system at a single location executing a workload that is delay tolerant, such as processing batch jobs. They further consider a pricing model where cost of electricity varies between two levels, a *base price* and a *peak price*. They propose simple schemes to defer the workload to less expensive base price periods, and show experimentally that it smoothly trades off costs for delay. The generalized scheduling problem we introduce models the effect temporal nature of variation in electricity prices on scheduling decisions. One of the highlights of our work is to analyze the model considered by authors in [10] from the theoretical perspective with worst case bounds (Section 2.2).

***Spot Pricing in Data Centers:*** An entirely different technological motivation for cost-aware scheduling comes from the Amazon EC2 cloud computing system, which allows users to rent virtual machines on the cloud for computational needs. Amazon offers various pricing schemes to rent machines, one of which is *spot pricing*. Spot pricing enables the users to bid for unused capacity, and prices get set based on supply and demand. Again, as in the previous example, the cost of renting the machine on EC2 varies dynamically over time, offering opportunities for optimizing the cost and QoS of batch jobs on such a system.

### 1.1 Our Model and the Two-Cost Problem

The optimization problems arising in above applications can be captured by the following problem that we term Two-Cost, which generalizes the classical single-machine preemptive scheduling framework. There is a set $J$ of $n$ jobs, where each job $J_j$ has processing time $p_j$, release time $r_j$, and weight $w_j$. Density of a job $J_j$ is defined as ratio of $\frac{w_j}{p_j}$. For simplicity, we assume time is discrete, and the processing times and release times of the jobs are integers. There is a *processing cost* function $e(t)$: If a job $J_j$ is scheduled at time $t$, it incurs processing cost $e(t)$. We assume that $e(t)$ is a piecewise constant function which takes exactly two distinct values, *high* and *low*, corresponding to the base and peak price of electricity markets. By scaling we assume that cost of processing at *high* time instants is $\beta$ and 1 in low time instants. Given any schedule, the *processing cost* of $J_j$ denoted by $E(j)$ is given by $\int_t e(t)x_j(t)dt$, where $x_j(t)$ indicates whether job $J_j$ was scheduled at the instant $t$; since we assume time is discrete, $E(j) = \sum_t e(t)x_j(t)$. The completion time of job $J_j$ denoted by $C_j$ is the last time instant when this job is scheduled. The flow time of $J_j$ is defined as $F_j = C_j - r_j$. Jobs arrive online and the cost function $e(t)$ changes in an online fashion as well. The objective is to minimize $\sum_{J_j \in J}(w_j F_j + E(j))$ – the sum of weighted flow time and processing cost.

### 1.2 Our Results and Techniques

In this paper we initiate the study of online scheduling problems with the objective of minimizing the processing costs plus some well-known QoS guarantees.

We show that these problems are significantly different from their counterparts without processing costs, hence requiring new algorithmic techniques. We assume the algorithm does not know the future job arrivals and the future cost function $e(t)$, and proceed via *speed augmentation* analysis, where we give the algorithm extra processing speed compared to $OPT$ for the purpose of analysis. The holy grail of speed augmentation analysis is to design a so-called *scalable* algorithm: For any $\epsilon > 0$, the algorithm is $(1+\epsilon)$-speed, $O(\text{poly}(\frac{1}{\epsilon}))$-competitive.

In Section 2, we first show that no deterministic online algorithm for Two-Cost can be constant competitive even for unit length, unit weight jobs, when $e(t)$ is known in advance. We then present our *main result*. We show a scalable $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^3})$-competitive algorithm for Two-Cost.

Our algorithm for Two-Cost in Section 2 is the most natural one: Always schedule at low cost time instants. For high cost time instants, if the total flow time accumulated since the last scheduling decision is at least the cost of processing, then schedule using the highest density first priority rule, else wait. We outline the idea behind the analysis in Section 2.1. The analysis is complicated by the non-uniform nature of the problem - the behavior of the algorithm is different in the high and low cost instants, and these instants themselves arrive in an online fashion. The key decision that the algorithm has to make is whether to schedule a job at the current step in a high cost time instant, or wait for a low cost time instant that may arrive soon in the future. However, waiting poses a risk in that jobs could arrive in the future and create a huge backlog at the low cost time instants. To partially mitigate the backlogging effect, we resort to speed augmentation for the analysis, and show that this is necessary as well.

*Technical Contributions.* Speed augmentation and potential functions have proven to be useful techniques in the analysis of online algorithms for weighted flow time problems. See for example, scheduling policies on unrelated machines [9] and *speed scaling* problems [5,4,13]. These potential functions follow a similar template (the so-called *standard potential function* [15]), and are defined in terms of the future online cost (or cost-to-go) of the algorithm assuming no more jobs arrive in the system and how far the online algorithm is behind the optimal schedule in work processed. Due to the online nature of the cost function $e(t)$, one of the chief technical difficulties we face while analyzing this problem is that it is hard to give a closed form expression for the future cost of an online algorithm. Hence, it is not clear how to define any sort of potential function for our problem. We instead proceed via simplifying the input and certain revealing structural properties of OPT. We then observe a simple yet powerful *majorization* property (Theorem 4) of our schedule relative to the optimal schedule, as consequence of the fact that our algorithm uses $HDF$ ordering of jobs. We use this characterization repeatedly in a non-trivial fashion to split the time horizon into suitably defined phases. Within these phases, we use a fairly simple charging argument to bound the cost of algorithm against the optimal. In effect we show that an algorithm that cannot estimate cost-to-go for its scheduling decision is competitive *even if* the costs in the future vary arbitrarily. We also believe Theorem 4 can be of independent interest in other scheduling problems.

