Improved Bounds for Speed Scaling in Devices Obeying the Cube-Root Rule

Nikhil Bansal[1], Ho-Leung Chan[2], Kirk Pruhs[3], and Dmitriy Katz[4] *

[1] IBM T.J. Watson, Yorktown Heights, NY, nikhil@us.ibm.com
[2] Max-Planck-Institut für Informatik, hlchan@mpi-inf.mpg.de
[3] Computer Science Dept., Univ. of Pittsburgh, kirk@cs.pitt.edu
[4] IBM T.J. Watson, Yorktown Heights, NY, dkatzrog@us.ibm.com

Abstract. Speed scaling is a power management technique that involves dynamically changing the speed of a processor. This gives rise to dual-objective scheduling problems, where the operating system both wants to conserve energy and optimize some Quality of Service (QoS) measure of the resulting schedule. In the most investigated speed scaling problem in the literature, the QoS constraint is deadline feasibility, and the objective is to minimize the energy used. The standard assumption is that the power consumption is the speed to some constant power α. We give the first non-trivial lower bound, namely $e^{\alpha-1}/\alpha$, on the competitive ratio for this problem. This comes close to the best upper bound which is about $2e^{\alpha+1}$.

We analyze a natural class of algorithms called qOA, where at any time, the processor works at $q \geq 1$ times the minimum speed required to ensure feasibility assuming no new jobs arrive. For CMOS based processors, and many other types of devices, $\alpha = 3$, that is, they satisfy the cube-root rule. When $\alpha = 3$, we show that qOA is 6.7-competitive, improving upon the previous best guarantee of 27 achieved by the algorithm Optimal Available (OA). So when the cube-root rule holds, our results reduce the range for the optimal competitive ratio from $[1.2, 27]$ to $[2.4, 6.7]$. We also analyze qOA for general α and give almost matching upper and lower bounds.

1 Introduction

Current processors produced by Intel and AMD allow the speed of the processor to be changed dynamically. Intel's SpeedStep and AMD's PowerNOW technologies allow the Windows XP operating system to dynamically change the speed of such a processor to conserve energy. 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. All theoretical studies we know of assume a speed to power function $P(s) = s^\alpha$, where s is the

* The work of H.L.Chan was done when he was a postdoc in University of Pittsburgh. K. Pruhs was supported in part by NSF grants CNS-0325353, CCF-0514058, IIS-0534531, and CCF-0830558, and an IBM Faculty Award.

speed and $\alpha > 1$ is some constant. Energy consumption 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.

The first theoretical study of speed scaling algorithms was in the seminal paper [16] by Yao, Demers, and Shenker. In the problem introduced in [16] the QoS objective was deadline feasibility, and the objective was to minimize the energy used. To date, this is the most investigated speed scaling problem in the literature [16, 2, 9, 6, 5, 14, 17, 12, 1, 11]. In this problem, each job i has a release time r_i when it arrives in the system, a work requirement w_i, and a deadline d_i by which the job must be finished. The deadlines might come from the application, or might arise from the system imposing a worst-case quality-of-service metric, such as maximum response time or maximum slow-down. It is clear that an optimal job selection policy is Earliest Deadline First (EDF). Thus the remaining issue is to find an online speed scaling policy to minimize energy.

1.1 The Story to Date

Yao, Demers and Shenker showed that the optimal offline schedule can be efficiently computed by a greedy algorithm [16]. [16] proposed two natural online speed scaling algorithms, Average Rate (AVR) and Optimal Available (OA). Conceptually, AVR is oblivious in that it runs each job in the way that would be optimal if there were no other jobs in the system. That is, AVR runs each job i (in parallel with other jobs) at the constant speed $w_i/(d_i - r_i)$ throughout interval $[r_i, d_i]$. The algorithm OA maintains the invariant that the speed at each time is optimal given the current state, and under the assumption that no more jobs will arrive in the future. In particular, let $w(x)$ denote the amount of unfinished work that has deadline within x time units from the current time. Then the current speed of OA is $\max_x w(x)/x$. Another online algorithm BKP is proposed in [5]. BKP runs at speed $e \cdot v(t)$ at time t, where $v(t) = \max_{t'>t} w(t, et - (e-1)t', t')/(e(t'-t))$ and $w(t, t_1, t_2)$ is the amount of work that has release time at least t_1, deadline at most t_2, and that has already arrived by time t. Clearly, if $w(t_1, t_2)$ is the total work of jobs that are released after t_1 and have deadline before t_2, then any algorithm must have an average speed of at least $w(t_1, t_2)/(t_2 - t_1)$ during $[t_1, t_2]$. Thus BKP can be viewed as computing a lower bound on the average speed in an online manner and running at e times that speed.

