

NONCLAIRVOYANT SPEED SCALING FOR FLOW AND ENERGY

HO-LEUNG CHAN¹ AND JEFF EDMONDS² AND TAK-WAH LAM³ AND LAP-KEI LEE³ AND
ALBERTO MARCHETTI-SPACCAMELA⁴ AND KIRK PRUHS⁵

¹ Max-Planck-Institut für Informatik
E-mail address, H.L. Chan: hlchan@mpi-inf.mpg.de

² Department of Computer Science and Engineering, York University
E-mail address, J.Edmonds: jeff@cse.yorku.ca

³ Department of Computer Science, University of Hong Kong
E-mail address, T.W.Lam: twlam@cs.hku.hk
E-mail address, L.K. Lee: lklee@cs.hku.hk

⁴ Dipartimento di Informatica e Sistemistica, Sapienza Università di Roma
E-mail address, A.Marchetti-Spaccamela: alberto@dis.uniroma1.it

⁵ Computer Science Department, University of Pittsburgh
E-mail address, K.Pruhs: kirk@cs.pitt.edu

ABSTRACT. We study online nonclairvoyant speed scaling to minimize total flow time plus energy. We first consider the traditional model where the power function is $P(s) = s^\alpha$. We give a nonclairvoyant algorithm that is shown to be $O(\alpha^3)$ -competitive. We then show an $\Omega(\alpha^{1/3-\epsilon})$ lower bound on the competitive ratio of any nonclairvoyant algorithm. We also show that there are power functions for which no nonclairvoyant algorithm can be $O(1)$ -competitive.

1. Introduction

Energy consumption has become a key issue in the design of microprocessors. Major chip manufacturers, such as Intel, AMD and IBM, now produce chips with dynamically scalable speeds, and produce associated software, such as Intel's SpeedStep and AMD's PowerNow, that enables an operating system to manage power by scaling processor speed. Thus the operating system should have a *speed scaling* policy for setting the speed of the processor, that ideally should work in tandem with a *job selection* policy for determining which job to run. The operating system has dual competing objectives, as it both wants to optimize some schedule quality of service objective, as well as some power related objective.

The work of H.L.Chan was done when he was a postdoc in University of Pittsburgh. T.W.Lam is partially supported by HKU Grant 7176104. A.Marchetti-Spaccamela is partially supported by MIUR FIRB grant RBIN047MH9 and by EU ICT-FET grant 215270 FRONTS. K.Pruhs is partially supported by an IBM faculty award, and from NSF grants CNS-0325353, CCF-0514058, IIS-0534531, and CCF-0830558.

In this paper, we will consider the objective of minimizing a linear combination of total flow and total energy used. For a formal definitions of the problem that we consider, see subsection 1.2. This objective of flow plus energy has a natural interpretation. Suppose that the user specifies how much improvement in flow, call this amount ρ , is necessary to justify spending one unit of energy. For example, the user might specify that he is willing to spend 1 erg of energy from the battery for a decrease of 5 micro-seconds in flow. Then the optimal schedule, from this user's perspective, is the schedule that optimizes $\rho = 5$ times the energy used plus the total flow. By changing the units of either energy or time, one may assume without loss of generality that $\rho = 1$.

In order to be implementable in a real system, the speed scaling and job selection policies must be online since the system will not in general know about jobs arriving in the future. Further, to be implementable in a generic operating system, these policies must be nonclairvoyant, since in general the operating system does not know the size/work of each process when the process is released to the operating system. All of the previous speed scaling literature on this objective has considered either offline or online clairvoyant policies. In subsection 1.1, we survey the literature on nonclairvoyant scheduling policies for flow objectives on fixed speed processors, and the speed scaling literature for flow plus energy objectives.

Our goal in this paper is to study nonclairvoyant speed scaling assuming an off-line adversary that dynamically chooses the speed of its own machine.

We first analyze the nonclairvoyant algorithm whose job selection policy is Latest Arrival Processor Sharing (LAPS) and whose speed scaling policy is to run at speed $(1 + \delta)$ times the number of active jobs. LAPS shares the processor equally among the latest arriving constant fraction of the jobs. We adopt the traditional model that the power function, which gives the power as a function of the speed of the processor, is $P(s) = s^\alpha$, where $\alpha > 1$ is some constant. Of particular interest is the case that $\alpha = 3$ since according to the well known cube-root rule, the dynamic power in CMOS based processors is approximately the cube of the speed. Using an amortized local competitiveness argument, we show in section 2 that this algorithm is $O(\alpha^3)$ -competitive. The potential function that we use is an amalgamation of the potential function used in [8] for the fixed speed analysis of LAPS, and the potential functions used for analyzing clairvoyant speed scaling policies. This result shows that it is possible for a nonclairvoyant policy to be $O(1)$ -competitive if the cube-root rule holds.