### 1.3 Related Work and Comparision with Other Models

Use of speed augmentation in the analysis of online scheduling problems, particularly involving flow time objectives was first considered in [18]. Since then, several papers have used speed augmentation to show scalable algorithms for weighted flow time on various machine environments [8,6,9,16]. The introduction of the cost function $e(t)$ drastically changes the nature of the problem compared to the classical flow time problems. It is instructive to point out that our problem does not admit a competitive algorithm even for unit weight and unit length jobs, where as $HDF$ is optimal for unit length jobs in this context for classical flow time. The fundamental difference in the complexity of this problem was noted earlier in the works on *robust machine scheduling*, which is a special case of our model [26,11]. In the offline case, Epstein *et al.* study the problem of minimizing weighted completion on a single machine when the machine can encounter unexpected failures [11] . They give a constant factor approximation algorithm on a single machine when jobs have no release dates and show polylogarithmic lowerbounds when jobs have release dates. Note that, in their problem only failure periods arrive online. These results sharply contrast with constant factor approximation algorithms known for minimizing weighted completion time on various machine environments. We outline the results we obtain for the offline version of our problem in Section 3.

Dynamic speed scaling and its variants have been studied extensively for power management. In this model pioneered by Yao *et al.* [28], the goal is to dynamically scale the speed of a processor to optimize power consumed (which is usually a convex function of speed) and some QoS metric like deadlines or flow time. This model has a rich literature in online algorithms and potential function design; see [5,4,23,7,2,3,21,12,22] for more details. Apart from the philosophical difference that we are concerned with minimizing the cost of power rather than efficient usage of it, we believe that our model is technically very different as well. Electricity costs vary with time in a non-monotone and adversarial fashion, whereas in speed scaling, the cost incurred by algorithms depends on the speed and is not a function of time. Therefore, the decision in speed scaling is to set the speed, while in our problem, the decision is about which time instants to process the jobs in. It is also interesting to note the technical similarities in these policies. In speed scaling algorithms, speed is set such that cost incurred on the speed is equal to the flow time of jobs at any time instant. Our algorithm also uses similar cost-balancing approach towards time slot selection policy. It would be interesting to study the effect of combining speed scaling with our model.

Another line of research which has technical similarities with our problem is power down mechanisms [17,19]. There are speed scalable processors, but transitioning from speed zero (sleep state) to non-zero speed (active state) incurs activation cost. These algorithms use algorithms for speed scaling as subroutine when the processor is active, but they also need to decide when to transition into and out of sleep states. These algorithms hence balance the cost of activation to the total flow time of jobs present at that time. Our algorithm also uses a similar cost balancing approach, but there are several subtle differences. For example,

the algorithms in [17,19] transition to sleep state only when there are no jobs in the system whereas our algorithm for Two-Cost may idle even when there are jobs present simply because processing cost is high. Further, the speed-scaling model is more sympathetic to lazy activation policies like ones used in [19] since any accumulated jobs can be cleared by varying the speed. But accumulating jobs in our problem poses a threat, since both jobs and high cost time instants arrive online. We emphasize that, even for unit length and unit weight jobs we cannot get a competitive algorithm where as almost all the problems considered in speed-scaling or power-down mechanisms admit competitive algorithms. In other words, the use of speed augmentation in speed scaling problems is for converting the schedule for fractional objective to the integral objective, whereas we need it even for a fractional schedule.

## 2 Online Algorithms for the Two-Cost Problem

In this section, we devise online algorithms for minimizing the sum of weighted flow time and processing cost on a single machine (with preemption). Recall that we denote the weights of the jobs by $w_i$, and the processing times by $p_i$. We assume without loss of generality that $e(t)$ takes either a value of 1 or $\beta$ at all the time instants.

Before we present the scalable algorithm for Two-Cost, we first present lower bounds on the achievable competitive ratio. In this section, we first show that no online algorithm can have competitive ratio independent of the values taken by cost function $e(t)$, even when all jobs have the same weight and unit processing length, and when the cost function $e(t)$ is known in *advance*. We defer the proof of this lower bound to Appendix A.

**Theorem 1.** *No deterministic online algorithm for* Two-Cost *can have a competitive ratio independent of the values taken by* $e(t)$*, even when all jobs have unit length and equal weight and* $e(t)$ *is known in advance.*

To get around this negative result, our algorithms will use *speed augmentation* to be competitive - this means that to show a $O(1)$-competitive ratio, we pretend the algorithm runs on a *faster* machine than the optimal solution; the extra speed trades off with the competitive ratio. Let $OPT$ denote both the optimal offline algorithm, as well as its value. Given any online algorithm and input sequence, there are two decisions the algorithm has to make every step:

**Time Slot Selection:** This policy decides which time slots to schedule jobs - we term these *active* time slots.
**Job Selection Policy:** Decides which job to schedule in each active time slot.

