Speed Scaling for Weighted Flow Time

Nikhil Bansal* Kirk Pruhs† Cliff Stein‡

1 Introduction

In addition to the traditional goal of efficiently managing time and space, many computers now need to efficiently manage power usage. For example, Intel's SpeedStep and AMD's PowerNOW technologies allow the Windows XP operating system to dynamically change the speed of the processor to prolong battery life. In this setting, the operating system must not only have a *job selection policy* to determine which job to run, but also a *speed scaling* policy to determine the speed at which the job will be run. These policies must be online since the operating system does not in general have knowledge of the future. In current CMOS based processors, the speed satisfies the well known cube-root-rule, that the speed is approximately the cube root of the power [Mud01, BBS+00]. Thus, in this work, we make the standard generalization that the power is equal to speed to some power $\alpha \geq 1$, where one should think of α as being approximately 3 [YDS95, BKP04]. Energy is power integrated over time. The operating system is faced with a dual objective optimization problem as it both wants to conserve energy, and optimize some Quality of Service (QoS) measure of the resulting schedule.

By far the most commonly used QoS measure in the computer systems literature is *average response/flow time* or more generally *weighted average response/flow time*. The flow time F_i of a job i is the time lag between when a job is released to the system and when the system completes that job. Pruhs, Uthaisombut, and Woeginger [PUW04] studied the problem of optimizing total flow time ($\sum_i F_i$) subject to the constraint that the total energy does not exceed some bound, say the energy in the battery, and showed how to efficiently construct offline an optimal schedule for instances with unit work jobs. For unit work jobs, all job selection policies that favor a partially executed job in favor of an unexecuted job are equivalent. Thus the job selection policy is essentially irrelevant.

If there is an upper bound on energy used, then there is no $O(1)$-competitive online speed scaling policy for total flow time. To understand intuitively why this is the case, consider the situation when the first job arrives. The scheduler has to allocate a constant fraction of the total energy to this job; otherwise, the scheduler would not be $O(1)$-competitive in the case that no more jobs arrive. However, if many more jobs arrive in the future, then the scheduler has wasted a constant fraction of its energy on only one job. By iterating this process, one obtains a bound of $\omega(1)$ on the competitive ratio with respect to total flow time (See Appendix Section A for the details).

Albers and Fujiwara [AF06] proposed combining the dual objectives of energy and flow time into the single of objective of energy used plus total flow time. Optimizing a linear combination of energy and total flow time has the following natural interpretation. Suppose that the user specifies how much improvement in flow time, call this amount ρ, is necessary to justify spending one unit of energy. For example, the user might specify to the Windows XP operating system that he is willing to spend 1 erg of energy from the battery for a decrease of 3 micro-seconds in response time. Then the optimal schedule, from this user's perspective, is the schedule that optimizes $\rho = 3$ times the energy used plus the total flow time. By changing the units of either energy or time, one may assume without loss of generality that $\rho = 1$.

[PUW04] observe that in any locally-optimal normal schedule, each job i is run at a power proportional to the number of jobs that depend on i. Roughly speaking, *normal* means that no job completes exactly when another job is released, and a job j *depends* on a job i if delaying i would delay j. In the online setting, an obvious lower bound to the number of jobs that depend on the selected job is the number of *active* jobs, where an active job is one that has been released but has not yet completed. Thus Albers and Fujiwara [AF06] propose the natural online speed scaling algorithm that always runs at a power equal to the number of active jobs. They again only consider the case of unit work jobs. They do not actually analyze this natural algorithm, but rather analyze a batched variation, in which

*IBM T.J. Watson Research, P.O. Box 218, Yorktown Heights, NY. nikhil@us.ibm.com.

†Computer Science Department. University of Pittsburgh. kirk@cs.pitt.edu. Supported in part by NSF grants CNS-0325353, CCF-0448196, CCF-0514058 and IIS-0534531.

‡Department of IEOR, Columbia University, New York, NY. cliff@ieor.columbia.edu.

jobs that are released while the current batch is running are ignored until the current batch finishes. They show that this batched algorithm is $8.3e(\frac{3+\sqrt{5}}{2})^{\alpha}$-competitive with respect to the objective of total flow time plus energy. They also give a dynamic programming algorithm to compute the offline optimal schedule for unit work jobs.

The reason that both [PUW04] and [AF06] consider only unit work jobs is that it seems that the optimal schedule for arbitrary work jobs is quite difficult to characterize.

1.1 Our Results We give significantly stronger results for the problem of minimizing (weighted) flow time and energy. We both improve the algorithm and analysis in the special case (unit jobs, no weights) considered previously [AF06] and then we give algorithms for the more general problem with weights and arbitrary work jobs.

First, we show that the natural online speed scaling algorithm proposed in [AF06] is 4-competitive for unit work jobs. This guarantee is independent of α. In comparison, the competitive ratio $8.3e(\frac{3+\sqrt{5}}{2})^{\alpha}$ obtained in [AF06] is bit over 400 when the cube-root rule holds ($\alpha = 3$).

More importantly, we consider the case of arbitrary work jobs, and consider a much more general QoS measure: weighted flow. We assume that each job has a positive integer weight w_i. The weighted flow objective is then the weighted sum of flow times, $\sum_i w_i F_i$. Weighted flow generalizes both total flow time, and total/average stretch, which is another common QoS measure in the systems literature. The stretch/slow-down of a job is the flow time divided by the work of the job. Many server systems, such as operating systems and databases, have mechanisms that allow the user or the system to give different priorities to different jobs. For example, Unix has the `nice` command. In our setting, the weight of a job is indicative of the flow time versus energy trade-off for it. The user may be willing to spend more energy to reduce the flow time of a higher priority job, than that for lower priority jobs.

