Scheduling for Speed Bounded Processors

Nikhil Bansal[1], Ho-Leung Chan[2], Tak-Wah Lam[3], and Lap-Kei Lee[3]

[1] IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY
nikhil@us.ibm.com
[2] Computer Science Department, University of Pittsburgh
hlchan@cs.pitt.edu
[3] Department of Computer Science, University of Hong Kong, Hong Kong
{twlam, lklee}@cs.hku.hk

Abstract. We consider online scheduling algorithms in the dynamic speed scaling model, where a processor can scale its speed between 0 and some maximum speed T. The processor uses energy at rate s^α when run at speed s, where $\alpha > 1$ is a constant. Most modern processors use dynamic speed scaling to manage their energy usage. This leads to the problem of designing execution strategies that are both energy efficient, and yet have almost optimum performance.

We consider two problems in this model and give essentially optimum possible algorithms for them. In the first problem, jobs with arbitrary sizes and deadlines arrive online and the goal is to maximize the throughput, i.e. the total size of jobs completed successfully. We give an algorithm that is 4-competitive for throughput and $O(1)$-competitive for the energy used. This improves upon the 14 throughput competitive algorithm of Chan et al. [10]. Our throughput guarantee is optimal as any online algorithm must be at least 4-competitive even if the energy concern is ignored [7]. In the second problem, we consider optimizing the trade-off between the total flow time incurred and the energy consumed by the jobs. We give a 4-competitive algorithm to minimize total flow time plus energy for unweighted unit size jobs, and a $(2 + o(1))\alpha/\ln\alpha$-competitive algorithm to minimize fractional weighted flow time plus energy. Prior to our work, these guarantees were known only when the processor speed was unbounded ($T = \infty$) [4].

1 Introduction

In the last few years, the increasing computing power of processors has caused dramatic increase in their energy consumption. This not only leads to high cooling costs but also to substantially reduced battery life in laptops and other mobile devices. Companies such as IBM, Intel and AMD have made power aware design a key priority and even scrapped the development of faster processors in favor of lower power ones. To be more energy efficient, many modern processors now adopt the technology of *dynamic speed (voltage) scaling* (see, e.g., [13, 21, 23]), where the processor can adjust its speed dynamically in some range without any overhead. For example, IBM's PowerPC 970FX [1] allows the operating system to dynamically vary the speed (with zero overhead) at various discrete points from 2.5GHz to 625MHz while the power consumption reduces from 100W to less than 10W. In general, the rate of energy usage varies approximately

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 409–420, 2008.

as s^α with speed s, where α is typically 2 or 3 [9,20][1]. Most research in dynamic voltage scaling has focused on how to exploit this capability to reduce energy consumption without an apparent reduction in the functionality offered by the system.

A theoretical investigation of dynamic speed scaling was initiated by the seminal paper of Yao, Demers and Shenker [24]. They considered a model where the processor can run at any speed between 0 and infinity, incurring an energy (cost) of s^α per time unit when run at speed s. We call this the *infinite* speed model. In this model, Yao et al. studied the problem where jobs with arbitrary sizes and deadlines are released over time in an online manner. The goal is to find a schedule that completes all jobs by their deadlines while minimizing the total energy used. They gave an algorithm that is $2^{\alpha-1}\alpha^\alpha$-competitive for energy, and proposed another algorithm OA. Bansal, Kimbrel and Pruhs [3] showed that OA is α^α-competitive and this ratio is tight (OA is discussed in further detail later). They gave another algorithm BKP which is about $2e^{\alpha+1}$-competitive for large α, and moreover showed that any algorithm must be $\Omega((4/3)^\alpha)$-competitive (as a function of α). The deadline scheduling problem has also been considered for other measures such as minimizing the maximum speed and minimizing the maximum temperature [3].

For scheduling jobs without deadlines, a commonly used Quality of Service (QoS) measure is the *flow time* or more generally the *weighted flow time*. The flow time of a job is the time taken to complete a job since it is released. Clearly, minimizing flow time and minimizing energy usage are orthogonal objectives. To understand their tradeoffs, Albers and Fujiwara [2] proposed combining the dual objectives into a single objective of total flow time plus energy.[2] Albers and Fujiwara focused on scheduling jobs of unit size, and they gave an $8.3e((3+\sqrt{5})/2)^\alpha$-competitive algorithm for minimizing unweighted flow time plus energy. Bansal, Pruhs and Stein [4] considered the more general problem of minimizing weighted flow time plus energy. They gave a 4-competitive algorithm for jobs of unit size and weight. They also gave a $\mu_\epsilon\gamma$-competitive algorithm for jobs of arbitrary size and weight, where ϵ can be any positive constant, $\mu_\epsilon = \max\{(1+1/\epsilon),(1+\epsilon)^\alpha\}$ and $\gamma = \max\{2, \frac{2(\alpha-1)}{\alpha-(\alpha-1)^{1-1/(\alpha-1)}}\}$. There are a large number of other related works for dynamic speed scaling (in the infinite speed model) and more generally for power management. We refer the readers to a survey by Irani and Pruhs [15] for more details.

Even though the infinite speed model is rather unsatisfying to model a real processor, it is a convenient theoretical model to work with. Typically it allows the algorithm to focus only on the speed setting aspect. For example in the deadline scheduling problem described above, as all jobs can always be completed, any algorithm can be assumed to schedule jobs by Earliest Deadline First (EDF) and hence it only needs to specify the speed at any given time. Moreover, the only relevant measures of the quality of the schedule are energy related. Another important reason is that it gives the online algorithm flexibility to recover from past mistakes; for example if the algorithm realizes that it has been working too slowly thus far, it can always try to catch up by speeding up. This often makes the algorithm design easier and more amenable to analysis.