We briefly describe the analysis technique of speed augmentation, which is implicit in previous work [9,5,4].

**Definition 1.** *Given an algorithm A and speed s, we say that algorithm B is a s-speed simulation of A if:*

- *The active time slots of $A$ and $B$ are the same (or $B$ simulates $A$).*
- *The job selection policy of $B$ is same as $A$; however, $B$ can process $s$ units of jobs in every active time slot.*

**Definition 2.** *An online algorithm $A$ is said to be $s$-speed, $c$-competitive if there is a $s$-speed simulation of $A$ that is $c$ competitive against $OPT$.*

We make this fine distinction for the reason that, schedule produced by $A$ when run a machine with speed $s$ can be completely different from schedule of $B$ which takes the schedule of $A$ on unit speed processor, but schedules $s$ units of jobs whenever $A$ processes 1 unit of job.We will show the following theorem in the sequel.

**Theorem 2.** Two-Cost *has a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^3})$-competitive algorithm.*

### 2.1 Proof Outline

We design our algorithm for the case when jobs have unit length with arbitrary weights and at each time step a single job is released. We later show how to convert this algorithm to handle jobs with arbitrary lengths using the ideas which have become standard now. For unit length jobs, the job selection policy of any algorithm is simple: Schedule that job $J_i$ from the current queue with highest *density* or weight. This is the well-known *Highest Density First* (HDF) policy. Our overall online algorithm for unit length jobs is the most natural one: Always schedule using HDF in low cost instants. For high cost instants, we follow a ski-rental kind of policy. If the total flow time accumulated since the last scheduling decision is at least $\beta$, then schedule using HDF, else wait. We call this algorithm Balance. The hard part in defining a (standard) potential function is the non-uniformity in the processing cost. Instead, we first transform and simplify the input so that we only have to deal with unit length jobs, only one of which arrives per step.

The crux of our analysis is a *majorization property* of the HDF schedule, Theorem 4: If an online algorithm processing using HDF always lags another algorithm in terms of number of units processed, but eventually catches up, then if the initial weight of jobs in the queue of the first algorithm was smaller, the final weight will be smaller as well. We show that Balance always lags $OPT$ in terms of number of jobs processed, hence the majorization result directly bounds the processing cost paid by Balance (Lemma 6).

To bound the flow time, we perform a speed augmentation analysis. Again, the analysis is complicated by the non-uniformity in processing costs between low and high cost time instants. We instead divide time into phases where the augmented Balance lags $OPT$. Using our majorization result, we show it is sufficient to analyze each phase separately. We now construct a simple charging scheme, and argue about the amortized cost, completing the proof (Lemma 7).

### 2.2 Simplifying the Input

By scaling the input, we can assume that $e(t)$ takes values either 1 or $\beta$. We also assume that processing times and release times of jobs take integer values. We

assume jobs are released at the beginning of a time slot. Our scheduling policies will be based on considering the weight of jobs in the queue *during* the time slot, and the processing happens at the end of the time slot.

We call a time instant $t$ as *high cost* time instant if $e(t) = \beta$ in the interval $[t, t+1)$. Other wise, we call it as *low cost* time instant. We assume without loss of generality that $e(t)$ changes only at integral values of $t$. Thus every time instant is either a low cost time instant or a high cost time instant.

*Step 1: Unit Length Jobs.* The following lemma is an easy consequence of similar results in [9,4,8]. The proof follows by replacing each job $J_i$ with $p_i$ jobs of unit length and weight $w_i/p_i$.

**Lemma 3.** *If an online algorithm $A$ is $s$-speed $c$-competitive for minimizing the objective $\sum_j w_j F_j$ for unit length jobs, then it is $(1+\epsilon)s$-speed $(1+\frac{1}{\epsilon})c$-competitive when jobs have arbitrary length.*

The above lemma allows us to focus on unit length jobs in designing the online algorithm. For unit length jobs, given the set of active slots, it is easy to characterize the job selection policy: If a slot is active, the algorithm will simply schedule that job $J_i$ from its queue with highest *density* or weight per unit length, $w_i$. This is the well-known *Highest Density First* (HDF) policy. Note however that even for unit length jobs, Theorem 1 shows there is no 1-speed algorithm with competitive ratio independent of $\beta$. We therefore need to use a speed augmentation analysis even for this case. We redefine $OPT$ to be the optimal offline algorithm for this new problem instance (with unit length jobs).

*Step 2: Modifying Release Dates.* We assume that only one job is released at each time step. This follows as consequence of Step 1 and we omit the details.

*Step 3: Modifying the optimal schedule.* Given any algorithm $A$, let $W^A(t)$ denote the total weight of jobs in $A$'s queue during time $t$. The proof of the following claim follows easily from the observation that making $OPT$ process a job has cost $\beta$, while the total weight of jobs in the queue contributes to the flow time. As a consequence, if $W^{OPT}(t) \geq \beta$, we can assume $OPT$ schedules at time $t$.

*Claim.* With $O(1)$ loss in competitive ratio, we can assume that in any interval $I = [s, d]$ where $OPT$ does not process jobs, $\sum_{t \in I} W^{OPT}(t) < \beta$.

### 2.3 Online Algorithm Balance

The online algorithm Balance is characterized by the following two rules. Here $W^{\mathcal{A}}(t)$ denotes the total weight of jobs in Balance's queue during time $t$.