Our analysis consists of two steps. We first relax the objective to fractional weighted flow plus energy, instead of weighted flow plus energy. In the fractional weighted flow time measure, at each time step a job contributes its weight times the fraction of unfinished work to the objective (see Section 2 for details). In the second step we show how to modify our algorithm for fractional weighted flow plus energy to obtain results for weighted flow plus energy at the loss of a small factor in the competitive ratio. The main reason for this two step analysis is that fractional flow is substantially easier to analyze than total flow. For example, for a constant speed processor, computing the optimal weighted flow schedule is NP-hard [LLLK84], but the simple algorithm Highest Density First (HDF) is optimal for fractional weighted flow. HDF is the algorithm that always runs the active job with the maximum *density*, where the density of a job is the ratio of its weight to its work. HDF is still the optimal job selection policy for fractional weighted flow when speed scaling is allowed.

Our algorithm is a natural generalization of the algorithm proposed in [AF06]. We define the algorithm A to be the one that uses HDF for job selection, and always runs at a power equal to the fractional weight of the active jobs. In Section 4 we consider the case of unit-work unit-weight jobs. We show that algorithm A is 2-competitive with respect to the objective of fractional flow plus energy. As a corollary to this, we show that the algorithm B (proposed by [AF06]), that runs at power equal to the number of unfinished jobs is 4-competitive for total flow plus energy.

In Section 5 we consider jobs with arbitrary work and arbitrary weights. Let $\gamma = \max\left(2, \frac{2(\alpha-1)}{\alpha-(\alpha-1)^{1-1/(\alpha-1)}}\right)$. We show that algorithm A is γ-competitive with respect to the objective of fractional flow plus energy. For any $\alpha > 1$, the value of $\gamma \leq \max(2, 2\alpha - 2)$. For large values of α, γ is approximately $\alpha/\ln\alpha$ (ignoring lower order terms) and for $\alpha = 3$, $\gamma \approx 2.52$. We then use A to define an algorithm C_ϵ, which is parameterized by $\epsilon > 0$, and is $\gamma\mu_\epsilon$-competitive with respect to the objective of total weighted flow plus energy, where $\mu_\epsilon = \max(1 + \frac{1}{\epsilon}, (1+\epsilon)^{\alpha})$. When the cube-root rule holds, by picking $\epsilon = .463$, μ_ϵ is about 3.15, and the competitive ratio $\gamma\mu_\epsilon$ is a bit less than 8. For large values of α, picking $\epsilon \approx \ln\alpha/\alpha$ implies that C_ϵ is approximately $\alpha^2/\ln^2\alpha$ competitive.

The analysis in [AF06] was based on comparing the online schedule directly to the optimal schedule. However, even for the case of unit work jobs, the optimal schedule can be rather complicated [PUW04, AF06]. Our analyses of algorithm A are based on the use of a potential function, and do not require us to understand the structure of the optimal schedule. There are two advantages to this approach. First, our analysis is simpler and tighter than the analysis in [AF06] in the case of unit-work unit-weight jobs. Second, we can extend our analysis to the case of arbitrary work and arbitrary weight jobs. We give an overview of our potential function analysis technique in Section 3.

Further, our results also give a way to compute the schedule that optimizes weighted flow subject to a constraint E on the energy used. It is not too hard to see that if we trace out all possible optimal weighted flow schedules for all energy bounds E, and we traced out all

optimal weighted flow plus ρ times energy schedules for all possible factors ρ, that the resulting schedules are the same. That is, a schedule S is an optimal weighted flow schedule for some energy bound E if and only if it is an optimal weighted flow plus ρ time energy schedule for some factor ρ. Thus, by performing a binary search over the possible ρ, and applying our algorithm C, one can find an $O(1)$-approximate weighted flow schedule subject to an energy bound of E.

1.2 Related Results Theoretical investigations of speed scaling algorithms were initiated by Yao, Demers, and Shenker [YDS95], who considered the problem of minimizing energy usage when each task has to be finished on one machine by its deadline. [YDS95] give an optimal offline greedy algorithm, and an $O(1)$-competitive online algorithm AVR. The online algorithm Optimal Available (OA), proposed in [YDS95], was shown to be $O(1)$-competitive in [BKP04] using a potential function analysis. OA runs at the speed that would be optimal, given the current state and given that no more jobs arrive in the future. The speed scaling component of our algorithm A is similar in spirit as it can be described as: run at a constant factor $\alpha-1$ times the optimal speed given the current state, and given that no more jobs arrive in the future. The potential functions that we use to analyze A are reminiscent, but certainly not the same, as the one used in [BKP04] to analyze OA.

Since the publication of [YDS95], most of the results in the literature focus on deadline feasibility as the measure for the quality of the schedule. However, in general computational settings, most processes do not have natural deadlines associated with them. As evidence of this fact, observe that neither Linux nor Microsoft Windows use deadline based schedulers. In general, algorithm design and analysis are significantly more difficult in speed scaling problems than in the corresponding scheduling problem on a fixed speed processor. But deadlines help constrain how the energy can be distributed throughout the schedule, thus making scheduling problems with deadlines more amenable to analysis. Given the number of papers that now have been published on speed scaling problems with deadlines, we will not survey these results here, but refer the reader to a relatively recent survey by Irani and Pruhs[IP05].

There have been several papers in the literature on speed scaling with the makespan objective. [PvSU05] give a poly-log approximation algorithm for makespan on identical machines with precedence constraints give a bound on the available energy. [Bun06] gives an efficient algorithm to compute all Pareto optimal schedules for tasks on one machine with release dates, and for unit-work tasks on multiple machines with release dates.