It is known that for essentially every power function, there is a 3-competitive *clairvoyant* speed scaling policy [3]. In contrast, we show that the competitiveness achievable by *nonclairvoyant* policies must depend on the power function. In the traditional model, we show in section 3 an $\Omega(\alpha^{1/3-\epsilon})$ lower bound on the competitive ratio of any deterministic nonclairvoyant algorithm. Further, we show in section 3 that there exists a particular power function for which there is no $O(1)$ -competitive deterministic nonclairvoyant speed scaling algorithm. The adversarial strategies for these lower bounds are based on the adversarial strategies in [13] for fixed speed processors. Perhaps these lower bound results are not so surprising given the fact that it is known that without speed scaling, resource augmentation is required to achieve $O(1)$ -competitiveness for a nonclairvoyant policy [13, 10]. Still a priori it wasn't completely clear that the lower bounds in [13] would carry over. The reason is that in these lower bound instances, the adversary forced the online algorithm into a situation in which the online algorithm had a lot of jobs with a small amount of remaining work, while the adversary had one job left with a lot of remaining work. In the fixed speed setting,

the online algorithm, without resource augmentation, can never get a chance to get rid of this backlog in the face of a steady stream of jobs. However, in a speed scaling setting, one might imagine an online algorithm that speeds up enough to remove the backlog, but not enough to make its energy usage more than a constant time optimal. Our lower bound shows that it is not possible for the online algorithm to accomplish this.

1.1. Related results

We start with some results in the literature about scheduling with the objective of total flow time on a fixed speed processor. It is well known that the online clairvoyant algorithm Shortest Remaining Processing Time (SRPT) is optimal. The competitive ratio of deterministic nonclairvoyant algorithm is $\Omega(n^{1/3})$, and the competitive ratio of every randomized algorithm against an oblivious adversary is $\Omega(\log n)$ [13]. A randomized version of the Multi-Level Feedback Queue algorithm is $O(\log n)$ -competitive [11, 5]. The non-clairvoyant algorithm Shortest Elapsed Time First (SETF) is scalable, that is, $(1 + \epsilon)$ -speed $O(1)$ -competitive [10]. SETF shares the processor equally among all jobs that have been run the least. The algorithm Round Robin RR (also called Equipartition and Processor Sharing) that shares the processor equally among all jobs is $(2 + \epsilon)$ -speed $O(1)$ -competitive [7].

Let us first consider the traditional model where the power function is $P = s^\alpha$. Most of the literature assumes the *unbounded speed model*, in which a processor can be run at any real speed in the range $[0, \infty)$. So let us now consider the unbounded speed model. [15] gave an efficient offline algorithm to find the schedule that minimizes average flow subject to a constraint on the amount of energy used, in the case that jobs have unit work. This algorithm can also be used to find optimal schedules when the objective is a linear combination of total flow and energy used. [15] observed that in any locally-optimal schedule, essentially each job i is run at a power proportional to the number of jobs that would be delayed if job i was delayed. [1] proposed the natural online speed scaling algorithm that always runs at a power equal to the number of unfinished jobs (which is lower bound to the number of jobs that would be delayed if the selected job was delayed). [1] did not actually analyze this natural algorithm, but rather analyzed a batched variation, in which jobs that are released while the current batch is running are ignored until the current batch finishes. [1] showed that for unit work jobs this batched algorithm is $O\left(\left(\frac{3+\sqrt{5}}{2}\right)^\alpha\right)$ -competitive by reasoning directly about the optimal schedule. [1] also gave an efficient offline dynamic programming algorithm. [4] considered the algorithm that runs at a power equal to the unfinished work (which is in general a bit less than the number of unfinished jobs for unit work jobs). [4] showed that for unit work jobs, this algorithm is 2-competitive with respect to the objective of fractional flow plus energy using an amortized local competitiveness argument. [4] then showed that the natural algorithm proposed in [1] is 4-competitive for total flow plus energy for unit work jobs.

In [4] the more general setting where jobs have arbitrary sizes and arbitrary weights and the objective is weighted flow plus energy has been considered. The authors analysed the algorithm that uses Highest Density First (HDF) for job selection, and always runs at a power equal to the fractional weight of the unfinished jobs. [4] showed that this algorithm is $O\left(\frac{\alpha}{\log \alpha}\right)$ -competitive for fractional weighted flow plus energy using an amortized local competitiveness argument. [4] then showed how to modify this algorithm to obtain an

algorithm that is $O(\frac{\alpha^2}{\log^2 \alpha})$ -competitive for (integral) weighted flow plus energy using the known resource augmentation analysis of HDF [6].