**Time slot selection Policy:** If $e(t) = 1$, then mark $t$ as active. If $e(t) = \beta$ then let $t'$ be the last active time instant. Mark $t$ as active if $\sum_{u \in (t', t]} W^{\mathcal{A}}(u) \geq \beta$.

**Job selection Policy (HDF):** If $t$ is active, then among the set of jobs available at the time $t$, schedule the one with highest weight.

### 2.4 Analysis of BALANCE

We begin this section by showing some important structural properties of the schedule produced by BALANCE, which we use subsequently in our analysis.

**Majorization Property of HDF.** Before we analyze BALANCE, we observe a simple yet important property of scheduling jobs in HDF, which may be of independent interest. Let $A$ and $B$ denote two scheduling algorithms which process same number of unit length jobs in the interval $[s, d]$. Suppose $A$ processes jobs in $HDF$ and always lags $B$, i.e., total number of jobs processed by $A$ at any time $t \in [s, d]$ is always less than $B$. Then, majorization property says that if $A$ and $B$ started off with equal weight at the beginning, and jobs arrive in the interval $[s, d]$, then the weight of jobs in $A$ will have at most that in $B$ in the end of this interval. But more importantly, for every job $J_j$ in $A$'s queue, total weight of jobs in $B$'s queue with weight at least $w_j$ will be at most that of $A$'s. We make the statement formal below.

For a scheduling algorithm $A$ processing unit length jobs in the interval $[t_1, t_2]$, let $Q^A(t)$ denote the set of jobs $A$ has at the time $t$. Let $Q^A_{\geq w}(t)$ denote the subset of those jobs with weight at least $w$. Let $N^A(t_1, t_2)$ denote the number of jobs $A$ has scheduled in the interval $[t_1, t_2]$. Then we have the following theorem about scheduling jobs in HDF.

**Theorem 4.** *(Proved in Appendix A) Let $A$ be a scheduling algorithm which processes unit length jobs using the HDF job selection policy, and $B$ be any other scheduling algorithm on the same input. Suppose $\forall J_i \in Q^B(t_1), |Q^A_{\geq w_i}(t_1)| \leq |Q^B_{\geq w_i}(t_1)|$. If $N^A(t_1, t_2) = N^B(t_1, t_2)$ and $\forall t \in [t_1, t_2], N^A(t_1, t) \leq N^B(t_1, t)$, then $\forall J_i \in Q^B(t_2), |Q^A_{\geq w_i}(t_2)| \leq |Q^B_{\geq w_i}(t_2)|$. Further, $W^A(t_2) \leq W^B(t_2)$.*

**Bounding the Processing Cost.** From this point on, we will use $\mathcal{A}$ to denote the schedule produced BALANCE. The following lemma is the crucial property of BALANCE: Compared to $OPT$, $\mathcal{A}$ always lags in total processing done.

**Lemma 5.** *(Proved in Appendix A) In the schedule $\mathcal{A}$ produced by* BALANCE, *$\forall t \in [0, t], N^{\mathcal{A}}(0, t) \leq N^{OPT}(0, t)$.*

Let $E^{\mathcal{A}}, E^{OPT}$ denote the total processing cost of BALANCE and $OPT$ respectively. The processing cost $E^{\mathcal{A}}$ can now be bounded using the above lemmas.

**Lemma 6.** *(Proved in Appendix A) Total processing cost of $\mathcal{A}$, $E^{\mathcal{A}} \leq E^{OPT}$.*

**Flow Time via Speed Augmentation.** We now analyze the schedule produced by BALANCE using speed augmentation. In particular, we consider a class of algorithms, SIMULATE-BALANCE($s$) that uses the same active time-slots as BALANCE (in that sense, it simulates BALANCE). However, on each active time slot of BALANCE, the new algorithm schedules at most $s$ units of jobs from its own queue using the HDF policy.

It follows directly from Lemma 6 that the processing cost of SIMULATE-BALANCE($s$) is at most $s \cdot E^{OPT}$. In the sequel, we bound the flow time of

SIMULATE-BALANCE$(1+\epsilon)$ against the flow time of $OPT$, which we denote $F^{OPT}$. We will use SIMULATE-BALANCE to mean SIMULATE-BALANCE$(1+\epsilon)$, where the factor $(1+\epsilon)$ will be implicit.

In order to bound the weighted flow time, we first split the contribution of the weighted flow time to individual time steps (and hence time intervals). We treat time as a discrete quantity here with each time instant $t$ denoting the time interval $[t, t+1)$. For an algorithm $B$, let $F^B$ denote the total weighted flow time, and let $W^B(t)$ denote the weight of jobs in queue of $B$ at time instant $t$ (this excludes the job getting processed at the time step $t$). Then, we have: $F^B = \sum_{t\geq 0} W^B(t) + \sum_j w_j$.

Let $F_\epsilon^{\mathcal{A}}$ denote the weighted flow time of SIMULATE-BALANCE. Let $W_\epsilon^{\mathcal{A}}(t)$ denote the total weight of the jobs in the queue of SIMULATE-BALANCE at time $t$. Recall that $F^{OPT}$ and $E^{OPT}$ are the weighted flow time and processing cost of $OPT$, respectively. We will prove the following lemma in the sequel, which will complete the proof of Theorem 2.