[Bun06] also shows that even for unit-work jobs, optimal flow time schedules can not generally be expressed with the standard four arithmetic operations, and the extraction of roots.

2 Definitions

An instance consists of n jobs, where job i has a release time r_i, a positive work y_i, and a positive integer weight $\overline{w}_i$. The density of job i is $\frac{\overline{w}_i}{y_i}$ and the inverse density is $\frac{y_i}{\overline{w}_i}$. We assume without loss of generality that $r_1 \leq r_2 \leq \ldots \leq r_n$. An online scheduler is not aware of job i until time r_i, and, at time r_i, learns y_i and weight $\overline{w}_i$. For each time, a schedule specifies a job to be run and a speed at which the job is run. A job i completes once y_i units of work have been performed on i. The speed is the rate at which work is completed; a job with work y run at a constant speed s completes in $\frac{y}{s}$ seconds. The power consumed when running at speed s is s^α, where we assume that $\alpha \geq 1$. The energy used is power integrated over time. We assume that preemption is allowed, that is, a job may be suspended and later restarted from the point of suspension. A job is *active* at time t if it has been released but not completed at time t.

As an algorithm runs, we will keep track of the total weight of jobs active at time t. There are actually two different notions of a job's weight. When we just use weight, we mean the original weight $\overline{w}_i$ of the job. When we say *fractional weight* of a job i at time t, we mean the weight of the job times the percentage of work on the job that has not yet been finished. We will use an overbar for weight and omit it for fractional weight.

Let X be an arbitrary algorithm. Let $\overline{w}_x(t)$ denote the weight of jobs active at time t for algorithm X. (Note that we make algorithms lower case in subscripts for typographic reasons.) Let $w_x(t)$ denote the fractional weight of the active jobs at time t for algorithm X. Let $s_x(t)$ be the speed at time t for algorithm x, and let $p_x(t) = (s_x(t))^\alpha$ be the power consumed at time t by algorithm X. Let $E_x(t) = \int_{k=0}^{t} p_x(k)dk$ be the energy spent up until time t by algorithm A.

Just as we defined weight and fractional weight, we can define weighted flow time and fractional weighted flow time analogously. We use the well-known observation that the total weighted flow time is the total weight of the set of active jobs, integrated over time. Let $W_x(t) = \int_{k=0}^{t} w_x(k)dk$ be the fractional weighted flow up until time t for algorithm X. Let $\overline{W}_x(t) = \int_{k=0}^{t} \overline{w}_x(k)dk$ be the weighted flow up until time t for algorithm X. Our objective function combines flow and energy and we let $G_x(t) = W_x(t) + E_x(t)$ be the fractional weighted flow and energy up until time

t for algorithm X, and $\overline{G}_x(t) = \overline{W}_x(t) + E_x(t)$ be the weighted flow and energy up until time t for algorithm X. Let $E_x = E_x(\infty)$, $W_x = W_x(\infty)$, $\overline{W_x} = \overline{W}_x(\infty)$, $G_x = G_x(\infty)$ and $\overline{G_x} = \overline{G}_x(\infty)$ be the energy, fractional weighted flow, weighted flow, fractional weighted flow plus energy, and weighted flow plus energy, respectively, for algorithm X. We use Opt to denote the offline adversary, and subscript a variable by "o" to denote the value of a variable for the adversary. So W_o is the fractional weighted flow for the adversary.

3 Amortized Local Competitiveness

A common notion to measure an on-line scheduling algorithm is *local competitiveness*, meaning roughly that the algorithm competitive at all times during the execution. Local competitiveness is generally not achievable in speed scaling problems because the adversary may spend essentially all of its energy in some small period of time, making it impossible for any online algorithm to be locally competitive at that time. Thus, we will analyze our algorithms using *amortized local competitiveness*, which we now define. Let X be an arbitrary online scheduling algorithm, and H an arbitrary objective function, Let $\frac{dH(t)}{dt}$ be the rate of increase of the objective H at time t. The online algorithm X is *amortized locally γ-competitive with potential function $\Phi(t)$ for objective function H* if the following two conditions hold:

Boundary Condition: Φ is initially 0, and and finally nonnegative. That is, $\Phi(0) = 0$, and there exists some time t' such that for all $t \geq t'$ it is the case that $\Phi(t) \geq 0$.

General Condition: For all times t,

$$\frac{dH_x(t)}{dt} - \gamma \frac{dH_o(t)}{dt} + \frac{d\Phi(t)}{dt} \leq 0 \tag{3.1}$$

We break the general condition into three cases:

Running Condition: For all times t when no job arrives,

$$\frac{dH_x(t)}{dt} - \gamma \frac{dH_o(t)}{dt} + \frac{d\Phi(t)}{dt} \leq 0 \tag{3.2}$$

Job Arrival Condition: Φ does not increase when a new job arrives.

Completion Condition: Φ does not increase when either the online algorithm or the adversary complete a job.

Observe that when $\Phi(t)$ is identically zero, we have ordinary local competitiveness. It is well known that amortized local γ-competitiveness implies that when the algorithm completes, the total cost of the online algorithm is at most γ times the total cost of the optimal offline algorithm.

LEMMA 3.1. *If online algorithm X is* amortized locally γ-competitive with potential function $\Phi(t)$ for objective function H, *then* $H_x \leq \gamma H_o$.

Proof. Let $t_1, \ldots, t_{3n}$ be the events that either a job is released, the online algorithm x completes a job, or the adversary completes a job. Let $\Delta(\Phi(t_i))$ denote the change in potential in response to event t_i. Let $t_0 = 0$ and $r_{3n+1} = +\infty$. Integrating equation 3.1 over time, we get that

$$H_x + \sum_{i=1}^{3n+1} \Delta(\Phi(t_i)) \leq \gamma H_o$$

By the job arrival condition, and the completion condition, we can conclude that $H_x + \Phi(\infty) - \Phi(0) \leq \gamma H_o$, and finally, by the boundary condition, we can conclude that $H_x \leq \gamma H_o$.