[1] This does not hold at very low speeds due to leakage power effects that do not scale with speed.

[2] Without loss of generality, by changing the units of time or energy if necessary, it can be assumed that the user is willing to spend one unit of energy to improve one unit of flow time.

Recently Chan et al. [10] introduced the *bounded* speed model, where the processor speed can vary between 0 and some maximum T. They considered the deadline scheduling problem in this model. Note that when the maximum speed is bounded, even the optimal offline algorithm may not be able to complete all jobs. A natural objective is to maximize the throughput, defined as the total work of jobs that are successfully completed by their deadlines. In traditional scheduling (where speed is fixed) Koren and Shasha [16] gave an algorithm D^{over} which is 4-competitive on throughput. Moreover Baruah et al. [7] showed that this is the best possible throughput competitive ratio for any online algorithm. Thus running D^{over} at speed T is clearly 4-competitive for throughput; however it can be arbitrarily worse with respect to energy. Chan et al. [10] considered energy efficient algorithms for throughput maximization. They designed an online algorithm that is 14-competitive for throughput, and its energy consumption is at most $O(1)$ times that of any offline solution that maximizes the throughput.

1.1 Our Results

We consider two problems in the bounded speed model: energy efficient algorithms for throughput maximization, and that for minimizing weighted flow time.

Throughput maximization. We first discuss the throughput maximization problem. Our main result is an algorithm Slow-D which matches the optimum throughput guarantee of Koren and Shasha [16] while being $O(1)$-competitive for energy.

Theorem 1. *There is an online algorithm Slow-D that is 4-competitive with respect to throughput and* $(\alpha^\alpha + \alpha^2 4^\alpha)$*-competitive with respect to energy.*

Roughly speaking, Slow-D is a combination of OA and D^{over}. At any time, if all the remaining jobs can be completed using speed T, Slow-D admits all jobs and runs at the same speed as OA; otherwise, i.e., not all jobs can be completed, Slow-D uses the same job selection rule as D^{over} and runs at speed T. Hence, Slow-D uses a more sophisticated job selection method than the 14-competitive algorithm of [10], and differs from D^{over} by running at a slower speed when there is little work remaining. To prove the competitive ratio of Slow-D, the main novelty is a tighter analysis which accounts for the jobs that D^{over} can complete but Slow-D may miss due to the slower speed.

The setting we have considered so far is *overloaded* in that there may be too much work and no algorithm can finish all the jobs even running at the maximum speed at all times. A special case is the *underloaded* setting where all jobs can be completed by running at the maximum speed at all times. In traditional scheduling with a fixed speed processor, running EDF clearly completes all jobs and hence is 1-competitive for throughput. Yet no energy efficient algorithm can be 1-competitive with respect to throughput[3]. However, we can show that near-optimum throughput can be achieved in the underloaded setting. In particular, we have devised an algorithm called TimeSlot(ϵ) that is $(1+\epsilon)$-competitive for throughput and $(1+1/\epsilon)^\alpha \alpha^\alpha$-competitive for energy. Details will be given in the full paper.

[3] To be 1-competitive, an algorithm must always run at maximum speed on whatever little work has arrived thus far, otherwise the adversary can release new jobs to make the algorithm fail to complete all jobs.

To achieve 1-competitiveness in throughput, we consider resource augmentation where the online algorithm is allowed to relax its maximum speed to make up for its lack of future knowledge. In the fixed speed setting without energy concern, Lam and To [17] gave a 1-throughput competitive algorithm using a $2T$-speed processor. Chan et al. [10] gave an energy efficient algorithm that is $(1 + 1/\epsilon)$-competitive for throughput by relaxing the maximum speed to $(1 + \epsilon)T$. We can show that for the overloaded setting, there is an online algorithm that is 1-competitive for throughput and $(1 + 2/\epsilon)^\alpha(\alpha^\alpha + \alpha^2 4^\alpha)$-competitive for energy when using maximum speed $(2 + \epsilon)T$; and for the underloaded setting, there is an online algorithm that is 1-competitive for throughput and $(1 + 1/\epsilon)^\alpha \alpha^\alpha$-competitive for energy when using maximum speed $(1 + \epsilon)T$.

Minimizing weighted flow time. We now discuss minimizing weighted flow time plus energy in the bounded speed model. Our results are obtained by first obtaining a guarantee for weighted *fractional* flow time plus energy (see Section 2 for definition) and then rounding it. We first consider the case of jobs with unit size and weight. We are able to show that there are online algorithms that are respectively 2-competitive for fractional flow time plus energy, and 4-competitive for total flow time plus energy.

In this paper we focus on the case of jobs with arbitrary size and weight. Note that when the maximum speed T is very small (say T^α is less than the smallest job weight), any algorithm must work at speed T whenever there is unfinished work. In this case, the problem reduces to the classic problem of minimizing weighted flow time on a fixed speed processor, without any energy concern [11, 6, 8]. For this problem it has been recently shown that no $O(1)$-competitive algorithm is possible without resource augmentation [5]. Thus we need to relax the maximum speed of the online algorithm to $(1 + \epsilon)T$ in order to be $O(1)$-competitive for total weighted flow time plus energy.