**Lemma 7.** $F_\epsilon^{\mathcal{A}} \leq O\left(\frac{1}{\epsilon^2}\right)(F^{OPT} + E^{OPT})$.

Let $\mathcal{W}_\epsilon^{\mathcal{A}}(t)$ denote the fractional weight of jobs in the queue of SIMULATE-BALANCE; $\mathcal{W}_\epsilon^{\mathcal{A}}(t) = \sum_{J_i \in Q_\epsilon^{\mathcal{A}}} w_i x_i(t)$, where $x_i(t) \in [0,1]$ denotes the remaining processing time of $J_i$ at the time $t$. Then, weighted fractional flow time of SIMULATE-BALANCE is defined as: $f_\epsilon^{\mathcal{A}} = \sum_{J_j \in J} \frac{w_j}{2} + \sum_{t\geq 0} \mathcal{W}_\epsilon^{\mathcal{A}}(t)$.

We start by bounding the weighted fractional flow time $f_\epsilon^{\mathcal{A}}$ of SIMULATE-BALANCE. For an interval $[t_1, t_2]$, let $P_\epsilon^{\mathcal{A}}(t_1, t_2)$, $P^{OPT}(t_1, t_2)$ denote the total units of processing done by SIMULATE-BALANCE and $OPT$ in the interval $[t_1, t_2]$.

**Definition 3.**
- *An interval $[s, d)$ is a* lag-interval *if $P_\epsilon^{\mathcal{A}}(s, d) \geq P^{OPT}(s, d)$, but for all $t \in [s, d)$, $P_\epsilon^{\mathcal{A}}(s, t) < P^{OPT}(s, t)$.*
- *An interval $[s, d)$ is a* lead-interval *if for all $t \in [s, d)$,* SIMULATE-BALANCE *processes more units that time instant than $OPT$, but at time $d$, it processes less units than $OPT$.*
- *The entire time horizon partitions into a sequence of alternating lag and lead intervals of the form $[0, d_1), [d_1, d_2), \ldots$, where $0 < d_1 < d_2 \cdots$. We call the interval $[d_i, d_{i+1})$ as the $i^{th}$ phase.*

The following lemma, proved in Appendix A follows by a repeated application of Theorem 4.

**Lemma 8.** *For any phase $[d_i, d_{i+1})$, $\mathcal{W}_\epsilon^{\mathcal{A}}(d_i) \leq W^{OPT}(d_i)$. Furthermore, for any $t$ in a lead phase $[d_i, d_{i+1})$, we have $\mathcal{W}_\epsilon^{\mathcal{A}}(t) \leq W^{OPT}(t)$.*

***Proof of Lemma 7:*** We will bound the fractional flow time of SIMULATE-BALANCE against that of $OPT$ within each phase. For a *lead-phase*, the bound is simple since at every time instant $t$ within the phase, we have $\mathcal{W}_\epsilon^{\mathcal{A}}(t) \leq W^{OPT}(t)$. We therefore focus only on lag-phases. Since SIMULATE-BALANCE always lags $OPT$ in terms of number of units processed within a lag phase, we have the following easy consequences for any time instant in a lag phase:

- If Simulate-Balance processes, it processes at least $\epsilon$ more units of jobs than $OPT$.
- If Simulate-Balance does not process, then it has to be a high cost instant.

We will now use a charging argument to show that the fractional flow time of Simulate-Balance within the lag-phase is at most $O(1/\epsilon)$ times the total cost spent by $OPT$ within the phase. Fix a lag-phase $i$. We use $P^{OPT}(t)$ and $P_\epsilon^{\mathcal{A}}(t)$ to abbreviate $P^{OPT}(d_i, t)$ and $P_\epsilon^{\mathcal{A}}(d_i, t)$ respectively. Instead of analysing elaborate charging rules, for each time instant $t \in [d_i, d_{i+1})$ we define the following simple potential function, which keeps track of how far Balance is behind compared to $OPT$.

$$\Phi(t) = \frac{2\beta}{\epsilon}\left(P^{OPT}(t) - P_\epsilon^{\mathcal{A}}(t)\right) \tag{1}$$

The amortized cost paid by Simulate-Balance is defined as :

$$\theta(t) = \mathcal{W}_\epsilon^{\mathcal{A}}(t) + \Phi(t) - \Phi(t-1) \tag{2}$$

Define $[t, t')$ as an *idle period* if neither $OPT$ nor Simulate-Balance schedule jobs in that interval. The following lemma follows from an (almost) straightfoward analysis of the potential function in Equation (1). Only corner cases are the periods when the change in potential is zero (idle periods) and one has to bound cost of algorithm during such periods. We push the details to Appendix A.