Now consider the case that the objective function is G, the fractional weighted flow plus energy. Then $\frac{dG(t)}{dt} = w(t) + p(t) = w(t) + s(t)^\alpha$, and equation 3.1 is equivalent to:

$$w_x(t) + s_x(t)^\alpha - \gamma(w_o(t) + s_o(t)^\alpha) + \frac{d\Phi(t)}{dt} \leq 0$$

For our purposes, we will always consider the algorithm A, where $s_a(t)^\alpha = w_a(t)$. Thus the above equation is equivalent to:

$$2w_a(t) - \gamma(w_o(t) + s_o(t)^\alpha) + \frac{d\Phi(t)}{dt} \leq 0\ . \tag{3.3}$$

To compute our competitive ratio, we will find the minimum γ which satisfies equation 3.3.

4 Unit Work and Unit Weight Jobs

In this section we restrict ourselves to jobs with unit work and unit weight. We first show that the speed scaling algorithm A, where $s_a(t) = w_a(t)^{1/\alpha}$ is 2-competitive for the objective function of fractional flow time plus energy. We then show how to modify A to obtain a 4-competitive algorithm for the objective function (integral) flow time plus energy.

THEOREM 4.1. *Assume that all jobs have unit work and unit weight. The speed scaling algorithm A, where $s_a(t) = w_a(t)^{1/\alpha}$ is 2-competitive with respect to the objective G of fractional flow plus energy.*

Proof. We prove that algorithm A is amortized locally 2-competitive using the potential function

$$\Phi(t) = \frac{2\alpha}{(\beta+1)} \left(\max(0, w_a(t) - w_o(t))\right)^{\beta+1}$$

where $\beta = (\alpha-1)/\alpha$.

We first need to verify the boundary condition. Clearly $\Phi(0) = 0$, as $w_a(0) = w_o(0)$, $\Phi(t)$ is always non-negative. Φ satisfies the job completion condition since the fractional weight of a job approaches zero continuously as the job nears completion and there is no discontinuity in $w_a(t)$ or $w_o(t)$ when a job completes. Φ satisfies the job arrival condition since both $w_a(t)$ and $w_o(t)$ increase simultaneously by 1 when a new job arrives

We are left to establish the running condition. We now break the argument into two cases. In the first case assume that that $w_a(t) < w_o(t)$. This case is simpler, since the offline adversary has large fractional weight. Here $\Phi(t) = 0$ and $\frac{d\Phi(t)}{dt} = 0$ by the definition of Φ. Thus inequality (3.3) is satisfied by setting γ to 2.

We now turn to the interesting case that $w_a(t) \geq w_o(t)$. For notational ease, we will drop the time t from the notation, since all variables are understood to be functions of t. We consider $d\Phi/dt$.

$$\begin{aligned} \frac{d\Phi(t)}{dt} &= \frac{2\alpha}{(\beta+1)} \frac{d\left((w_a(t) - w_o(t))^{\beta+1}\right)}{dt} \\ &= 2\alpha(w_a - w_o)^{\beta} \frac{d(w_a - w_o)}{dt} \end{aligned} \tag{4.4}$$

Since jobs have unit density, the rate at which the fractional weight decreases is exactly the rate at which unfinished work decreases, which is just the speed of the algorithm. Thus $\frac{dw}{dt} = -s$. Moreover as $s_a(t) = w_a(t)^{1/\alpha}$, by the definition of A, equation 4.4 can be written as

$$\begin{aligned} \frac{d\Phi(t)}{dt} &= -2\alpha(w_a - w_o)^{\beta}(s_a - s_o) \\ &= -2\alpha(w_a - w_o)^{\beta}(w_a^{1/\alpha} - s_o) \end{aligned} \tag{4.5}$$

Since $w_a \geq w_a - w_o$, it follows that $-w_a^{1/\alpha} \leq -(w_a - w_o)^{1/\alpha}$ and as $\beta + 1/\alpha = 1$ by definition of β, equation 4.5 implies that

$$\frac{d\Phi(t)}{dt} \leq -2\alpha(w_a - w_o) + 2\alpha(w_a - w_o)^{\beta} s_o \tag{4.6}$$

Applying Young's inequality (c.f. Corollary 6.1) with $\mu = 1$, $a = s_0$, $p = \alpha$, $b = (w_a - w_o)^{\beta}$, and $q = \frac{1}{\beta}$, we obtain that $(w_a - w_o)^{\beta} s_o \leq \beta(w_a - w_o) + s_o^{\alpha}/\alpha$. Thus (4.6) can be written as

$$\begin{aligned} \frac{d\Phi(t)}{dt} &\leq -2\alpha(w_a - w_o) + \\ &\quad 2\alpha\beta(w_a - w_o) + 2s_o^{\alpha} \\ &= -2(w_a - w_o) + 2s_o^{\alpha} \end{aligned} \tag{4.7}$$

Plugging (4.7) into 3.3 and solving for γ, we obtain a bound on the competitive ratio of:

$$\gamma \leq \frac{2w_a + \frac{d\Phi}{dt}}{w_o + s_o^{\alpha}} \tag{4.8}$$

$$\leq \frac{2w_a + (-2w_a + 2w_o + 2s_o^{\alpha})}{w_o + s_o^{\alpha}} \tag{4.9}$$

$$\leq \frac{2w_o + 2s_o^{\alpha}}{w_o + s_o^{\alpha}} \tag{4.10}$$

$$= 2 \tag{4.11}$$

Note that for any instance, there is always one job that algorithm A never completes. This does not contradict our analysis since the fractional weight for A will be geometrically decreasing at the end of the schedule.