Theorem 2. *For jobs with arbitrary size and weight, there is an online algorithm that is* $((2 + o(1))\alpha/\ln \alpha)$*-competitive for fractional weighted flow time plus energy. Furthermore, there is an online algorithm that given any* $\epsilon > 0$*, uses a processor with maximum speed* $(1 + \epsilon)T$*, and is* $\mu_\epsilon((2 + o(1))\alpha/\ln \alpha)$*-competitive for total weighted flow time plus energy, where* $\mu_\epsilon = \max\{(1 + 1/\epsilon), (1 + \epsilon)^\alpha\}$.

Note that both guarantees mentioned above essentially match those of [4] for the special case of the infinite speed model. Our results here are based on generalizing and extending the analysis in [4]. As we will discuss in Section 2, the algorithms in the infinite speed model set the speed such that at any time the rate of increase of flow time is equal to the rate of increase of energy. However, this is not always possible in the bounded speed model, which makes the analysis substantially harder.

Discrete speed levels. Our results can be easily adapted to discrete speed levels. The idea is to set the maximum speed to the highest speed level and round up the speed function to the next higher level. It maintains the performance on throughput and flow-time, while the energy usage is increased by at most a factor of Δ^α, where Δ is the maximum ratio of two consecutive non-zero speed levels.

2 Preliminaries

Throughout the paper we assume jobs arrive online over time. We use $r(J)$, $p(J)$, $d(J)$ and $w(J)$ (wherever applicable) to denote the release time, size (processing requirement), deadline and weight respectively of a job J. We consider scheduling algorithms for a single processor, and assume that jobs can be preempted arbitrarily without any penalty. The throughput of a schedule is defined as the total size of jobs that are successfully completed by their deadlines (no partial credit is obtained for incomplete jobs). All our results for throughput assume that the jobs are unweighted.

Given a schedule, the *flow time* of a job is the amount of time since this job is released until it completes. Equivalently, flow time of a job is simply its total cost if it pays one unit for each unit of time until it completes. The *fractional* flow time of a job is defined as its total cost if at each time unit it pays an amount equal to its unfinished fraction. The weighted flow time of a job and the fractional weighted flow time are defined analogously. Often fractional weighted flow time is much more convenient to work with. The (online) Highest Density First (HDF) algorithm, which at any time works on the job with the highest weight to size ratio, is optimal for minimizing fractional weighted flow time. On the other hand, no O(1)-competitive online algorithm exists for weighted flow time [5]. Note that at any time the total weighted flow time increases at a rate equal to the total weight of currently unfinished jobs, and the total energy usage increases at a rate s^α where s is the current speed. To trade off energy and flow time, a natural algorithm first proposed by Albers and Fujiwara [2] and analyzed in [4], is to set the speed s such that s^α is equal to the unfinished weight. However, this cannot be done in the bounded speed model as the unfinished weight can be much larger than T^α.

Algorithm OA. We explain the Optimal Available (OA) algorithm proposed by Yao, Demers and Shenker [24] for the infinite speed model. Note that in the infinite speed model it suffices to specify the speed at any time (jobs are always scheduled by EDF). Roughly speaking, the OA algorithm is the "laziest" possible algorithm that at any time works at the average speed just enough to complete all jobs feasibly. Formally, let $p_t(x)$ denote the amount of unfinished work at time t that has deadline within the next x units, then $p_t(x)/x$ is a lower bound on the average rate at which any feasible algorithm must work. At any time t, OA works at speed $s_{OA}(t) = \max_x p_t(x)/x$. For example, consider the instance where a job of size 1 and deadline n arrives at each of the times $0, 1, 2, \ldots, n-1$. The optimum schedule works at speed 1, and incurs total energy of n. On the other hand, OA starts with speed $1/n$ during $[0, 1]$, and has speed $1/n+\ldots+1/(n-i)$ during $[i, i+1]$ for $i = 0, \ldots, n-1$. Note that during $[n-1, n]$, OA consumes energy at rate about $(\ln n)^\alpha$, which is substantially larger than that consumed by the optimum solution at any time. Interestingly however, OA is α^α-competitive with respect to energy [3].

Another useful view of OA is the following. At any time t, OA computes the optimum energy schedule for the unfinished work assuming that no more jobs will arrive, and proceeds accordingly. When more jobs arrive in the future it recomputes this schedule and continues. Let $s^t_{OA}(t')$ denote the speed function (for times $t' > t$) computed by OA at time t, assuming no more jobs arrive after time t. s^t_{OA} is a decreasing piecewise

step function, i.e., it has speed s_j during $I_j = [t_j, t_{j+1}]$ for $j = 0, 1 \ldots$, where $s_0 > s_1 > s_2 > \ldots$, and $t_0 = t$. Moreover, only jobs with deadlines in the interval I_j are executed during I_j. The intervals I_j change when new jobs arrive, but for a fixed t' the computed speed $s^t_{OA}(t')$ can only increase with time t.

3 Energy Efficient Throughput Maximization

Recall that no algorithm can be better than 4-competitive for throughput even without energy concern. Moreover, executing D^{over} at the maximum speed T is 4-competitive for throughput, but it does not guarantee energy efficiency. In this section, we give an energy efficient algorithm Slow-D that is optimally competitive for throughput.

3.1 Algorithm Description

Consider the execution of OA with an unbounded speed processor. Let $s_{OA}(t)$ denote the speed of OA at time t, and without loss of generality we assume that OA follows the EDF policy. We design an algorithm Slow-D that simulates OA and makes decisions based on its own state and that of OA. At any time t, Slow-D works at speed $\tilde{s}(t) = \min\{s_{OA}(t), T\}$. Note that unlike OA (which works in infinite speed model), Slow-D may not complete all the jobs, so we need to specify carefully a job selection and execution strategy.