**Lemma 9.** *For any lag-phase $[d_i, d_{i+1})$ and any idle period $X = [t, t')$ so that $t, t' \in [d_i, d_{i+1})$, we have:*

$$\sum_{t \in X} \theta(t) \leq O(\frac{1}{\epsilon}) \left( \sum_{t \in X} \left( W^{OPT}(t) + e(t) \cdot \mathcal{I}^{OPT}(t) \right) \right) \tag{3}$$

*where $\mathcal{I}^{OPT}(t)$ is the indicator variable denoting whether $OPT$ schedules at $t$.*

Equation (3) is true for lead-phases directly from Lemma 8. Therefore, summing over all phases, we conclude that the weighted fractional flow time of Simulate-Balance is at most $O(1/\epsilon)$ times the total cost of $OPT$. We then convert the weighted fractional flow time into weighted flow time by augmenting Simulate-Balance with another $(1+\epsilon)$-speed using ideas similar to that in Lemma 3. Therefore, we conclude that on a machine with $(1+\epsilon)$-speed, $F_\epsilon^{\mathcal{A}} \leq O(\frac{1}{\epsilon^2})(F^{OPT} + E^{OPT})$. This concludes the proof of Lemma 7.

***Proof of Theorem 2:*** We first bound the competitive ratio of Simulate-Balance for the objective minimizing $\sum_j (w_j F_j + E(j))$ when jobs are of unit length. From Lemma 6, it follows that total processing cost of Simulate-Balance is at most $(1+\epsilon)E^{OPT}$. Lemma 7 shows that on a machine with $(1+\epsilon)$-speed, $F_\epsilon^{\mathcal{A}} \leq \frac{1}{\epsilon^2}(F^{OPT} + E^{OPT})$. Putting all the pieces together, we conclude that Balance is $(1+\epsilon)$-speed $O\left(\frac{1}{\epsilon^2}\right)$-competitive for unit length jobs with arbitrary weights. Using Lemma 3 we finally conclude that Balance is $(1+\epsilon)$-speed $O\left(\frac{1}{\epsilon^3}\right)$-competitive for arbitrary length jobs.

## 3 Conclusion

In this paper, we presented a scalable online algorithm for the Two-Cost problem. In the full version of the paper, we show that with $K > 2$ levels of electricity cost, we need a speed augmentation of at least $K - 1$ to achieve bounded competitive ratio, even when electricity costs are known in advance. An interesting question that we seek to explore is whether such lower bounds can be circumvented using the framework of *speed scaling* [5], where the processor can be made to run faster by paying cost which is a convex function of the processing speed.

In the full version of the paper, we also study offline version of this problem with completion time objective, since approximating flow-time even without the cost function is one of the most important open problems in scheduling theory. We show that for the *offline* setting the LP formulation of Hall *et al.* [14] can be extended to yield a $O(\frac{1}{\epsilon})$ approximation to $1|r_j, pmtn| \sum_j (w_j F_j + E(j))$ with $(1+\epsilon)$-speed augmentation,for arbitrary $e(t)$. We also establish interesting connections of this problem to *universal scheduling* and scheduling with limited machine availability [11,26], which yield pseudo-polynomial time constant factor approximation algorithms for $1|r_j, pmtn| \sum_j (w_j F_j + E(j))$. See Appendix B.

**Acknowledgment.** We thank Sudipto Guha, Bruce Maggs, and Jeff Chase for several helpful discussions.

## References

1. Albers, S.: Algorithms for energy saving. In: Albers, S., Alt, H., Näher, S. (eds.) Festschrift Mehlhorn. LNCS, vol. 5760, pp. 173–186. Springer, Heidelberg (2009)
2. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. ACM Transactions on Algorithms 3(4) (2007)
3. Bansal, N., Chan, H.-L., Pruhs, K.: Speed scaling with an arbitrary power function. In: SODA (2009)
4. Bansal, N., Chan, H.-L., Pruhs, K.: Competitive algorithms for due date scheduling. Algorithmica 59(4) (2011)
5. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. J. ACM 54(1) (2007)
6. Bansal, N., Pruhs, K.: Server scheduling in the $l_p$ norm: A rising tide lifts all boat. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, STOC (2003)
7. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. In: SODA (2007)
8. Becchetti, L., Leonardi, S., Marchetti-Spaccamela, A., Pruhs, K.R.: Online weighted flow time and deadline scheduling. In: Goemans, M.X., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX-RANDOM 2001. LNCS, vol. 2129, pp. 36–47. Springer, Heidelberg (2001)
9. Chadha, J.S., Garg, N., Kumar, A., Muralidhara, V.N.: A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. In: STOC (2009)
10. Chase, J.: Demand response for computing centers, `http://www.cs.duke.edu/~chase/dr.pdf`

