Nonclairvoyant Speed Scaling for Flow and Energy

Ho-Leung Chan[*]   Jeff Edmonds[†]   Tak-Wah Lam[*]   Lap-Kei Lee[‡]
Alberto Marchetti-Spaccamela[§]   Kirk Pruhs[¶]

Abstract

We give three results related to online nonclairvoyant speed scaling to minimize total flow time plus energy. We give a nonclairvoyant algorithm LAPS, and show that for every power function of the form $P(s) = s^\alpha$, LAPS is $O(1)$-competitive; more precisely, the competitive ratio is 8 for $\alpha = 2$, 13 for $\alpha = 3$, and $\frac{2\alpha^2}{\ln \alpha}$ for $\alpha > 3$. We then show that there is no constant c, and no deterministic nonclairvoyant algorithm A, such that A is c-competitive for every power function of the form $P(s) = s^\alpha$. So necessarily the achievable competitive ratio increases as the steepness of the power function increases. Finally we show that there is a fixed, very steep, power function 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.

In this paper, we will consider the objective of minimizing a linear combination of total flow and total energy used. For a formal definition 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 unit 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 policies using competitive analysis.

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 the speed such that the power equals 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 $\alpha = 3$ 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 $O(1)$-competitive for each power function of the form $P(s) = s^\alpha$. More precisely, the competitive ratio is 8 for $\alpha = 2$, 13 for $\alpha = 3$, and $\frac{2\alpha^2}{\ln \alpha}$ for $\alpha > 3$. The potential function that we use is an amalgamation of the potential function used in [9] 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 2-competitive *clairvoyant* speed scaling policy [2, 4]. In contrast, we show in Section 3 that the competitiveness achievable by *nonclairvoyant* policies must depend on the steepness of the power function. We show that there is no constant c, and no deterministic nonclairvoyant algorithm A, such that A is c-competitive for every power function of the form $P(s) = s^\alpha$. And we show that there is a fixed, very steep, power function for which no nonclairvoyant algorithm can be $O(1)$-competitive. 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 [10, 13]. 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 times 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 [6, 11]. 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 [8].

Let us first consider the traditional model where the power function is $P(s) = s^\alpha$. Most of the literature assumes the *unbounded speed model*, in which a processor can be run at any real speed in the range $[0, \infty)$. In this 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. Culminating a line of several papers [1, 5, 12] on online algorithms for arbitrary work jobs, [4] showed that the speed scaling 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-competitive for the objective of total flow plus energy on arbitrary-work unit-weight jobs for essentially arbitrary power functions. This analysis was improved to show that this algorithm is 2-competitive in [2].

[4] further showed that 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. [5] then showed how to modify this algorithm to obtain an algorithm that is $O(1)$-competitive for (integral) weighted flow plus energy for power functions of the form $P(s) = s^\alpha$, using the known resource augmentation analysis of HDF [7].

[3] extended the result of [5] for the unbounded speed model to the bounded speed model, which was later improved by [12].

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*.

3

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 $E_A = \int_0^\infty (s_A(t))^\alpha dt$. 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, $F_{Opt(I)} + E_{Opt(I)}$ 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 $O(\frac{\alpha^2}{\log \alpha})$-competitive Algorithm

In this section, we give an online nonclairvoyant algorithm and show that it is $O(\frac{\alpha^2}{\log \alpha})$-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 has not yet completed by time t. Our algorithm is defined as follows.

Algorithm LAPS(β). Let $0 < \beta < 1$ be any real. At any time t, let $n_a(t)$ be the number of active jobs remaining in LAPS. Then LAPS runs at speed $(n_a(t))^{1/\alpha}$, and it processes the $\lceil \beta n_a(t) \rceil$ active jobs with the latest release times (ties broken by job identity) by splitting the processor speed equally among these jobs.

Our main result is the following.