We first define the notion of down-time(t) that is critically used by Slow-D. At any time t, consider the schedule s^t_{OA} computed by OA assuming no new jobs arrive. We define down-time(t) as the latest time $t' \geq t$ such that the speed $s^t_{OA}(t') \geq T$. If $s_{OA}(t) < T$, and no such t' exists, we set down-time(t) to be the last time before t when the speed was at least T (or 0 if the speed was always below T). By the nature of OA, down-time(t) is a non-decreasing function of t no matter how jobs arrive. At any time t, we label all released jobs (including those OA has completed) based on down-time(t). A job J is called *t-urgent* if $d(J) \leq$ down-time(t), and is called *t-slack* otherwise. Note that a t-slack job may turn into a t'-urgent job at a later time $t' > t$. However, since down-time is non-decreasing, a t-urgent jobs stays urgent until it completes or is discarded. We call a job *slack* if it always remains slack during its entire lifespan; otherwise, it is called *urgent*. We now describe Slow-D.

Slow-D stores all released jobs in two queues $\mathrm{Q}_{\mathrm{work}}$ and $\mathrm{Q}_{\mathrm{wait}}$. At any time t, it processes the job in $\mathrm{Q}_{\mathrm{work}}$ with the earliest deadline at speed $\tilde{s}(t) = \min\{s_{OA}(t), T\}$. As we shall see $\mathrm{Q}_{\mathrm{wait}}$ is empty whenever $\mathrm{Q}_{\mathrm{work}}$ is empty. Slow-D admits jobs as follows:

> **Job arrival.** Consider a job J released at time r_j. J is admitted to $\mathrm{Q}_{\mathrm{work}}$ if either J is r_j-slack, or J and the remaining work of all r_j-urgent jobs in $\mathrm{Q}_{\mathrm{work}}$ can be completed using speed T. Otherwise, J is admitted to $\mathrm{Q}_{\mathrm{wait}}$. Note that jobs admitted to $\mathrm{Q}_{\mathrm{wait}}$ are all urgent.
> We say that an *urgent period* begins at r_j if $\mathrm{Q}_{\mathrm{work}}$ contains no urgent job before r_j and J is an urgent job admitted into $\mathrm{Q}_{\mathrm{work}}$.
> **Latest starting time (*ℓst*) interrupt.** Whenever a job J in $\mathrm{Q}_{\mathrm{wait}}$ reaches its latest starting time, i.e., current time $t = d(J) - (p(J)/T)$, it raises an *ℓst interrupt*. At an interrupt we either discard J or else expel all t-urgent jobs in $\mathrm{Q}_{\mathrm{work}}$ to make room for J as follows:

In the current urgent period[4], let J_0 be the last job admitted from $\mathrm{Q}_{\mathrm{wait}}$ to $\mathrm{Q}_{\mathrm{work}}$ (if no jobs have been admitted from $\mathrm{Q}_{\mathrm{wait}}$ so far, let J_0 be a dummy job of size zero admitted just before the current period starts). Consider all the jobs ever admitted to $\mathrm{Q}_{\mathrm{work}}$ that have become urgent after J_0 has been admitted to $\mathrm{Q}_{\mathrm{work}}$, and let W denote the total original size of these jobs. If $p(J) > 2(p(J_0)+W)$, all t-urgent jobs in $\mathrm{Q}_{\mathrm{work}}$ are expelled and J is admitted to $\mathrm{Q}_{\mathrm{work}}$.

Job completion. When a job J completes at time t, remove it from $\mathrm{Q}_{\mathrm{work}}$. If $\mathrm{Q}_{\mathrm{work}}$ contains no more t-urgent job, the current urgent period ends.

Note that the above *urgent period* is defined in such a way that at any time t during an urgent period, only t-urgent job is being processed.

3.2 Analysis

As Slow-D works at speed $\min(s^t_{OA}, T)$, the energy competitiveness follows directly from the result of [10].

Theorem 3. [10] *Any algorithm that works according to the speed function* $\tilde{s}(t) = \min(s^t_{OA}, T)$ *is* $(\alpha^\alpha + \alpha^2 4^\alpha)$*-competitive for energy against any offline algorithm that maximizes the throughput.*

Our goal now is to show that Slow-D is 4-competitive with respect to throughput. We partition the job sequence I into three sets. Let I_ℓ be the set of jobs admitted to $\mathrm{Q}_{\mathrm{wait}}$ upon arrival (these may join $\mathrm{Q}_{\mathrm{work}}$ later after raising *ℓst* interrupts). Among the jobs admitted immediately to $\mathrm{Q}_{\mathrm{work}}$ upon arrival, we let I_s denote those that are slack through their lifespan, and let I_t denote the ones that become urgent at some time before they expire. Note that I_s, I_t and I_ℓ are disjoint. Indeed, I_s and $I_t \cup I_\ell$ are respectively the sets of all slack jobs and all urgent jobs in I.

We first show that all slack jobs are completed by Slow-D.

Lemma 1. *Slow-D completes all jobs in* I_s.