11. Epstein, L., Levin, A., Marchetti-Spaccamela, A., Megow, N., Mestre, J., Skutella, M., Stougie, L.: Universal sequencing on a single machine. In: Eisenbrand, F., Shepherd, F.B. (eds.) IPCO 2010. LNCS, vol. 6080, pp. 230–243. Springer, Heidelberg (2010)
12. Gupta, A., Im, S., Krishnaswamy, R., Moseley, B., Pruhs, K.: Scheduling heterogeneous processors isn't as easy as you think. In: SODA (2012)
13. Gupta, A., Krishnaswamy, R., Pruhs, K.: Scalably scheduling power-heterogeneous processors. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part I. LNCS, vol. 6198, pp. 312–323. Springer, Heidelberg (2010)
14. Hall, L.A., Shmoys, D.B., Wein, J.: Scheduling to minimize average completion time: Off-line and on-line algorithms. In: SODA (1996)
15. Im, S., Moseley, B., Pruhs, K.: A tutorial on amortized local competitiveness in online scheduling. SIGACT News 42(2) (2011)
16. Im, S., Moseley, B., Pruhs, K.: Online scheduling with general cost functions. In: SODA (2012)
17. Irani, S., Shukla, S., Gupta, R.: Algorithms for power savings. ACM Trans. Algorithms 3(4) (November 2007)
18. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM 47 (July 2000)
19. Lam, T.-W., Lee, L.-K., Ting, H.-F., To, I.K.K., Wong, P.W.H.: Sleep with guilt and work faster to minimize flow plus energy. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 665–676. Springer, Heidelberg (2009)
20. Murugesan, S.: Harnessing green it: Principles and practices. IT Professional 10(1) (2008)
21. Pruhs, K.: Green computing algorithmics. In: FOCS, pp. 3–4 (2011)
22. Pruhs, K., Stein, C.: How to schedule when you have to buy your energy. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX and RANDOM 2010. LNCS, vol. 6302, pp. 352–365. Springer, Heidelberg (2010)
23. Pruhs, K., Uthaisombut, P., Woeginger, G.: Getting the best response for your erg. ACM Trans. Algorithms 4, 38:1–38:17 (2008)
24. Qureshi, A., Weber, R., Balakrishnan, H., Guttag, J., Maggs, B.: Cutting the electric bill for internet-scale systems. In: SIGCOMM (2009)
25. Electricity rates, `http://www.pge.com/tariffs/electric.shtml`
26. Schmidt, G.: Scheduling with limited machine availability. European Journal of Operational Research 121, 1–15 (1998)
27. Official Statistics. United States Department of Energy, `http://www.eia.doe.gov`
28. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced cpu energy. In: FOCS (1995)

## A Missing Proofs

**Proof of Theorem 1.** Let $e(t) = \beta$ in the interval $[1, \ldots \sqrt{\beta}]$ and $e(t) = 1$ elsewhere. The adversary releases a unit length, unit weight job at each time instant $t \in [1, \ldots \sqrt{\beta}]$. Let $A$ be any online algorithm. Consider the number of jobs in the queue of $A$ at time $t = \sqrt{\beta}$. If $A$ has more than $\beta^{\frac{1}{4}}$ jobs then the adversary releases one job at each time instant $t > \sqrt{\beta}$. For this input, the

optimal offline algorithm will process each job released in the interval $[1, \ldots \sqrt{\beta}]$ by paying a processing cost of $\beta$, hence number of jobs it has at any time is at most one. However, $A$ accumulates $\beta^{\frac{1}{4}}$ jobs by the time $t = \sqrt{\beta}$ which it cannot clear subsequently. Hence there are at least $\beta^{\frac{1}{4}}$ jobs in its queue at every time instant $t > \sqrt{\beta}$, incurring a cost of $\beta^{\frac{1}{4}}$ towards flowtime at each time step. Therefore, the competitive ratio of $A$ is at least $\beta^{\frac{1}{4}}$. Next, consider the case when $A$ has less than $\beta^{\frac{1}{4}}$ jobs at time $t = \sqrt{\beta}$. In this case, the adversary will not release any more jobs. For this instance, the optimal offline algorithm will not process any jobs in the interval $[1, \ldots \sqrt{\beta}]$ and processes all jobs in the low cost time instants following $t > \sqrt{\beta}$ incurring a total cost of $O(\beta)$. The competitive ratio of $A$ is at least $\beta^{\frac{1}{4}}$, since the algorithm pays at least $\beta^{\frac{5}{4}}$ towards the processing cost.

**Proof of Theorem 4.** We prove this by contradiction. Suppose at time $t_2$, there is a job $J_i \in Q^B(t_2)$ such that $|Q^A_{\geq w_i}(t_2)| > |Q^B_{\geq w_i}(t_2)|$. Consider the set of jobs processed by $A$ in the interval $[t_1, t_2]$. If the weight of all these jobs is at least $w_i$, then since both algorithms process equal number of jobs in $[t_1, t_2]$ and $B$ had more jobs initially of weight at least $w_i$, it must have more jobs with at least weight $w_i$ at $t_2$. This is an immediate contradiction. Next, consider the case where $A$ processes a job of weight less than $w_i$ in the interval $[t_1, t_2]$. Let $t' \in [t_1, t_2]$ be the last time instant when $A$ scheduled a job with weight less than $w_i$. Since $A$ schedules jobs using HDF, it must be the case that $|Q^A_{\geq w_i}(t')| = 0$. Now observe that $A$ processes at least as many jobs as $B$ in the interval $[t', t_2]$. If $J_{\geq w_i}(t', t_2)$ denotes the set of jobs with weight greater than $w_i$ released in the interval $[t', t_2]$ then $|Q^A_{\geq w_i}(t_2)| = |J_{\geq w_i}(t', t_2)| - N^A(t', t_2)$. Since, $N^A(t', t_2) \geq N^B(t', t_2)$, we have $|Q^A_{\geq w_i}(t_2)| \leq |Q^B_{\geq w_i}(t_2)|$. Hence we get a contradiction.