Recently, [12] improves on the obtainable competitive ratio for total flow plus energy for arbitrary work and unit weight jobs by considering the job selection algorithm Shortest Remaining Processing Time (SRPT) and the speed scaling algorithm of running at a power proportional to the number of unfinished jobs. [12] proved that this algorithm is $O(\frac{\alpha}{\log \alpha})$ -competitive for arbitrary size and unit weight jobs.

In [2] the authors extended the results of [4] for the unbounded speed model to the bounded speed model, where there is an upper bound on the processor speed. The speed scaling algorithm was to run at the minimum of the speed recommended by the speed scaling algorithm in the unbounded speed model and the maximum speed of the processor. The results for the bounded speed model in [2] were improved in [12] proving competitive ratios of the form $O(\frac{\alpha}{\log \alpha})$.

[3] consider a more general model. They assume that the allowable speeds are a countable collection of disjoint subintervals of $[0, \infty)$, and consider arbitrary power functions P that are non-negative, and continuous and differentiable on all but countably many points. They give two main results in this general model. The scheduling algorithm, that uses Shortest Remaining Processing Time (SRPT) for job selection and power equal to one more than the number of unfinished jobs for speed scaling, is $(3 + \epsilon)$ -competitive for the objective of total flow plus energy on arbitrary-work unit-weight jobs. The scheduling algorithm, that uses Highest Density First (HDF) for job selection and power equal to the fractional weight of the unfinished jobs for speed scaling, is $(2 + \epsilon)$ -competitive for the objective of fractional weighted flow plus energy on arbitrary-work arbitrary-weight jobs.

1.2. Formal Problem Definition and Notations

We study online scheduling on a single processor. Jobs arrive over time and we have no information about a job until it arrives. For each job j , its release time and work requirement (or size) are denoted as $r(j)$ and $p(j)$, respectively. We consider the *nonclairvoyant* model, i.e., when a job j arrives, $p(j)$ is not given and it is known only when j is completed. Preemption is allowed and has no cost; a preempted job can resume at the point of preemption. The processor can vary its speed dynamically to any value in $[0, \infty)$. When running at speed s , the processor processes s units of work per unit time and consumes $P(s) = s^\alpha$ units of energy per unit time, where $\alpha > 1$ is some fixed constant. We call $P(s)$ the *power function*.

Consider any job sequence I and a certain schedule A of I . For any job j in I , the flow time of j , denoted $F_A(j)$, is the amount of time elapsed since it arrives until it is completed. The total flow time of the schedule is $F_A = \sum_{j \in I} F_A(j)$. We can also interpret F_A as follows. Let $n_A(t)$ be the number of jobs released by time t but not yet completed by time t . Then $F_A = \int_0^\infty n_A(t) dt$. Let $s_A(t)$ be the speed of the processor at time t in the schedule. Then the total energy usage of the schedule is $E_A = \int_0^\infty (s(t))^\alpha dt$. The objective is to minimize the sum of total flow time and energy usage, i.e., $F_A + E_A$.

For any job sequence I , a scheduling algorithm ALG needs to specify at any time the speed of the processor and the jobs being processed. We denote $\text{ALG}(I)$ as the schedule produced for I by ALG. Let Opt be the optimal offline algorithm such that for any job sequence I , $F_{\text{Opt}(I)} + E_{\text{Opt}(I)}$ is minimized among all schedules of I . An algorithm ALG is

said to be c -competitive, for any $c \geq 1$, if for all job sequence I ,

$$F_{ALG(I)} + E_{ALG(I)} \leq c \cdot (F_{Opt(I)} + E_{Opt(I)})$$

2. An $O(\alpha^3)$ -competitive Algorithm

In this section, we give an online nonclairvoyant algorithm that is $O(\alpha^3)$ -competitive for total flow time plus energy. We say a job j is *active* at time t if j is released by time t but not yet completed by time t . Our algorithm is defined as follows.

Algorithm LAPS(δ, β). Let $0 < \delta, \beta \leq 1$ be any real. At any time t , the processor speed is $(1 + \delta)(n(t))^{1/\alpha}$, where $n(t)$ is the number of active jobs at time t . The processor processes the $\lceil \beta n(t) \rceil$ active jobs with the latest release times (ties are broken by job ids) by splitting the processing speed equally among these jobs.