Proof. Consider the execution of slack jobs under the unbounded speed OA. Since these jobs never become urgent, they are always executed at speed strictly less than T by OA. On the other hand, whenever OA runs at speed T or above, OA is working on an urgent job. To ease our discussion, for any time t, we call t a peak time if OA is running at speed T or above; otherwise, t is said to be a leisure time. At any leisure time t, Slow-D as well as OA can only work on some t-slack job (because down-time$(t) < t$ and all t-urgent jobs must have deadline before t). A t-slack job can never been executed at any peak time before t by OA; yet this might be possible for Slow-D. Together with the fact that both Slow-D and OA are using EDF, we conclude that at any leisure time t, Slow-D does not lag behind OA on any t-slack job (including any slack job). Thus all the slack jobs are completed by Slow-D. □

We will show that Slow-D completes enough jobs in I_t and I_ℓ. In particular, we consider each urgent period $P = [S_P, E_P]$ separately and derive a lower bound of the total size

[4] As we shall see *ℓst* interrupts occur only during urgent periods.

of urgent jobs that Slow-D completes in P. The following lemma is key to our lower bound. It requires two notations: join(P) denotes the total size of jobs in I_t that become urgent at some time in P. Let J^* be the latest-deadline job in I_ℓ that is released during P (irrespective of whether it is admitted to $\mathrm{Q}_{\mathrm{work}}$ later or not). We define a *secured interval* $P' = [S'_P, E'_P]$ for P, where $S'_P = S_P$ and $E'_P = \max\{d(J^*), E_P\}$.

Lemma 2. *For any urgent period P, the total size of urgent jobs completed by Slow-D is at least $\frac{1}{4}$ of (join$(P) + |P'| \times T$).*

Before proving Lemma 2, we show how it implies our main result. Let $p(I_t)$ be the total size of all jobs in I_t. Let span(I_ℓ) be the union of the spans of all jobs in I_ℓ, which may consist of a number of disjoint intervals, and let $|\text{span}(I_\ell)|$ be the total length of these intervals.

Lemma 3. *The total size of urgent jobs completed by Slow-D is at least $\frac{1}{4}(p(I_t) + |$span$(I_\ell)| \times T)$.*

Proof. Let C be the collection of all urgent periods. By Lemma 2, the total size of urgent jobs completed by Slow-D over all urgent periods is at least $\frac{1}{4}\sum_{P\in C}(\text{join}(P) + |P'| \times T)$.

For each job $J \in I_t$, J joins $\mathrm{Q}_{\mathrm{work}}$ during some urgent period, so $p(I_t) = \sum_{P\in C} \text{join}(P)$. Similarly, for each job $J \in I_\ell$, J is released during some urgent period, so $|\text{span}(I_\ell)| \le \sum_{P\in C} |P'|$. Summing the above two equality and inequality, we obtain $\sum_{P\in C}(\text{join}(P) + |P'| \times T) \ge p(I_t) + |\text{span}(I_\ell)| \times T$. □

Theorem 4. *Slow-D is 4-competitive on throughput.*

Proof. Let $p(I_s)$ be the total size of all jobs in I_s. By Lemma 1 and 3, the total size of jobs completed by Slow-D is at least $p(I_s) + (1/4) \times (p(I_t) + |\text{span}(I_\ell)| \times T)$. Clearly, any offline algorithm can complete at most $p(I_s)$ work on jobs in I_s, at most $p(I_t)$ work on jobs in I_t and at most $|\text{span}(I_\ell)| \times T$ work on jobs in I_ℓ, which implies the result. □

The rest of this section proves Lemma 2. Consider an arbitrary urgent period $P = [S_P, E_P]$. At S_P, some urgent jobs start to appear in $\mathrm{Q}_{\mathrm{work}}$. These jobs may be released at S_P, or already in $\mathrm{Q}_{\mathrm{work}}$ before S_P but just become urgent as down-time(S_P) becomes greater than their deadlines. As time goes on, more urgent jobs may appear in $\mathrm{Q}_{\mathrm{work}}$, and jobs in $\mathrm{Q}_{\mathrm{wait}}$ may raise *ℓst* interrupts. Assume that k jobs J_1, J_2, ..., J_k in $\mathrm{Q}_{\mathrm{wait}}$ are admitted successfully to $\mathrm{Q}_{\mathrm{work}}$ during P at times $L_1 \le L_2 \le \cdots \le L_k$, respectively. For notational convenience, we let J_0 and J_{k+1} be jobs of size zero, admitted at $L_0 = S_P$ just before any urgent job appears in $\mathrm{Q}_{\mathrm{work}}$ and at $L_{k+1} = E_P$ just after all urgent jobs are removed from $\mathrm{Q}_{\mathrm{work}}$, respectively. Note that during P, Slow-D always runs at speed T and works on urgent jobs. It completes at least J_k and all jobs in $\mathrm{Q}_{\mathrm{work}}$ that are found to be urgent after J_k is admitted and before E_P, as these urgent jobs are not expelled by an interrupt.

We now show a useful property about *ℓst* interrupts of jobs in I_ℓ.

Lemma 4. *Every job $J \in I_\ell$ that is released during an urgent period P must raise an ℓst interrupt in P (irrespective of whether it is admitted to $\mathrm{Q}_{\mathrm{work}}$ later or not).*

Proof. Consider the time $r(J)$ during P when J is released. Since J was placed in $\mathrm{Q}_{\mathrm{wait}}$, there was more than $T(d(J) - r(J) - (p(J)/T))$ urgent work in $\mathrm{Q}_{\mathrm{work}}$ at $r(J)$. At time progresses, more jobs in I_t may join $\mathrm{Q}_{\mathrm{work}}$, or some jobs in I_ℓ may be admitted successfully. In either case, the total amount of admitted urgent work can only increase. Thus, $E_P > r(J) + (d(J) - r(J) - (p(J)/T)) = d(J) - (p(J)/T)$ which is exactly when J raises the interrupt. □

To lower bound the work completed by Slow-D in P, we refine the notation join as follows: For any i, i' such that $0 \leq i < i' \leq k+1$, let $\mathrm{join}(J_i, J_{i'})$ be the total size of jobs in I_t that become urgent after J_i and before $J_{i'}$ are admitted to $\mathrm{Q}_{\mathrm{work}}$. Note that $\mathrm{join}(P) = \mathrm{join}(J_0, J_{k+1})$.