We now modify the algorithm A to handle integral flow time. Consider the algorithm B that at all times gives preference to the (there is at most one) partially finished job, and works at power equal to the (integral) weight of unfinished jobs. That is $s_b(t) = \overline{w}_b^{1/\alpha}$. To analyze algorithm B, we relate B to algorithm A. For the optimum algorithm we use its total fractional flow time plus energy as a lower bound to the integral objective. We begin by observing that under any algorithm each job incurs a flow time plus energy of at least 1.

LEMMA 4.1. *If all jobs have unit work and unit weight, then for any instance with n jobs, $\overline{G}_o \geq n$.*

Proof. Suppose a job has flow time f, then by convexity of the power function its energy is minimized if it is run at speed $1/f$ throughout, and hence it uses at least $f \cdot (1/f)^{\alpha} = (1/f)^{\alpha-1}$ amount of energy. It suffices to show that $f + (1/f)^{\alpha-1} \geq 1$ for any $f > 0$. Clearly, this is true if $f \geq 1$. If $f \leq 1$, then $(1/f) \geq 1$ and hence $(1/f)^{\alpha-1} \geq 1$ as $(\alpha - 1) \geq 0$.

LEMMA 4.2. *Assume that all jobs have unit work and unit weight. The algorithm B, where $s_b(t) = \overline{w}(t)^{1/\alpha}$, is 4-competitive with respect to the objective $\overline{G}$ of total flow plus energy.*

Proof. Consider B and A running on the same input instance. At any time t, the fractional weight $w_b(t)$ under B never exceeds that under A, since if they were

ever equal, then algorithm B must run at least as fast as algorithm A. As B has at most one partially executed job at any time this implies that $\overline{w}_b(t) \leq w_a(t) + 1$. Since B runs at speed at least 1 when it is not idling, it follows that $\overline{W}_b \leq W_a + n$. Since $E_b = \overline{W}_b$ and $E_a = W_a$ it follows that $G_b = \overline{W}_b + E_b = 2\overline{W}_b \leq 2W_a + 2n = G_a + 2n$. By Theorem 4.1, we have that $G_a \leq 2G_o$ and since $G_o \leq \overline{G}_o$, it follows that $G_b \leq G_a + 2n \leq 2\overline{G}_o + 2n \leq 4\overline{G}_o$. The last step follows as $\overline{G}_o \geq n$ by Lemma 4.1.

Before we proceed to the general case in the next section, we wish to point out (details deferred to the full version) that the above analysis works identically for the slightly more general case where jobs have unit density (i.e. when the weight of a job is equal to its work). Thus we can generalize Lemma 4.2 to obtain

COROLLARY 4.1. *Assume that all jobs have unit density. The algorithm B, where $s_b(t) = \overline{w}(t)^{1/\alpha}$, is 4-competitive with respect to the objective $\overline{G}$ of total flow plus energy.*

5 Arbitrary Size and Weight Jobs

In this section we consider jobs with arbitrary work and arbitrary weight. We first show that the algorithm A is $\gamma = \max(2, \frac{2(\alpha-1)}{\alpha-(\alpha-1)^{1-1/(\alpha-1)}})$ competitive with respect to fractional weighted flow plus energy. In particular, $\gamma = 2$ for $\alpha \in [1, 2]$ and $\gamma = 2(\alpha-1)/(\alpha - (\alpha-1)^{1-1/(\alpha-1)})$. For $\alpha = 3$, γ is about 2.52 and varies asymptotically as $\alpha/\ln\alpha$ for large values of α

Later we show how to use A to obtain an algorithm for (integral) weighted flow time plus energy. This algorithm will be parameterized by ϵ and denoted as C_ϵ. The competitive ratio of C_ϵ will be $\gamma\mu_\epsilon$ where $\mu_\epsilon = \max(1 + \frac{1}{\epsilon}, (1+\epsilon)^\alpha)$. Choosing ϵ optimally as a function of α, the competitive ratio of C_ϵ will be approximately $\alpha^2/\ln^2\alpha$ (ignoring lower order terms).

THEOREM 5.1. *The speed scaling algorithm A, where the job selection policy is HDF and $s_a(t) = w(t)^{1/\alpha}$, is $\gamma = \max\left(2, \frac{2(\alpha-1)}{\alpha-(\alpha-1)^{1-1/(\alpha-1)}}\right)$ competitive with respect to the objective G of fractional weighted flow plus energy.*

Proof. For technical reasons it will be very convenient to work with inverse density which is defined as the ratio of the work of a job divided by its weight. In this terminology the algorithm HDF is the one that works on the job with the least inverse density at any time.

Let $w_a(h)$ and $w_o(h)$ be functions of time t denoting the *total fractional weight of the active jobs which have an inverse density of at least h*, for algorithm A and some fixed optimum algorithm *Opt* respectively. Note that for $h = 0$, these terms simply correspond to the total fractional weight at time t. We will prove that A is amortized locally γ-competitive using the potential function $\Phi(t)$ defined by:

$$\eta \int_{h=0}^{\infty} \left(w_a(h)^\beta (w_a(h) - (\beta+1)w_o(h))\right) dh \tag{5.12}$$

where $\beta = (\alpha-1)/\alpha$, and η is some constant that we will set later.

That Φ satisfies the boundary condition follows easily since $w_a(h) = w_o(h) = 0$ for all values of h at time $t = 0$ and as time approaches infinity. Similarly, Φ satisfies the job completion condition since the fractional weight of a job approaches zero as the job nears completion, and there are no discontinuities.