Theorem 1. *For any $0 < \beta < 1$, LAPS(β) is c-competitive for total flow time plus energy, where $c = \max\{\frac{2}{(1-\beta)^\alpha}, \frac{2\alpha}{\beta}\}$.*

The proof of Theorem 1 is based to a large extent on the techniques from [9]. We start with some definitions and notations. Recall that $n_a(t)$ is the number of active jobs under LAPS at time t. Let $s_a(t) = (n_a(t))^{1/\alpha}$ which is the speed of LAPS at time t. Let

$$G_a(t) = F_a(t) + E_a(t) = \int_0^t n_a(x)dx + \int_0^t (s_a(x))^\alpha dx$$

be the total flow time plus energy incurred up to time t by LAPS. Note that $\frac{dG_a(t)}{dt} = n_a(t) + (s_a(t))^\alpha$ is its instantaneous cost at time t. Similarly, define $n_o(t)$, $s_o(t)$, and $G_o(t)$ for that of Opt. Since LAPS uses the common technique of balancing its flow and energy costs (e.g., as in [5, 12]), we have that

$$G_a(t) = \int_0^t n_a(x)dx + \int_0^t P(s_a(x))^\alpha dx = \int_0^t 2n_a(x)dx \ .$$

4

It follows that $\frac{dG_a(t)}{dt} = 2n_a(t)$. Opt may choose to set its speed differently.

We use an amortized local competitiveness argument (see for example [14]). To show that LAPS 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 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 or Opt, Φ does not increase.
- *Running condition:* At any other time, the rate of change of G_a plus the rate of change of Φ is no more than c times the rate of change of G_o, i.e., $\frac{dG_a(t)}{dt} + \frac{d\Phi(t)}{dt} \le c \cdot \frac{dG_o(t)}{dt}$, during any period of time without job arrival or completion.

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 and Opt, respectively. Let $\{j_1, \ldots, j_{n_a(t)}\}$ be the set of active jobs in LAPS, ordered by their release times such that $r(j_1) \le r(j_2) \le \cdots \le r(j_{n_a(t)})$. 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 $\gamma > 0$ is some constant that we will set later. 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 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 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.

Hence, it remains to check the running condition. In the following, we focus on a certain time t 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, j is a *lagging job* at time t if LAPS has processed less than Opt on j up to time t, i.e., $q_a(j) - q_o(j) > 0$. Let ℓ be the number of lagging jobs that LAPS is processing at time t. We start by evaluating $\frac{d\Phi}{dt}$.

Lemma 2. *At time t,*

$$\frac{d\Phi}{dt} \le \gamma \frac{1}{\alpha(1-\beta)^{\alpha-1}} s_o^\alpha + \gamma \frac{(1-\beta)}{\beta} n_o - \gamma \frac{1}{\alpha}(1-\beta) n_a$$

5

Proof. First consider the easier case when $n_a = 0$ at time t. Then the claim of the lemma becomes that $\frac{d\Phi}{dt} \le \gamma \frac{1}{\alpha(1-\beta)^{\alpha-1}} s_o^\alpha + \gamma \frac{(1-\beta)}{\beta} n_o$. Note that the right hand side is non-negative, and $\frac{d\Phi}{dt} = 0$ when $n_a = 0$. Hence the lemma is true when $n_a = 0$. The rest of the proof considers the case when $n_a > 0$.

We consider $\frac{d\Phi}{dt}$ as the combined effect due to the processing of LAPS and Opt. Let $\frac{d\Phi_1}{dt}$ and $\frac{d\Phi_2}{dt}$ be the rates of change in Φ due to Opt and LAPS, respectively. Then $\frac{d\Phi}{dt} = \frac{d\Phi_1}{dt} + \frac{d\Phi_2}{dt}$. For any job j, $q_o(j)$ is decreasing or remains the same. Hence, $\frac{d\Phi_1}{dt}$ is non-negative. Similarly, $\frac{d\Phi_2}{dt}$ is non-positive.

We first bound $\frac{d\Phi_1}{dt}$. The worst case is that Opt is processing the job with the largest coefficient, i.e., $n_a^{1-1/\alpha}$. Hence $\frac{d\Phi_1}{dt}$ 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, which is formally stated in Lemma 3, by setting $f(x) = (\frac{x}{1-\beta})^{\alpha-1}$, $f^{-1}(y) = (1-\beta)y^{1/(\alpha-1)}$, $g = s_o$ and $h = n_a^{1-1/\alpha}$. Then, we have

$$
\begin{aligned}
\gamma s_o n_a^{1-1/\alpha} &\le \gamma \left(\int_0^{s_o} (\frac{x}{1-\beta})^{\alpha-1} dx + \int_0^{n_a^{1-1/\alpha}} (1-\beta)y^{1/(\alpha-1)} dy \right) \\
&= \gamma \frac{1}{\alpha(1-\beta)^{\alpha-1}} s_o^\alpha + \gamma(1 - \frac{1}{\alpha})(1-\beta)n_a \ .
\end{aligned}
$$