Following the above discussion of P, the total size of urgent jobs that Slow-D completes during P is at least $p(J_k) + \mathrm{join}(J_k, J_{k+1})$. To prove Lemma 2, if suffices to show that $p(J_k) + \mathrm{join}(J_k, J_{k+1})$ is at least $\frac{1}{4}(\mathrm{join}(P) + |P'| \times T)$. We prove this via an inductive argument whose statement is described by Lemma 5.

Lemma 5. *For any urgent period $P = [S_P, E_P]$ and its secured interval P', we have that* (1) $p(J_i) \geq$ *join*$(J_0, J_i) + (L_i - S_P) \times T$, *for* $i = 0, 1, \ldots, k$, *and* (2) *join*$(J_0, J_k) + |P'| \times T \leq 4 \times p(J_k) + 3 \times$ *join*(J_k, J_{k+1}).

Proof. We prove Statement (1) by induction. For $i = 0$, $p(J_0) = 0$, $\mathrm{join}(J_0, J_i)$, and $(L_0 - S_P)$ all equal to zero. Assume the claim is true for i. For $i+1$ (s.t. $i+1 \leq k$),

$$\begin{aligned}
&\mathrm{join}(J_0, J_{i+1}) + (L_{i+1} - S_P) \times T \\
&= \mathrm{join}(J_0, J_i) + \mathrm{join}(J_i, J_{i+1}) + ((L_{i+1} - L_i) + (L_i - S_P)) \times T \\
&\leq p(J_i) + \mathrm{join}(J_i, J_{i+1}) + (L_{i+1} - L_i) \times T \qquad \text{(by induction)} \\
&\leq 2 \times (p(J_i) + \mathrm{join}(J_i, J_{i+1})) < p(J_{i+1}) \ .
\end{aligned}$$

The second last step follows as the maximum work Slow-D can process during $[L_i, L_{i+1}]$ is at most $(p(J_i) + \mathrm{join}(J_i, J_{i+1}))$. The final step follows as J_{i+1} is admitted from $\mathrm{Q}_{\mathrm{wait}}$.

Before proving Statement (2), we observe the following bounds of J^*:

$$(d(J^*) - E_p) \times T \leq p(J^*) \leq 2(p(J_k) + \mathrm{join}(J_k, J_{k+1})) \ .$$

The first inequality follows from Lemma 4. If J^* is admitted to $\mathrm{Q}_{\mathrm{work}}$, then $J^* = J_i$ for some $i \leq k$; otherwise, $p(J^*) \leq 2(p(J_i) + \mathrm{join}(J_i, J_{i+1}))$ for some $i \leq k$. In both cases, $p(J^*) \leq 2(p(J_k) + \mathrm{join}(J_k, J_{k+1}))$. Thus,

$$\begin{aligned}
\mathrm{join}(J_0, J_k) + |P'| \times T &= \mathrm{join}(J_0, J_k) + (E_P - S_P) \times T + \max\{d(J^*) - E_P, 0\} \times T \\
&\leq \mathrm{join}(J_0, J_k) + (L_k - S_P) \times T + (E_P - L_k) \times T + p(J^*) \\
&\leq p(J_k) + (E_P - L_k) \times T + p(J^*) \qquad (\text{by (1)}) \\
&\leq p(J_k) + p(J_k) + \mathrm{join}(J_k, J_{k+1}) + p(J^*) \\
&\leq 4 \times p(J_k) + 3 \times \mathrm{join}(J_k, J_{k+1}) \ .
\end{aligned}$$

□

4 Minimizing Weighted Flow Time Plus Energy

We first consider the problem of minimizing the total fractional weighted flow time plus energy on a variable speed processor, in the bounded speed setting. Let $w_a(t)$ and $w_o(t)$ denote the total fractional weight of jobs in the online algorithm and some fixed optimum schedule at time t. Consider the algorithm that works at speed $s_a(t) = \min\{w_a(t)^{1/\alpha}, T\}$ and schedules jobs using HDF. We prove Theorem 5, which matches the guarantee of [4] up to lower order terms, who showed it for the case of $T = \infty$. Then by the same technique in [4], we can use this result to obtain a competitive algorithm for total (integral) weighted flow time plus energy using a processor with maximum speed $(1+\epsilon)T$ (recall that the speed augmentation is necessary to obtain a constant guarantee for this measure). We begin with a useful lemma relating the total fractional weight of jobs under both online and any offline algorithm.

Lemma 6. *For any offline algorithm, at any time t we have that $w_a(t) - w_o(t) \leq T^\alpha$.*

Proof. Clearly the result holds if $w_a(t) \leq T^\alpha$, and hence we consider the case when $w_a(t) \geq T^\alpha$. Thus at the current time t, the speed $s_a(t) = T$. Let t_0 be the latest time before t such that the speed just before t_0 (we denote this time by t_0^-) is less than T. Consider the set S of jobs that arrive during $[t_0, t]$ and let $w(S)$ denote their total weight. Let $\tilde{w}$ denote the amount of fractional weight completed by the offline algorithm by time t restricted to the jobs in S. As our algorithm always schedules the highest density job, the amount of fractional weight completed by our algorithm during $[t_0, t]$ when considered over all possible jobs (and not just jobs in S) is at least $\tilde{w}$. Thus it follows that $w_a(t) - w_a(t_0^-) \leq w(S) - \tilde{w} \leq w_o(t)$. This implies that $w_a(t) - w_o(t) \leq w_a(t_0^-)$ which is at most T^α since $s_a(t_0^-) < T$. □