Our main result is the following.

Theorem 2.1. *When $\delta = \frac{3}{\alpha}$ and $\beta = \frac{1}{2\alpha}$, LAPS(δ, β) is c -competitive for total flow time plus energy, where $c = 4\alpha^3(1 + (1 + \frac{3}{\alpha})^\alpha) = O(\alpha^3)$.*

The rest of this section is devoted to proving Theorem 2.1. We use an amortized local competitiveness argument (see for example [14]). To show that an algorithm is c -competitive it is sufficient to show a potential function such that at any time t the increase in the objective cost of the algorithm plus the change of the potential is at most c times the increase in the objective of the optimum.

For any time t , let $G_a(t)$ and $G_o(t)$ be the total flow time plus energy incurred up to time t by LAPS(δ, β) and the optimal algorithm Opt, respectively. To show that LAPS(δ, β) is c -competitive, it suffices to give a potential function $\Phi(t)$ such that the following four conditions hold.

- *Boundary condition:* $\Phi = 0$ before any job is released and $\Phi \geq 0$ after all jobs are completed.
- *Job arrival:* When a job is released, Φ does not increase.
- *Job completion:* When a job is completed by LAPS(δ, β) or OPT, Φ does not increase.
- *Running condition:* At any other time, the rate of change of G_a plus that of Φ is no more than c times the rate of change of G_o . That is, $\frac{dG_a(t)}{dt} + \frac{d\Phi(t)}{dt} \leq c \cdot \frac{dG_o(t)}{dt}$ during any period of time without job arrival or completion.

Let $n_a(t)$ and $s_a(t)$ be the number of active jobs and the speed in LAPS(δ, β) at time t , respectively. Define $n_o(t)$ and $s_o(t)$ similarly for that of Opt. Then

$$\frac{dG_a(t)}{dt} = \frac{dF_{LAPS}(t)}{dt} + E_{LAPS}(t) = n_a(t) + (s_a(t))^\alpha$$

and, similarly, $\frac{dG_o(t)}{dt} = n_o(t) + (s_o(t))^\alpha$. We define our potential function as follows.

Potential function $\Phi(t)$. Consider any time t . For any job j , let $q_a(j, t)$ and $q_o(j, t)$ be the remaining work of j at time t in LAPS(δ, β) and Opt, respectively. Let $\{j_1, \dots, j_{n_a(t)}\}$ be the set of active jobs in LAPS(δ, β),

ordered by their release time such that $r(j_1) \leq r(j_2) \leq \dots \leq r(j_{n_a(t)})$.

Then,

$$\Phi(t) = \gamma \sum_{i=1}^{n_a(t)} \left(i^{1-1/\alpha} \cdot \max\{0, q_a(j_i, t) - q_o(j_i, t)\} \right)$$

where $\gamma = \alpha(1 + (1 + \frac{3}{\alpha})^\alpha)$. We call $i^{1-1/\alpha}$ the *coefficient* of j_i .

We first check the boundary, job arrival and job completion conditions. Before any job is released or after all jobs are completed, there is no active job in both LAPS(δ, β) and Opt, so $\Phi = 0$ and the boundary condition holds. When a new job j arrives at time t , $q_a(j, t) - q_o(j, t) = 0$ and the coefficients of all other jobs remain the same, so Φ does not change. If LAPS(δ, β) completes a job j , the term for j in Φ is removed. The coefficient of any other job either stays the same or decreases, so Φ does not increase. If Opt completes a job, Φ does not change.

It remains to check the running condition. In the following, we focus on a certain time t within a period of time without job arrival or completion. We omit the parameter t from the notations as t refers only to this certain time. For example, we denote $n_a(t)$ and $q_a(j, t)$ as n_a and $q_a(j)$, respectively. For any job j , if LAPS(δ, β) has processed less than Opt on j at time t , i.e., $q_a(j) - q_o(j) > 0$, then we say that j is a *lagging job* at time t . We start by evaluating $\frac{d\Phi}{dt}$.