We then bound $\frac{d\Phi_2}{dt}$. LAPS works on the $\lceil \beta n_a \rceil$ jobs with the latest arrival times. Ideally, for each of these jobs j, the term $\max\{0, q_a(j) - q_o(j)\}$ in the potential function decreases at a rate equal to the speed $\frac{s_a}{\lceil \beta n_a \rceil}$. Then $\frac{d\Phi_2}{dt}$ would be minimized. However, LAPS may be ahead of Opt on some jobs j, that is, $q_a(j) - q_o(j) \le 0$. For these non-lagging jobs, $\max\{0, q_a(j) - q_o(j)\}$ is zero and will not decrease. To address this issue, recall that ℓ is the number of lagging jobs LAPS is processing at time t. Let j be one of these lagging jobs. Then j is among the $\lceil \beta n_a \rceil$ active jobs with the latest release times and its coefficient is at least $(n_a - \lceil \beta n_a \rceil + 1)^{1-1/\alpha}$. Also, j is being processed at speed $\frac{s_a}{\lceil \beta n_a \rceil}$, so $q_a(j)$ is decreasing at this rate. It follows that $\frac{d\Phi_2}{dt}$ is at most

$$
\begin{aligned}
& \gamma \ell (n_a - \lceil \beta n_a \rceil + 1)^{1-1/\alpha} \left(\frac{-s_a}{\lceil \beta n_a \rceil} \right) \\
\le & \gamma \ell (n_a - \beta n_a)^{1-1/\alpha} \left(\frac{-s_a}{\lceil \beta n_a \rceil} \right) && (\text{since } -\lceil \beta n_a \rceil + 1 > -\beta n_a) \\
\le & \gamma \ell (1-\beta) \frac{-n_a}{\lceil \beta n_a \rceil} && (\text{since } s_a = n_a^{1/\alpha} \text{ and } (1-\beta)^{1-1/\alpha} > 1-\beta)
\end{aligned}
$$

Summing up $\frac{d\Phi_1}{dt}$ and $\frac{d\Phi_2}{dt}$, we obtain that

$$
\begin{aligned}
\frac{d\Phi}{dt} &\le \gamma \frac{1}{\alpha(1-\beta)^{\alpha-1}} s_o^\alpha + \gamma(1 - \frac{1}{\alpha})(1-\beta)n_a + \gamma \ell(1-\beta)\frac{-n_a}{\lceil \beta n_a \rceil} \\
&= \gamma \frac{1}{\alpha(1-\beta)^{\alpha-1}} s_o^\alpha + \gamma \frac{(1-\beta)n_a}{\lceil \beta n_a \rceil}(\lceil \beta n_a \rceil - \ell) - \gamma \frac{1}{\alpha}(1-\beta)n_a
\end{aligned}
$$

We observe that $\frac{n_a}{\lceil \beta n_a \rceil} \le \frac{n_a}{\beta n_a} = \frac{1}{\beta}$ and that the number of non-lagging jobs being processed

6

by LAPS, $\lceil\beta n_a\rceil - \ell$, is at most the number of jobs n_o unfinished by Opt. Thus, we have

$$\frac{d\Phi}{dt} \le \gamma \frac{1}{\alpha(1-\beta)^{\alpha-1}} s_o^\alpha + \gamma \frac{(1-\beta)}{\beta} n_o - \gamma \frac{1}{\alpha}(1-\beta)n_a \ . \qquad \Box$$

Below is the formal statement of Young's Inequality, which is used in the proof of Lemma 2.

Lemma 3 (Young's Inequality). *Let f be any real-valued, continuous and strictly increasing function such that $f(0) = 0$. Then, for all $g, h \ge 0$, $\int_0^g f(x)dx + \int_0^h f^{-1}(y)dy \ge gh$, where f^{-1} is the inverse function of f.*

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

Lemma 4. *Let $\gamma = \frac{2\alpha}{1-\beta}$ and $c = \max\{\frac{2}{(1-\beta)^\alpha}, \frac{2\alpha}{\beta}\}$. Then at time t, $\frac{dG_a}{dt} + \frac{d\Phi}{dt} \le c \cdot \frac{dG_o}{dt}$.*