Theorem 5. *The algorithm described above is $2\alpha/(\alpha - (\alpha-1)^{1-1/(\alpha-1)}) = ((2 + o(1))\alpha/\ln\alpha)$-competitive with respect to fractional weighted flow time plus energy.*

Proof. For notational ease, we will drop the time t from the notation, since all variables are understood to be functions of t. We will show that there is a potential function Φ, such that the value of the function is 0 at the beginning and at end of the schedule, never increases upon the release of a job, and satisfies the condition $\frac{d\Phi}{dt} \leq c(w_o + s_o^\alpha) - (w_a + s_a^\alpha)$ for some $c \geq 1$. As observed by [4], this proves that the algorithm is c-competitive for total fractional weighted flow plus energy. In fact as $w_a \geq s_a^\alpha$ for our algorithm, it suffices to show that

$$\frac{d\Phi}{dt} \leq c(w_o + s_o^\alpha) - 2w_a \ . \tag{1}$$

Consider the potential function $\Phi = \frac{\eta}{(\beta+1)} \int_0^\infty \left(w_a(h)^{\beta+1} - (\beta+1)w_a(h)^\beta w_o(h)\right) dh$, where $\beta = 1 - 1/\alpha$, and $w_a(h)$ and $w_o(h)$ are the total fractional weight of active jobs that have an inverse density (defined as the size of a job divided by its weight) of at least h under our algorithm and for some fixed optimum schedule respectively. The constant η will be chosen appropriately later.

That the potential function does not increase upon job arrival follows from Lemma 10 in [4] which implies that the content of the integral $w_a(h)^{\beta+1} - (\beta+1)w_a(h)^\beta w_o(h)$ never increases when both $w_a(h)$ and $w_o(h)$ are increased by the same amount. Thus

we analyze the case when a job is being executed. The analysis starts similar to [4]. We let m_a and m_o be the inverse density of the job being executed by our algorithm and the optimum schedule at the current time. Then we have

$$\begin{aligned}
\frac{d\Phi}{dt} &= -\eta\Big(\int_0^{\infty} (w_a(h)^{\beta} - \beta w_a(h)^{\beta-1} w_o(h)) \frac{dw_a(h)}{dt} - w_a(h)^{\beta} \frac{dw_o}{dt} dh\Big) \\
&= -\eta\Big(\int_0^{m_a} (w_a(h)^{\beta} - \beta w_a(h)^{\beta-1} w_o(h)) \frac{s_a}{m_a} dh\Big) + \eta\Big(\int_0^{m_o} w_a(h)^{\beta} \frac{s_o}{m_o} dh\Big) \\
&\leq -\eta(w_a^{\beta} s_a - \beta w_a^{\beta-1} w_o s_a) + \eta \int_0^{m_o} w_a^{\beta} \frac{s_o}{m_o} dh\Big) \\
&= -\eta w_a^{\beta} s_a + \eta\beta w_a^{\beta-1} w_o s_a + \eta w_a^{\beta} s_o \\
&\leq -\eta w_a^{\beta} s_a + \eta\beta w_a^{\beta-1} w_o s_a + \eta(\mu\beta w_a + \beta s_o^{\alpha}) \ . \qquad (2)
\end{aligned}$$

The second step follows from the HDF nature of our algorithm. The final step follows by applying Young's inequality with $a = w_a^{\beta}$, $b = s_0$, $p = 1/\beta$, and $q = \alpha$, and $\mu = (\alpha-1)^{-1/(\alpha-1)}$, which gives $w_a^{\beta} s_o \leq \mu\beta w_a + \left(\frac{1}{\mu}\right)^{\alpha\beta} \left(\frac{s_o^{\alpha}}{\alpha}\right) = \mu\beta w_a + \beta s_o^{\alpha}$.

We now show that (1) holds. We divide the analysis in three different cases. In each case we show that $d\Phi/dt \leq -\eta(1-\mu\beta)w_a + \eta(w_0 + s_o^{\alpha})$. Setting $\eta = 2/(1-\mu\beta)$, this implies a competitive ratio of $2/(1-\mu\beta)$, which by substituting the values of μ and β gives our claimed guarantee. The first case is when $w_o \geq w_a \geq T^{\alpha}$. Observe that in this case $s_a = T$. Starting with (2),

$$\begin{aligned}
\frac{d\Phi}{dt} &\leq \eta(-w_a^{\beta} T + \beta w_a^{\beta-1} w_o T + \mu\beta w_a + \beta s_o^{\alpha}) \\
&< \eta(-w_a^{\beta} T + w_a^{\beta-1} w_o T + \mu\beta w_a + s_o^{\alpha}) \qquad (\text{as } \beta < 1) \\
&= \eta(w_a^{\beta-1}(w_o - w_a)T + \mu\beta w_a + s_o^{\alpha}) \\
&\leq \eta(w_a^{\beta-1}(w_o - w_a) w_a^{1/\alpha} + \mu\beta w_a + s_o^{\alpha}) \qquad (\text{as } w_a > T^{\alpha} \text{ and } w_o > w_a) \\
&= -\eta(1-\mu\beta)w_a + \eta w_o + \eta s_o^{\alpha} \ .
\end{aligned}$$