Lemma 2.2. *Assume $\delta = \frac{3}{\alpha}$ and $\beta = \frac{1}{2\alpha}$. At time t , if LAPS(δ, β) is processing less than $(1 - \frac{1}{2\alpha})\lceil \beta n_a \rceil$ lagging jobs, then $\frac{d\Phi}{dt} \leq \frac{\gamma}{\alpha} s_o^\alpha + \gamma(1 - \frac{1}{\alpha})n_a$. Else if LAPS(δ, β) is processing at least $(1 - \frac{1}{2\alpha})\lceil \beta n_a \rceil$ lagging jobs, then $\frac{d\Phi}{dt} \leq \frac{\gamma}{\alpha} s_o^\alpha - \frac{\gamma}{\alpha} n_a$.*

Proof. We consider $\frac{d\Phi}{dt}$ as the combined effect due to the processing of LAPS(δ, β) and Opt. Note that for any job j , $q_a(j)$ is decreasing at a rate of either 0 or $-s_a/\lceil \beta n_a \rceil$. Thus the rate of change of Φ due to LAPS(δ, β) is non-positive. Similarly, the rate of change of Φ due to Opt is non-negative.

We first bound the rate of change of Φ due to Opt. The worst case is that Opt is processing the job with the largest coefficient, i.e., $n_a^{1-1/\alpha}$. Thus the rate of change of Φ due to Opt is at most $\gamma n_a^{1-1/\alpha} (-\frac{dq_o(j_{n_a})}{dt}) = \gamma n_a^{1-1/\alpha} s_o$. We apply Young's Inequality [9], which is formally stated in Lemma 2.3, by setting $f(x) = x^{\alpha-1}$, $f^{-1}(x) = x^{1/(\alpha-1)}$, $g = s_o$ and $h = n_a^{1-1/\alpha}$. Then, we have

$$s_o n_a^{1-1/\alpha} \leq \int_0^{s_o} x^{\alpha-1} dx + \int_0^{n_a^{1-1/\alpha}} x^{1/(\alpha-1)} dx = \frac{1}{\alpha} s_o^\alpha + (1 - \frac{1}{\alpha}) n_a$$

If LAPS(δ, β) is processing less than $(1 - \frac{1}{2\alpha})\lceil \beta n_a \rceil$ lagging jobs, we just ignore the effect due to LAPS(δ, β) and take the bound that $\frac{d\Phi}{dt} \leq \frac{\gamma}{\alpha} s_o^\alpha + \gamma(1 - \frac{1}{\alpha})n_a$.

If LAPS(δ, β) is processing at least $(1 - \frac{1}{2\alpha})\lceil \beta n_a \rceil$ lagging jobs, let j_i be one of these lagging jobs. We notice that j_i is among the $\lceil \beta n_a \rceil$ active jobs with the latest release times. Thus, the coefficient of j_i is at least $(n_a - \lceil \beta n_a \rceil + 1)^{1-1/\alpha}$. Also, j_i is being processed at a speed of $s_a/\lceil \beta n_a \rceil$, so $q_a(j_i, t)$ is decreasing at this rate. LAPS(δ, β) is processing at least $(1 - \frac{1}{2\alpha})\lceil \beta n_a \rceil$ such lagging jobs, so the rate of change of Φ due to LAPS(δ, β) is more

negative than

$$\begin{aligned}
& \gamma \left(\left(1 - \frac{1}{2\alpha}\right) \lceil \beta n_a \rceil \right) (n_a - \lceil \beta n_a \rceil + 1)^{1-1/\alpha} \left(\frac{-s_a}{\lceil \beta n_a \rceil} \right) \\
& \leq -\gamma \left(1 - \frac{1}{2\alpha}\right) (n_a - \beta n_a)^{1-1/\alpha} (s_a) \quad (\text{since } -\lceil \beta n_a \rceil + 1 \geq -\beta n_a) \\
& \leq -\gamma \left(1 - \frac{1}{2\alpha}\right) (1 - \beta) (1 + \delta) n_a \quad (\text{since } s_a = (1 + \delta) n_a^{1/\alpha})
\end{aligned}$$

When $\beta = \frac{1}{2\alpha}$ and $\delta = \frac{3}{\alpha}$, simple calculation shows that $(1 - \frac{1}{2\alpha})(1 - \beta)(1 + \delta) \geq 1$ and hence the last term above is at most $-\gamma n_a$. It follows that $\frac{d\Phi}{dt} \leq \frac{\gamma}{\alpha} s_o^\alpha + \gamma(1 - \frac{1}{\alpha}) n_a - \gamma n_a = \frac{\gamma}{\alpha} s_o^\alpha - \frac{\gamma}{\alpha} n_a$. ■

Below is the formal statement of Young's Inequality, which is used in the proof of Lemma 2.2.

Lemma 2.3 (Young's Inequality [9]). *Let f be any real-value, continuous and strictly increasing function f such that $f(0) = 0$. Then, for all $g, h \geq 0$, $\int_0^g f(x)dx + \int_0^h f^{-1}(x)dx \geq gh$, where f^{-1} is the inverse function of f .*

We are now ready to show the following lemma about the running condition.