Now consider the arrival condition (which is somewhat non-trivial in this case). Suppose a job i with inverse density h_i and weight w_i arrives at time t. If $h \leq h_i$ then both $w_a(h)$ and $w_o(h)$ increase simultaneously by w_i. If $h > h_i$ then both $w_a(h)$ and $w_o(h)$ remain unchanged. Thus after the arrival of job i the change in the potential function is

$$\eta \int_{h=0}^{h_i} \big[(w_a(h)+w_i)^\beta (w_a(h) - (\beta+1)w_o(h) - \beta w_i) - w_a(h)^\beta (w_a(h) - (\beta+1)w_o(h))\big] dh \tag{5.13}$$

The fact that each summand

$$(w_a(h)+w_i)^\beta (w_a(h) - (\beta+1)w_o(h) - \beta w_i) - w_a(h)^\beta (w_a(h) - (\beta+1)w_o(h)) \tag{5.14}$$

in the integral above is not positive follows from Lemma 6.1 by setting $q = w_a(h), r = w_o(h), \delta = w_i$ and $\mu = \beta + 1$. Thus the arrival condition holds.

We now consider the running condition. Let m_a and m_o denote the minimum inverse density of a job that is alive under A and *Opt* at time t, respectively. Assume algorithm A is running job i with inverse density $h_i = m_a$. Then for $h \leq m_a$, let us consider the rate at which $w_a(h)$ changes with t. The remaining work decreases at rate $-s_a$ and hence the fractional weight decreases at rate $-s_a \cdot (w_i/y_i)$ where y_i is the (original) work of this job. Thus,

$$\frac{dw_a(h)}{dt} = -s_a \cdot \frac{w_i}{y_i} = -\frac{s_a}{m_a}$$

and if $h > m_a$ then $\frac{dw_a(h)}{dt} = 0$. Similarly, $\frac{dw_o(h)}{dt} = -\frac{s_o}{m_o}$ if $h \leq m_o$ and is 0 otherwise. We now

evaluate $\frac{d\Phi}{dt}$ to be:

$$\begin{aligned}(5.15)\quad \eta \int_{h=0}^{\infty} \Bigg(& \frac{d(w_a(h)^{\beta+1})}{dt} - \\ & (\beta+1)\frac{d((w_a(h)^{\beta} w_o(h)))}{dt} \Bigg) dh \\ = \eta(\beta+1) \Bigg[& \int_{h=0}^{\infty} w_a(h)^{\beta} \frac{dw_a(h)}{dt} dh \\ & - \int_{h=0}^{\infty} w_a(h)^{\beta} \frac{dw_o(h)}{dt} dh \\ & - \beta \int_{h=0}^{\infty} w_a(h)^{\beta-1} w_o(h) \frac{dw_a(h)}{dt} dh \Bigg]\end{aligned}$$

We now focus on the first integral in equation 5.15. Since A works on job with minimum inverse density m_a, there is non-zero contribution only when $h \in [0, m_a]$. Further, for $h \in [0, m_a)$, it is the case that $w_a(h) = w_a(0) = w_a$. Thus,

$$\begin{aligned}(5.16)\quad & \int_{h=0}^{\infty} w_a(h)^{\beta} \frac{dw_a(h)}{dt} dh \\ &= \int_{h=0}^{m_a} w_a(h)^{\beta} \frac{dw_a(h)}{dt} dh \\ &= m_a \left(w_a^{\beta} \left(\frac{-s_a}{m_a} \right) \right) \\ &= -w_a^{\beta} s_a \\ &= -w_a\end{aligned}$$

The fact that $-w_a^{\beta} s_a = w_a$ follows by the definition of A, as $s_a = w_a^{1/\alpha}$.

We now focus on the second integral in equation 5.15.

$$\begin{aligned}(5.17)\quad & -\int_{h=0}^{\infty} w_a(h)^{\beta} \frac{dw_o(h)}{dt} dh \\ &= \int_{h=0}^{m_o} w_a(h)^{\beta} \frac{s_o}{m_o} dh \\ &\leq \int_{h=0}^{m_o} w_a^{\beta} \frac{s_o}{m_o} dh \\ &= w_a^{\beta} s_o\end{aligned}$$

The inequality in equation 5.17 follows since $w_a(h)$ is non-increasing function of h, and $w_a(0) = w_a$.

We now focus on the third integral in equation 5.15. Recalling that $w_a(h) = w_a$ for $h \in [0, m_a)$, and $w_o(h)$ is non-increasing with h, we get,

$$\begin{aligned}(5.18)\quad & -\int_{h=0}^{\infty} w_a(h)^{\beta-1} w_o(h) \frac{dw_a(h)}{dt} dh \\ &= \int_{h=0}^{m_a} w_a(h)^{\beta-1} w_o \frac{s_a}{m_a} dh \\ &\leq \int_{h=0}^{m_o} w_a^{\beta-1} w_o \frac{s_a}{m_a} dh \\ &= w_a^{\beta-1} w_o s_a = w_o\end{aligned}$$

Combining equations 5.15, 5.16, 5.17 and 5.18 we get that:

$$(5.19)\quad \frac{d\Phi}{dt} \leq (\beta+1)\eta(-w_a + \beta w_o + w_a^{\beta} s_o)$$