The second case is when $w_a \geq T^{\alpha}$ and $w_a > w_o$. Again, $s_a = T$. 79

$$\begin{aligned}
\frac{d\Phi}{dt} &\leq \eta(-w_a^{\beta} s_a + \beta w_a^{\beta-1} w_o s_a + \mu\beta w_a + \beta s_o^{\alpha}) \\
&= -\eta w_a^{\beta-1}(w_a - \beta w_o)T + \eta\mu\beta\, w_a + \eta s_o^{\alpha} \qquad (\text{as } \beta < 1) \\
&\leq -\eta w_a^{\beta-1}(w_a - \beta w_o)(w_a - w_o)^{1/\alpha} + \eta\mu\beta w_a + \eta s_o^{\alpha} \quad (\text{Lemma 6 and } (w_a - \beta w_o) > 0) \\
&\leq -\eta w_a^{\beta-1} w_a^{1-\beta}(w_a - w_o)^{\beta}(w_a - w_o)^{1/\alpha} + \eta\mu\beta w_a + \eta s_o^{\alpha} \\
&= -\eta(1-\mu\beta)w_a + \eta w_o + \eta s_o^{\alpha} \ .
\end{aligned}$$

The third step follows as $(w_a - w_o) < T^{\alpha}$ by Lemma 6. The fourth step follows by using that fact that $(1-\beta x) \geq (1-x)^{\beta}$ for any $0 \leq \beta \leq 1$ and $0 \leq x < 1$, which implies that $(w_a - \beta w_o) \geq w_a^{1-\beta}(w_a - w_o)^{\beta}$.

Finally we consider the case that $w_a < T$. Here $s_a = w_a^{1/\alpha}$, and this is exactly the case handled by [4]. We reprove it here for completeness. $\frac{d\Phi}{dt} \leq -\eta w_a^{\beta} s_a + \eta\beta w_a^{\beta-1} w_o s_a + \eta\beta\mu w_a + \eta\beta s_o^{\alpha} = -\eta(1-\beta\mu)w_a + \eta\beta w_o + \eta\beta s_o^{\alpha} \leq -\eta(1-\beta\mu)w_a + \eta w_o + \eta s_o^{\alpha}$. Thus the result follows. □

References

1. http://www-03.ibm.com/chips/power/powerpc/newsletter/sep2004/technical2.html
2. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 621–633. Springer, Heidelberg (2006)
3. Bansal, N., Kimbrel, T., Pruhs, K.: Dynamic speed scaling to manage energy and temperature. Journal of the ACM 51(1) (2007)
4. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. In: Proc. SODA, pp. 805–813 (2007)
5. Bansal, N., Chan, H.L.: Weighted flow time does not have O(1) competitive algorithms (manuscript)
6. Bansal, N., Dhamdhere, K.: Minimizing weighted flow time. In: Proc. SODA, pp. 508–516 (2003)
7. Baruah, S., Koren, G., Mishra, B., Raghunathan, A., Rosier, L., Shasha, D.: On-line scheduling in the presence of overload. In: Proc. FOCS, pp. 100–110 (1991)
8. Becchetti, L., Leonardi, S., Marchetti-Spaccamela, A., Pruhs, K.: Online Weighted Flow Time and Deadline Scheduling. In: Proc. RANDOM-APPROX, pp. 36–47 (2001)
9. Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J.D., Zyuban, V., Gupta, M., Cook, P.W.: Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. IEEE Micro. 20(6), 26–44 (2000)
10. Chan, H.L., Chan, W.T., Lam, T.W., Lee, L.K., Mak, K.S., Wong, P.: Energy efficient online deadline scheduling. In: Proc. SODA, pp. 795–804 (2007)
11. Chekuri, C., Khanna, S., Zhu, A.: Algorithms for minimizing weighted flow time. In: Proc. STOC, pp. 84–93 (2001)
12. Dertouzos, M.L.: Control robotics: the procedural control of physical processes. In: Proc. IFIP Congress, pp. 807–813 (1974)
13. Grunwald, D., Levis, P., Farkas, K.I., Morrey, C.B., Neufeld, M.: Policies for dynamic clock scheduling. In: Proc. OSDI, pp. 73–86 (2000)
14. Hardy, G.H., Littlewood, J.E., Polya, G.: Inequalities. Cambridge University Press, Cambridge (1952)
15. Irani, S., Pruhs, K.: Algorithmic problems in power management. SIGACT News (2005)
16. Koren, G., Shasha, D.: D^{over}: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. SIAM J. Comput. 24(2), 318–339 (1995)
17. Lam, T.W., To, K.K.: Performance Guarantee for Online Deadline Scheduling in the Presence of Overload. In: Proc. SODA, pp. 755–764 (2001)
18. Li, M., Liu, B.J., Yao, F.F.: Min-energy voltage allocations for tree-structured tasks. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 283–296. Springer, Heidelberg (2005)
19. Li, M., Yao, F.: An efficient algorithm for computing optimal discrete voltage schedules. SIAM J. Comput. 35(3), 658–671 (2005)
20. Mudge, T.: Power: A first-class architectural design constraint. Computer 34(4), 52–58 (2001)
21. Pillai, P., Shin, K.G.: Real-time dynamic voltage scaling for low-power embedded operating systems. In: Proc. SOSP, pp. 89–102 (2001)
22. Pruhs, K., Uthaisombut, P., Woeginger, G.: Getting the best response for your erg. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 14–25. Springer, Heidelberg (2004)
23. Weiser, M., Welch, B., Demers, A., Shenker, S.: Scheduling for reduced CPU energy. In: Proc. OSDI, pp. 13–23 (1994)
24. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proc. FOCS, pp. 374–382 (1995)