Lemma 2.4. *Assume $\delta = \frac{3}{\alpha}$ and $\beta = \frac{1}{2\alpha}$. At time t , $\frac{dG_a}{dt} + \frac{d\Phi}{dt} \leq c \cdot \frac{dG_o}{dt}$, where $c = 4\alpha^3(1 + (1 + \frac{3}{\alpha})^\alpha)$.*

Proof. We consider two cases depending on the number of lagging jobs that LAPS(δ, β) is processing at time t . If LAPS(δ, β) is processing at least $(1 - \frac{1}{\alpha})\lceil \beta n_a \rceil$ lagging jobs, then

$$\begin{aligned}
\frac{dG_a}{dt} + \frac{d\Phi}{dt} &= n_a + s_a^\alpha + \frac{d\Phi}{dt} \\
&\leq n_a + (1 + \delta)^\alpha n_a + \frac{\gamma}{\alpha} s_o^\alpha - \frac{\gamma}{\alpha} n_a \quad (\text{by Lemma 2.2}) \\
&= (1 + (1 + \delta)^\alpha - \frac{\gamma}{\alpha}) n_a + \frac{\gamma}{\alpha} s_o^\alpha
\end{aligned}$$

Since $\delta = \frac{3}{\alpha}$ and $\gamma = \alpha(1 + (1 + \frac{3}{\alpha})^\alpha)$, the coefficient of n_a becomes zero and $\frac{dG_a}{dt} + \frac{d\Phi}{dt} \leq \frac{\gamma}{\alpha} s_o^\alpha$. Note that $\frac{\gamma}{\alpha} = (1 + (1 + \frac{3}{\alpha})^\alpha) \leq c$ and $\frac{dG_o}{dt} = n_o + s_o^\alpha$, so we have $\frac{dG_a}{dt} + \frac{d\Phi}{dt} \leq c \cdot \frac{dG_o}{dt}$.

If LAPS(δ, β) is processing less than $(1 - \frac{1}{2\alpha})\lceil \beta n_a \rceil$ lagging jobs, the number of jobs remaining in Opt is $n_o \geq \lceil \beta n_a \rceil - (1 - \frac{1}{2\alpha})\lceil \beta n_a \rceil = \frac{1}{2\alpha}\lceil \beta n_a \rceil \geq \frac{1}{2\alpha}\beta n_a = \frac{1}{4\alpha^2} n_a$. Therefore,

$$\begin{aligned}
\frac{dG_a}{dt} + \frac{d\Phi}{dt} &= n_a + s_a^\alpha + \frac{d\Phi}{dt} \\
&\leq n_a + (1 + \delta)^\alpha n_a + \frac{\gamma}{\alpha} s_o^\alpha + \gamma(1 - \frac{1}{\alpha}) n_a \quad (\text{by Lemma 2.2}) \\
&= (1 + (1 + \delta)^\alpha + \gamma(1 - \frac{1}{\alpha})) n_a + \frac{\gamma}{\alpha} s_o^\alpha \\
&\leq 4\alpha^2(1 + (1 + \delta)^\alpha + \gamma(1 - \frac{1}{\alpha})) n_o + \frac{\gamma}{\alpha} s_o^\alpha
\end{aligned}$$

Since $\delta = \frac{3}{\alpha}$ and $\gamma = \alpha(1 + (1 + \frac{3}{\alpha})^\alpha)$, the coefficient of n_o becomes $4\alpha^3(1 + (1 + \frac{3}{\alpha})^\alpha) = c$. The coefficient of s_o^α is $(1 + (1 + \frac{3}{\alpha})^\alpha) \leq c$. Since $\frac{dG_o(t)}{dt} = n_o + s_o^\alpha$, we obtain $\frac{dG_a(t)}{dt} + \frac{d\Phi}{dt} \leq c \cdot \frac{dG_o(t)}{dt}$. Note that this case is the bottleneck leading to the current competitive ratio. ■

Combining Lemma 2.4 with the discussion on the boundary, job arrival and job completion conditions, Theorem 2.1 follows.

3. Lower Bounds

In this section, we show that every nonclairvoyant algorithm is $\Omega(\alpha^{1/3-\epsilon})$ -competitive in the traditional model where the power function $P(s) = s^\alpha$. We further extend the lower bound to other power functions P and show that for some power function, any algorithm is $\omega(1)$ -competitive. We first prove the following lemma.