We now consider two cases depending on whether $\alpha \in [1, 2]$ or whether $\alpha > 2$. For $\alpha \leq 2$, we apply Young's inequality (c.f. Corollary 6.1), with $a = w_a^{\beta}$, $b = s_0$, $p = 1/\beta$, and $q = \alpha$, and $\mu = 1$, which yields that

$$(5.20)\quad w_o^{\beta} s_o \leq \beta w_o + \frac{s_o^{\alpha}}{\alpha}$$

Plugging this in the inequality (5.19) we obtain that $\frac{d\Phi}{dt}$ is at most

$$(5.21)\quad \eta(\beta+1)(-w_a + 2\beta w_o + \frac{s_o^{\alpha}}{\alpha})$$

Plugging the bound on $\frac{d\Phi}{dt}$ given in equation 5.21 into equation 3.3, and regrouping terms, we obtain a bound on the competitive ratio of

$$\gamma \leq \frac{2 - \eta(\beta+1)w_a + 2\eta\beta(\beta+1)w_o + \eta(\beta+1)s_o^{\alpha}/\alpha}{w_o + s_o^{\alpha}}$$

Setting $\eta = 2/(\beta+1)$ to eliminate the w_a term, and observing that $\beta = (1 - 1/\alpha) \leq 1/2$ for $\alpha \leq 2$ and that $2/\alpha \leq 2$, we obtain the desired competitive ratio of 2 for this case.

We now consider the case of $\alpha > 2$. Applying Young's inequality (c.f. Corollary 6.1), with $a = w_a^{\beta}$, $b = s_0$, $p = 1/\beta$, and $q = \alpha$, and $\mu = (\alpha-1)^{-1/(\alpha-1)}$, we get that

$$w_a^{\beta} s_o \leq \mu\beta w_a + \left(\frac{1}{\mu}\right)^{\alpha\beta} \left(\frac{s_o^{\alpha}}{\alpha}\right)$$

As $\alpha\beta = \alpha - 1$ and in plugging the value of μ, this implies that $\mu^{-\alpha\beta} = \alpha - 1 = \alpha\beta$ and hence,

$$(5.22)\quad w_a^{\beta} s_o \leq \mu\beta w_a + \beta s_o^{\alpha}.$$

Plugging equation 5.22 into equation 5.19 we get that $\frac{d\Phi}{dt}$ is at most

(5.23)

$$\eta\left[-(\beta+1)w_a + \beta(\beta+1)w_o + \beta(\beta+1)\mu w_a + \beta(\beta+1)s_0^{\alpha}\right]$$

Plugging the bound on $\frac{d\Phi}{dt}$ given in equation 5.23 into equation 3.3, and regrouping terms, we obtain a bound on the competitive ratio of

$$\gamma \leq \frac{(2+\eta(\beta+1)(\mu\beta-1))w_a + \eta\beta(\beta+1)(w_o+s_o^\alpha)}{w_o+s_o^\alpha}$$

We set $\eta = -2/(\beta+1)(\mu\beta-1)$ to eliminate the w_a term, thus obtaining a bound on the competitive ratio of

$$\frac{2\beta}{1-\mu\beta} = \frac{2(\alpha-1)}{\alpha-(\alpha-1)^{1-1/(\alpha-1)}} = \gamma \tag{5.24}$$

By bounding equation 5.24 for some values of α, we get the following bounds on the competitive ratio for A: If $1 < \alpha \leq 2$ then $\gamma = 2$, if $\alpha > 2$ then $\gamma \leq 2(\alpha-1)$, if $\alpha \geq 2+e$ then $\gamma \leq \alpha - 1$, and finally, for large α, $\gamma \approx \alpha/\ln\alpha$.

To obtain a guarantee for integral weighted flow time, we define the algorithm C_ϵ to be the one that uses HDF for job selection, and whenever there is unfinished work, it runs at a power equal to $(1+\epsilon)$ times the power that algorithm A would run at. Note that C_ϵ must simulate algorithm A, and is not the same algorithm as run at power $(1+\epsilon)$ times the fractional weight of the active jobs.

COROLLARY 5.1. *Let $\mu_\epsilon = \max((1+\frac{1}{\epsilon}),(1+\epsilon)^\alpha)$. The algorithm C_ϵ, where $s_{c_\epsilon}(t) = (1+\epsilon)\underline{s_a}(t)$, is $\mu_\epsilon\gamma$-competitive with respect to the objective $\overline{G}$ of weighted flow plus energy. For large values of α, choosing $\epsilon \approx \ln\alpha/\alpha$ optimizes $\mu_\epsilon \approx \alpha/\ln\alpha$.*

Proof. (Sketch) Using ideas from [BLMSP01], it can be easily shown that at any time t, if some job j is alive under C_ϵ, then j has at least an $\epsilon/(1+\epsilon)$ fraction of its weight unfinished under A. Thus $\overline{W}_{c_\epsilon} \leq (1+\frac{1}{\epsilon})W_a$. Moreover $E_{c_\epsilon} \leq (1+\epsilon)^\alpha E_a$ as the speed under C_ϵ is always within $(1+\epsilon)$ times that of A. Together with Theorem 5.1 the result follows.

6 Technical Facts

We use the following classic inequality and its corollary from Section 8.3 of [HLP52]).

THEOREM 6.1. (YOUNG'S INEQUALITY) *Let f be a real-valued, continuous, and strictly increasing function on $[0,c]$ with $c > 0$. If $f(0) = 0$, and a, b such that $a \in [0,c]$, and $b \in [0, f(c)]$, then $\int_0^a f(x)dx + \int_0^b f^{(-1)}(x)dx \geq ab$ where $f^{(-1)}$ is the inverse function of f.*

COROLLARY 6.1. *For reals a, b, μ, p and q such $a \geq 0$, $b \geq 0$, $p \geq 1$, $\mu > 0$ and $1/p + 1/q = 1$, $\mu\frac{a^p}{p} + \left(\frac{1}{\mu}\right)^{q/p}\frac{b^q}{q} \geq ab$.*

The following lemma can be found in [BKP04].