Proof. Plugging $\gamma = \frac{2\alpha}{1-\beta}$ into Lemma 2, we obtain that

$$\frac{dG_a}{dt} + \frac{d\Phi}{dt} \le 2n_a + \frac{2}{(1-\beta)^\alpha} s_o^\alpha + \frac{2\alpha}{\beta} n_o - 2n_a \le c \cdot \frac{dG_o}{dt} \ . \qquad \Box$$

Combining Lemma 4 with the discussion on the boundary, job arrival and job completion conditions, Theorem 1 follows. Below we choose appropriate β to optimize the competitive ratio c for different values of α; in particular, for large α, c grows as $\Theta(\frac{\alpha^2}{\log \alpha})$.

Corollary 5. *If $\alpha = 2$, then $\mathrm{LAPS}(0.5)$ is 8-competitive. If $\alpha = 3$, then $\mathrm{LAPS}(0.46)$ is 13-competitive. If $\alpha > 3$ and $\beta = \frac{\ln \alpha}{\alpha}$, then $\mathrm{LAPS}(\beta)$ is $\frac{2\alpha^2}{\ln \alpha}$-competitive.*

Proof. If $\alpha = 2$ and $\beta = 0.5$, then c evaluates to 8. If $\alpha = 3$ and $\beta = 0.46$, then c evaluates to 13. We now show that if we pick $\beta = \frac{\ln \alpha}{\alpha}$, then for $\alpha > 3$, we have

$$c = \max\{\frac{2}{(1-\beta)^\alpha}, \frac{2\alpha}{\beta}\} \le \frac{2\alpha^2}{\ln \alpha} \ .$$

Since $\beta = \frac{\ln \alpha}{\alpha}$, we have that $\frac{2\alpha}{\beta} = \frac{2\alpha^2}{\ln \alpha}$. It remains to show that

$$\frac{2}{(1-\frac{\ln\alpha}{\alpha})^\alpha} \le \frac{2\alpha^2}{\ln \alpha} \ , \quad \text{or equivalently,} \quad \ln \alpha \le \alpha^2 (1 - \frac{\ln \alpha}{\alpha})^\alpha \ .$$

Observe that the function $\frac{\alpha}{\ln \alpha}$ is increasing with α and the function $(1 - \frac{1}{x})^x$ is increasing with x. Hence, for $\alpha > 3$, we have

$$(1 - \frac{\ln \alpha}{\alpha})^\alpha = (1 - \frac{1}{\frac{\alpha}{\ln \alpha}})^{\frac{\alpha}{\ln \alpha} \cdot \ln \alpha} \ge (1 - \frac{1}{\frac{3}{\ln 3}})^{\frac{3}{\ln 3} \cdot \ln \alpha} = 0.28^{\ln \alpha}$$

We take a loose bound that $0.28 > e^{-1.5}$, so $0.28^{\ln \alpha} > \alpha^{-1.5}$. Hence, $\alpha^2(1 - \frac{\ln \alpha}{\alpha})^\alpha > \alpha^{0.5}$. Finally, we can check easily (by differentiation) that $\alpha^{0.5} > \ln \alpha$ for $\alpha > 3$, and the corollary follows. $\qquad \Box$

7

3 Lower Bounds

In this section, we show that necessarily the competitive ratio achievable by a deterministic nonclairvoyant algorithm grows with α, and there is a sufficiently steep power function such that no deterministic nonclairvoyant algorithm can be $O(1)$-competitive.

Lemma 6. *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 every deterministic nonclairvoyant 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. We start with an intuitive roadmap of the proof, which is similar to the lower bound in [13] on the competitive ratio for nonclairvoyant algorithms on a fixed speed processor. Initially ALG is given n jobs at time 0. If ALG runs these jobs too fast, unnecessarily wasting too much energy, then this is the final instance that establishes the lower bound. Otherwise, at some later point in time, ALG intuitively learns that it has one unit of work left unprocessed on each job, while the adversary has n units of work left on one job. Further at this time, a stream of unit work job arises. After this time, the adversary can go at unit speed incurring a cost of $\Theta(1)$ per unit time. If ALG goes at an average speed of 1 or less, then ALG incurs a cost of $\Theta(n)$ per unit time for flow time. If ALG averages faster than speed 1, then by the steepness of the power function, it uses much more energy than optimal.