Lemma 3.1. *Let $P(s)$ be any non-negative, continuous and super-linear power function. Let $k, v \geq 1$ be any real such that $P(v) \geq 1$. Then, any algorithm is $\Omega(\min\{k, P(v + \frac{1}{16(kP(v))^3})/P(v)\})$ -competitive.*

Proof. Let ALG be any algorithm and Opt be the offline adversary. Let $n = \lceil kP(v) \rceil$. We release n jobs j_1, j_2, \dots, j_n at time 0. Let T be the first time that some job in ALG is processed for at least n units of work. Let $G(T)$ be the total flow time plus energy incurred by ALG up to T . We consider two cases depending on $G(T) \geq kn^3$ or $G(T) < kn^3$. If $G(T) \geq kn^3$, Opt reveals that all jobs are of size n . By running at speed 1, Opt completes all jobs by time n^2 . The total flow time plus energy of Opt is at most $n^3 + n^2P(1) \leq 2n^3$, so ALG is $\Omega(k)$ -competitive.

The rest of the proof assumes $G(T) < kn^3$. Let q_1, q_2, \dots, q_n be the amount of work ALG has processed for each of the n jobs. Without loss of generality, we assume $q_n = n$. Opt reveals that the size of each job j_i is $p_i = q_i + 1$. Thus, at time T , ALG has n remaining jobs, each of size 1. For Opt, it runs at the same speed as ALG during $[0, T]$ and processes exactly the same job as ALG except on j_n . By distributing the n units of work processed on j_n to all the n jobs, Opt can complete j_1, \dots, j_{n-1} by time T and the remaining size of j_n is n . As Opt is simulating ALG on all jobs except j_n , the total flow plus energy incurred by Opt up to T is at most $G(T) < kn^3$.

During $[T, T + n^4]$, Opt releases a stream of small jobs. Specifically, let $\epsilon < \frac{1}{n^5v^2}$ be any real. For $i = 1, \dots, \frac{n^4}{\epsilon}$, a small job j'_i is released at $T + (i-1)\epsilon$ with size ϵv . Opt can run at speed v and complete each small job before the next one is released. Thus, Opt has at most one small job and j_n remaining at any time during $[T, T + n^4]$. The flow time plus energy incurred during this period is $2n^4 + n^4P(v)$. Opt can complete j_n by running at speed 1 during $[T + n^4, T + n^4 + n]$, incurring a cost of $n + nP(1)$. Thus, the total flow time plus energy of Opt for the whole job sequence is at most $kn^3 + 2n^4 + n^4P(v) + n + nP(1) = O(n^4P(v))$.

For ALG, we first show that its total work done on the small jobs during $[T, T + n^4]$ is at least $n^4v - 1$. Otherwise, there are at least $\frac{1}{\epsilon v} > n^5v$ small jobs not completed by $T + n^4$. The best case is when these jobs are released during $[T + n^4 - \frac{1}{v}, T + n^4]$ and their total flow time incurred is $\Omega(n^5)$. It means that ALG is $\Omega(k)$ -competitive as $n = \lceil kP(v) \rceil$.

We call j_1, \dots, j_n big jobs and then consider the number of big jobs completed by ALG by time $T + n^4$. If ALG completes less than $\frac{1}{2}n + 1$ big jobs by time $T + n^4$, then ALG has at least $\frac{1}{2}n - 1$ big jobs remaining at any time during $[T, T + n^4]$. The total flow time of ALG is at least $\Omega(n^5)$, meaning that ALG is $\Omega(k)$ -competitive. If ALG completes at least $\frac{1}{2}n + 1$ big jobs by time $T + n^4$, the total work done by ALG during $[T, T + n^4]$ is at least

$n^4v - 1 + \frac{1}{2}n + 1$. The total energy used by ALG is at least

$$P\left(\frac{n^4v + \frac{1}{2}n}{n^4}\right) \times n^4 = P\left(v + \frac{1}{2n^3}\right) \times n^4 \geq P\left(v + \frac{1}{16(kP(v))^3}\right) \times n^4$$

The last inequality comes from the fact that $n = \lceil kP(v) \rceil \leq 2kP(v)$. Hence, ALG is at least $\Omega(P(v + \frac{1}{16(kP(v))^3})/P(v))$ -competitive. ■

Then, we can apply Lemma 3.1 to obtain the lower bound for the power function $P(s) = s^\alpha$.