LEMMA 6.1. *Let $q, r, \delta \geq 0$ and $\mu \geq 1$. Then $(q+\delta)^{\mu-1}(q-\mu r-(\mu-1)\delta) - q^{\mu-1}(q-\mu r) \leq 0$.*

References

[AF06] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. In *Lecture Notes in Computer Science (STACS)*, volume 3884, pages 621 – 633, 2006.

[BBS+00] David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[BKP04] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Dynamic speed scaling to manage energy and temperature. In *IEEE Syposium on Foundations of Computer Science*, pages 520 – 529, 2004.

[BLMSP01] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk R. Pruhs. Online weighted flow time and deadline scheduling. In *Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 36–47, 2001.

[Bun06] David Bunde. Power-aware scheduling for makespan and flow. In *Proceedings of the 2006 ACM Symposium on Parallel Algorithms and Architectures*, 2006. To appear.

[HLP52] G. H. Hardy, J. E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, 1952.

[IP05] Sandy Irani and Kirk R. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

[LLLK84] J. Labetoulle, E. Lawler, J.K. Lenstra, and A. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. *Progress in Combinatorial Optimization*, pages 245–261, 1984.

[Mud01] Trevor Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.

[PUW04] Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard Woeginger. Getting the best response for your erg. In *Scandanavian Workshop on Algorithms and Theory*, 2004.

[PvSU05] Kirk Pruhs, Rob van Stee, and Patchrawat Uthaisombut. Speed scaling of tasks with precedence constraints. In *Workshop on Approximation and Online Algorithms*, 2005.

[YDS95] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *IEEE Syposium on Foundations of Computer Science*, page 374, 1995.

A Online Lower Bound

In this section, we show that the problem of minimizing flow time subject to a fixed energy bound online has no

constant competitive algorithm. This records into the literature a fact that was generally known by researchers in this area.

THEOREM A.1. *Assume that there is some fixed energy bound E, and the objective is to minimize total flow time. Then there is no $O(1)$-competitive online speed scaling algorithm even for unit work and unit weight instances.*

Proof. We will give an adversarial strategy for generating an input. The jobs are divided into batches $B_1, \ldots, B_\ell$. All the jobs in B_i arrive together some time after the online algorithm has finished all the jobs in B_{i-1}.

Before describing how we generate B_i, let's consider what happens when there are no unfinished jobs and a set of jobs all arrive at the same time. This subproblem is basically equivalent to the off-line problem without release dates. Thus, with n_i jobs and an energy budget of E_i, we can use the results in [PUW04] to derive the optimal strategy. These results say that we should run job j at power equal to $\sigma = \rho(n_i - (j-1))$ for some constant ρ. Thus we are running at speed $\sigma^{1/\alpha}$ and it takes time $\sigma^{-1/\alpha}$ to finish job j. The total energy expended is the integral of power over time, which is

$$\sum_{j=1}^{n_i} = \sigma^{1-1/\alpha} = \rho^{1-1/\alpha} \sum_{j=1}^{n_i} j^{1-1/\alpha} ,$$

which implies that

$$\rho = \left(E_i / \sum_{j=1}^{n_i} j^{1-1/\alpha} \right)^{\alpha/(\alpha-1)} .$$

The sum of flow time for this set of jobs is

$$\begin{aligned} & \sum_{j=1}^{n_i} (n_i - (j-1)) \sigma^{-1/\alpha} \\ = \; & \rho^{-1/\alpha} \sum_{j=1}^{n_i} j^{1-1/\alpha} \\ = \; & \frac{\left(\sum_{j=1}^{n_i} j^{1-1/\alpha} \right)^{\alpha/(\alpha-1)}}{E_i^{1/(\alpha-1)}} \end{aligned}$$

We now approximate the sum by an integral (the error is negligible for these calculations) and obtain a bound of

$$\left(n_i^{2\alpha-1} / ((2 - 1/\alpha) E_i) \right)^{1/(\alpha-1)} . \tag{1.25}$$

This term has E_i in the denominator, so if E_i were off by more than a constant factor, the entire flow time would be off by more than a constant factor. This motivates our construction, in which we divide time into batches and show that in each batch, the energy used has to be within a constant factor of the total energy. Since we will have a non-constant number of batches, we will derive a contradiction.

Let batch B_i contain $n_i = (2-1/\alpha)^{1/(2\alpha-1)} 2^{i/(2\alpha-1)}$ unit work jobs. Now we argue that if the online algorithm doesn't know ℓ, the number of blocks, it can not be $O(1)$ competitive.

Plugging our choice of n_i into (1.25), we get a flow time of $2^{i/(\alpha-1)}/E_i^{1/(\alpha-1)}$ for batch i. The adversary, who knows ℓ, could set $E_i = 2^i E / \sum_{k=1}^{\ell} 2^k$. With this choice, we clearly have that $\sum_{i=1}^{\ell} E_i = E$, as desired. We also get that the flow time of each batch is $\left(\sum_{k=1}^{\ell} 2^k / E \right)^{1/(\alpha-1)} = \left((2^{\ell+1} - 2)/E \right)^{1/(\alpha-1)}$ and that the total flow time is $\left(\ell \left(2^{\ell+1} - 2 \right) / E \right)^{1/(\alpha-1)}$. Note also that even by allocating all the energy on the last batch, the flow time $(2^\ell/E)^{1/(\alpha-1)}$ is only less by a factor of $\Theta(\ell)$. Thus, since the online algorithm does not know ℓ (or the future job arrivals), it must allocate an $\omega(1/i)$ fraction of the energy to each batch B_i. Thus, after seeing ℓ batches, it will have allocated $\Omega(E \log \ell)$ energy, which is impossible.