**Proof of Lemma 5.** For the sake of contradiction, let $t_1$ be the first time instant when $N^{\mathcal{A}}(0, t_1) > N^{OPT}(0, t_1)$ and $t_2 < t_1$ be the last time instant when $\mathcal{A}$ scheduled a job. By the definition of $t_1$ and $t_2$, we note that both $OPT$ and $\mathcal{A}$ do not process any jobs in the interval $(t_2, t_1]$ and $N^{\mathcal{A}}(0, t_2) = N^{OPT}(0, t_2)$. Moreover, in the interval $[0, t_2]$ $\mathcal{A}$ lags $OPT$; that is, $\forall t \in [0, t_2]$, $N^{\mathcal{A}}(0, t) \leq N^{OPT}(0, t)$. Since BALANCE processes jobs in HDF, we apply Lemma 4 over the interval $[0, t_2]$ to claim that $W^{\mathcal{A}}(t_2) \leq W^{OPT}(t_2)$. If $t_1$ is a low cost time instant, we conclude that $OPT$ will also process a job since $W^{OPT}(t_1) > 0$. Next, consider the case when $t_1$ is a high cost time instant. Since $\mathcal{A}$ is processing a job at $t_1$, from the description of BALANCE, we have $\sum_{t=t_2+1}^{t_1} W^{\mathcal{A}}(t) \geq \beta$. However, $W^{OPT}(t_2) \geq W^{\mathcal{A}}(t_2)$ and $OPT$ does not process jobs in the interval $(t_2, t_1]$, then it must be the case that $\sum_{t=t_2+1}^{t} W^{OPT}(t) \geq \beta$. Therefore, by Claim 2.2 $OPT$ also processes at $t_1$. This completes the proof.

**Proof of Lemma 6.** Proof follows from Lemma 5. For the sake of contradiction, let $t$ be the first time instant when total processing cost of $\mathcal{A}$ is more than $OPT$. We note that in the interval $[0, t]$, when there are jobs to process, $\mathcal{A}$ has no idle time slots during the low cost time instants. Therefore, $\mathcal{A}$ schedules at least as many jobs as $OPT$ in the low cost time instants of the interval $[0, t]$. Since the total processing cost of $\mathcal{A}$ is more than $OPT$ at time $t$, $\mathcal{A}$ must have scheduled

more jobs in high cost time instants compared to $OPT$. This implies that total number of jobs scheduled by BALANCE in the interval $[0, t]$ is greater than that of $OPT$, which contradicts Lemma 5.

**Proof of Lemma 8.** We prove this by induction on $i$. If the phase is a lead phase, the induction is trivial since at each step, SIMULATE-BALANCE processes more units than OPT, hence for any $t \in [d_i, d_{i+1})$, we have $\mathcal{W}_\epsilon^{\mathcal{A}}(t) \leq W^{OPT}(t)$. For a lag phase $[d_i, d_{i+1})$, SIMULATE-BALANCE processes jobs in HDF, always lags OPT in terms of number of units processed within the phase, but catches up with OPT at time $d_{i+1}$. We simply invoke the Theorem 4 on the jobs processed within the phase to argue that if $\mathcal{W}_\epsilon^{\mathcal{A}}(d_i) \leq W^{OPT}(d_i)$, then $\mathcal{W}_\epsilon^{\mathcal{A}}(d_{i+1}) \leq W^{OPT}(d_{i+1})$. The details are straightforward and omitted.

**Proof of Lemma 9.** By the description of BALANCE and since at most one job arrives each time step, we have: $\sum_{t \in X} \mathcal{W}_\epsilon^{\mathcal{A}}(t) \leq 2\beta$. At time $t'$, there are two cases:

SIMULATE-BALANCE **schedules:** In this case, it schedule $\epsilon$ more units than $OPT$, so the potential drops by at least $2\beta$. The sum of flow time over the idle period is at most $2\beta$ and hence, the amortized cost is at most 0.

SIMULATE-BALANCE **does not schedule:** In this case, this time instant is a high cost time instant. $OPT$ pays at least $\beta$ in processing cost. The potential increases by at most $\frac{2\beta}{\epsilon}$, while SIMULATE-BALANCE pays at most $2\beta$ in flow time. Therefore, the amortized cost of SIMULATE-BALANCE is at most $O(\beta/\epsilon)$.

In either case, the amortized cost paid by SIMULATE-BALANCE is at most $O(1/\epsilon)$ times $OPT$'s flow time plus processing cost. Next, note that $\Phi(d_i) - \Phi(d_{i+1} - 1)$ is non-negative in the entire time interval $[d_i, d_{i+1})$ since SIMULATE-BALANCE lags $OPT$. From Lemma 8 we know that $\mathcal{W}_\epsilon^{\mathcal{A}}(d_i) \leq W^{OPT}(d_i)$. Hence we conclude that over the interval $[d_i, d_{i+1})$, total weighted fractional flow time of SIMULATE-BALANCE is upper bounded by:

$$\sum_{t=d_i}^{d_{i+1}-1} \mathcal{W}_\epsilon^{\mathcal{A}}(t) \leq O(\frac{1}{\epsilon}) \sum_{t=d_i}^{d_{i+1}-1} \left(W^{OPT}(t) + e(t) \cdot \mathcal{I}^{OPT}(t)\right)$$

## B Discussion on Offline Case

See the link http://www.cs.le.ac.uk/events/WAOA2012/AppendixKM.pdf