Theorem 3.2. *When the power function is $P(s) = s^\alpha$ for some $\alpha > 1$, any algorithm is $\Omega(\alpha^{1/3-\epsilon})$ -competitive for any $0 < \epsilon < 1/3$.*

Proof. We apply Lemma 3.1 by putting $k = \alpha^{1/3-\epsilon}$ and $v = 1$. Then, $P(v) = 1$ and

$$P\left(v + \frac{1}{16(kP(v))^3}\right)/P(v) = \left(1 + \frac{1}{16(\alpha^{1/3-\epsilon})^3}\right)^\alpha = \left(1 + \frac{1}{16\alpha^{1-3\epsilon}}\right)^{(\alpha^{1-3\epsilon}) \times \alpha^{3\epsilon}}$$

Since $(1 + \frac{1}{16x})^x$ is increasing with x and $\alpha^{1-3\epsilon} \geq 1$, the last term above is at least $(1 + \frac{1}{16})^{\alpha^{3\epsilon}}$. Thus, $\min\{k, P(v + \frac{1}{16(kP(v))^3})/P(v)\} \geq \min\{\alpha^{1/3-\epsilon}, (\frac{17}{16})^{\alpha^{3\epsilon}}\} = \Omega(\alpha^{1/3-\epsilon})$, and the theorem follows. ■

We also show that for some power function, any algorithm is $\omega(1)$ -competitive.

Theorem 3.3. *There exists some power function P such that any algorithm is $\omega(1)$ -competitive.*

Proof. We want to find a power function P such that for any $k \geq 1$, there exists a speed v such that $P(v + \frac{1}{16(kP(v))^3})/P(v) \geq k$. Then by setting k and v correspondingly to Lemma 3.1, any algorithm is at least k -competitive for any $k \geq 1$. It implies that any algorithm is $\omega(1)$ -competitive. For example, consider the power function

$$P(s) = \frac{1}{(4(2-s))^{1/4}}, \quad 0 \leq s < 2$$

Let P' be the derivative of P . We can verify that $P'(s) = (P(s))^5$ for all $0 \leq s < 2$. For any k , let $v \geq 1$ be a speed such that $P(v) \geq 16k^4$. Then,

$$P\left(v + \frac{1}{16(kP(v))^3}\right) \geq P(v) + P'(v) \frac{1}{16(kP(v))^3} \geq (P(v))^5 \frac{1}{16(kP(v))^3} \geq kP(v)$$

Thus, $P(v + \frac{1}{16(kP(v))^3})/P(v) \geq k$ and the theorem follows. ■

4. Conclusion

We show that nonclairvoyant policies can be $O(1)$ -competitive in the traditional power model. However, we showed that in contrast to the case for clairvoyant algorithms, there are power functions that are sufficiently quickly growing that nonclairvoyant algorithms can not be $O(1)$ -competitive.

One obvious open problem is to reduce the competitive ratio achievable by a nonclairvoyant algorithm in the case that the cube-root rule holds to something significantly more reasonable than the rather high bound achieved here.

The standard/best nonclairvoyant job selection policy for a fixed speed processor is Short Elapsed Time First (SETF). The most obvious candidate speed scaling policy would be to use SETF for job selection, and to run at power somewhat higher than the number of active jobs. The difficulty with analyzing this speed scaling algorithm is that it is hard to find potential functions that interact well with SETF. It would be interesting to provide an analysis of this algorithm.

References

- [1] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4), 2007.
- [2] N. Bansal, H.L. Chan, T.W. Lam, and L.K. Lee. Scheduling for bounded speed processors. In *Proc. of International Colloquium on Automata, Languages and Programming, ICALP*, pages 409 – 420, 2008.
- [3] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. submitted.
- [4] Nikhil Bansal, Kirk Pruhs, and Cliff Stein. Speed scaling for weighted flow time. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 805–813, 2007.
- [5] Luca Becchetti and Stefano Leonardi. Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. *J. ACM*, 51(4):517–539, 2004.
- [6] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Online weighted flow time and deadline scheduling. *J. Discrete Algorithms*, 4(3):339–352, 2006.
- [7] Jeff Edmonds. Scheduling in the dark. *Theor. Comput. Sci.*, 235(1):109–141, 2000.
- [8] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. Manuscript.
- [9] G. H. Hardy, J. E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, 1952.
- [10] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.
- [11] Bala Kalyanasundaram and Kirk Pruhs. Minimizing flow time nonclairvoyantly. *J. ACM*, 50(4):551–567, 2003.
- [12] T.W. Lam, L.K. Lee, Isaac To, and P. Wong. Speed scaling functions for flow time scheduling based on active job count. In *Proc. of European Symposium on Algorithms, ESA*, 2008, to appear.
- [13] Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theoretical Computer Science*, 130(1):17–47, 1994.
- [14] Kirk Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.
- [15] Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3), 2008.