We now make this intuition more formal. Let $n = \lceil kP(v) \rceil$. We release n jobs $j_1, j_2, \ldots, j_n$ 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 $n^3 + n^2 P(1) \leq 2n^3$, so ALG is $\Omega(k)$-competitive.

The rest of the proof assumes $G(T) < kn^3$. Let $q_1, q_2, \ldots, q_n$ 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 with remaining processing time equal to 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 $j_1, \ldots, j_{n-1}$ 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 $G(T)$.

During $[T, T + n^4]$, Opt releases a stream of small jobs. Specifically, let $\epsilon < \frac{1}{n^5 v^2}$ be any real. For $i = 1, \ldots, \frac{n^4}{\epsilon}$, 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 two jobs (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 $2n^4 + n^4 P(v)$. Opt can complete j_n by running at speed 1 during $[T + n^4, T + n^4 + n]$, incurring a cost of $n + nP(1)$. Since $G(T) < kn^3$ then the total flow time plus energy of Opt for the whole job sequence is at most

$$kn^3 + 2n^4 + n^4 P(v) + n + nP(1) = O(n^4 P(v)) .$$

For ALG, we first show that its total work done on the small jobs during $[T, T + n^4]$ is at least $n^4 v - 1$. Otherwise, there are at least $\frac{1}{\epsilon v} > n^5 v$ 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 $j_1, \ldots, j_n$ *big* jobs. If ALG completes less that $\frac{1}{2}n + 1$ 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 $n^4 v - 1 + \frac{1}{2}n + 1$. The total energy used by ALG is at least

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

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. $\qquad \square$

We can apply Lemma 6 to obtain the lower bound for the power function $P(s) = s^\alpha$.

Theorem 7. *For every deterministic nonclairvoyant algorithm, and every constant c, this algorithm is not c-competitive for every power function of the form $P(s) = s^\alpha$.*

Proof. We look at the competitive ratio of an arbitrary nonclairvoyant algorithm as α grows. We show that the competitive ratio is $\Omega(\alpha^{1/3-\epsilon})$, for every $0 < \epsilon < 1/3$. We apply Lemma 6 by putting $k = \alpha^{1/3-\epsilon}$ and $v = 1$. Then, $P(v) = 1$ and

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

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, $\min\{k, P(v + \frac{1}{16(kP(v))^3})/P(v)\} \geq \min\{\alpha^{1/3-\epsilon}, (\frac{17}{16})^{\alpha^{3\epsilon}}\} = \Omega(\alpha^{1/3-\epsilon})$. $\qquad \square$

Theorem 8. *There exists some power function P such that every nonclairvoyant deterministic 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 6, 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,

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

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

9

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 cannot be $O(1)$-competitive.

The standard and the best nonclairvoyant job selection policy for a fixed speed processor is Shortest Elapsed Time First (SETF). The most obvious candidate speed scaling policy would be to use SETF for job selection, and to run at power equal to the number of active jobs (or perhaps a bit faster). 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] Lachlan L.H. Andrew, Adam Wierman, and Ao Tang. Optimal speed scaling under arbitrary power functions. *SIGMETRICS Performance Evaluation Review*, 37(2):39–41, 2009.

[3] Nikhil Bansal, Ho-Leung Chan, Tak-Wah Lam, and Lap-Kei Lee. Scheduling for bounded speed processors. In *International Colloquium on Automata, Languages and Programming*, pages 409 – 420, 2008.

[4] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *ACM-SIAM Syposium on Discrete Algorithms*, pages 693–701, 2009.

[5] Nikhil Bansal, Kirk Pruhs, and Cliff Stein. Speed scaling for weighted flow time. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 805–813, 2007.

[6] Luca Becchetti and Stefano Leonardi. Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. *Journal of the ACM*, 51(4):517–539, 2004.

[7] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Online weighted flow time and deadline scheduling. *Journal of Discrete Algorithms*, 4(3):339–352, 2006.

[8] Jeff Edmonds. Scheduling in the dark. *Theoretical Computer Science*, 235(1):109–141, 2000.

[9] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 685–692, 2009.

[10] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[11] Bala Kalyanasundaram and Kirk Pruhs. Minimizing flow time nonclairvoyantly. *Journal of the ACM*, 50(4):551–567, 2003.

[12] Tak-Wah Lam, Lap-Kei Lee, Isaac K. K. To, and Prudence W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *European Symposium on Algorithms*, pages 647–659, 2008.

[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.