Table 1 summarizes the previous results. The competitive ratio of AVR is at most $2^{\alpha-1}\alpha^\alpha$. This was first shown in [16], and a simpler amortized local competitiveness analysis was given in [2]. The competitive ratio of AVR is least $(2-\delta)^{\alpha-1}\alpha^\alpha$, where δ is a function of α that approaches zero as α approaches infinity [2]. The competitive ratio of OA is exactly α^α [5], where the upper bound is proved using an amortized local competitiveness argument. Thus the competitive ratio of AVR is strictly inferior to that of OA. The competitive ratio of BKP is at most $2(\alpha/(\alpha-1))^\alpha e^\alpha$ [5], which is about $2e^{\alpha+1}$ for large α . It is better than that of OA only for $\alpha \geq 5$. On the other hand, the lower bounds for

Previous Results

Algorithm	General α		$\alpha = 2$		$\alpha = 3$	
	Upper	Lower	Upper	Lower	Upper	Lower
General		$\left(\frac{4}{3}\right)^{\alpha}/2$		1.1		1.2
AVR	$2^{\alpha-1}\alpha^{\alpha}$	$(2-\delta)^{\alpha-1}\alpha^{\alpha}$	8	4	108	48.2
OA	α^{α}	α^{α}	4	4	27	27
BKP	$2(\alpha/(\alpha-1))^{\alpha}e^{\alpha}$		59.1		135.6	

Our Contributions

Algorithm	General α		$\alpha = 2$		$\alpha = 3$	
	Upper	Lower	Upper	Lower	Upper	Lower
General		$e^{\alpha-1}/\alpha$		1.3		2.4
qOA	$4^{\alpha}/(2\sqrt{e\alpha})$	$\frac{1}{4\alpha}4^{\alpha}(1-\frac{2}{\alpha})^{\alpha/2}$	2.4		6.7	

Table 1. Results on the competitive ratio for energy minimization with deadline feasibility.

general algorithms are rather weak. Somewhat surprisingly, the best known lower bound instance is the worst-possible instance consisting of two jobs. [4] shows a lower bound of $\left(\frac{4}{3}\right)^{\alpha}/2$ on the competitive ratio using a two job instance. If one tries to find the worst 3, 4, ... job instances, the calculations get messy quickly.

The most interesting value of α seems to be three. Most importantly, 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 [8]. The power is also roughly proportional to the cube of the speed in many common devices/machines, such as vehicles/automobiles, and some types of motors. It seems likely that α would be in the range $[2, 3]$ for most conceivable devices. The best known guarantee for α in this range is α^{α} achieved by OA, which evaluates to 4 for $\alpha = 2$ and 27 for $\alpha = 3$. Our motivating goal is to focus on the case that $\alpha = 3$, and to a lesser extent on $\alpha = 2$, and to obtain better algorithms and lower bounds in these cases.

1.2 Our Contributions

We show, using an amortized local competitiveness analysis, that if q is set to $2 - \frac{1}{\alpha}$, then the competitive ratio of qOA is at most $4^{\alpha}/(2\sqrt{e\alpha})$. This bound is approximately 3.4 when $\alpha = 2$, and 11.2 when $\alpha = 3$. Using an analysis specialized to the specific cases that $\alpha = 2$ and $\alpha = 3$, we show that qOA is at worst 2.4-competitive when $\alpha = 2$, and at worst 6.7-competitive when $\alpha = 3$.

Our main technical idea is to introduce a new potential function which is quite different from the one used in the analysis of OA in [5] (and the potential function used to analyze AVR in [2]). This is necessary, since potential functions similar to those used earlier cannot yield guarantees of the form c^{α} where c is independent of α. The potential function we use is more similar to the one used in [7] to analyze a speed scaling algorithm for the (different) objective of minimizing flow time plus energy. However, here we will need a different analysis approach. The analysis in [7], and almost all of the amortized local competitiveness analyses in the speed scaling literature, rely critically on the Young's inequality. However, in the current setting, Young's inequality gives a bound that is too weak to be useful when analyzing qOA. The key insight that allows us to avoid the use of Young's inequality was to observe that certain expressions that arise in the analysis are convex, which allows us to reduce the analysis of the general case down to just two extreme cases. To the best of our knowledge, this convexity technique can replace all of the uses of Young's inequality in the speed scaling literature. In all cases, the resulting bound that one obtains using this convexity technique is at least as good as the bound that one obtains using Young's inequality, and the resulting proof is simpler and more intuitive. In some cases, this convexity technique gives a better bound. For example, if one applies this convexity technique to the analysis of the LAPS algorithm in [10], one obtains a bound on the competitive ratio of $O(\alpha^2/\log^2\alpha)$, whereas using Young's technique one can only get a bound of $O(\alpha^3)$.

In Section 4 we consider lower bounds. We give the first non-trivial lower bound on the competitive ratio for any algorithm. We show that every deterministic algorithm must have a competitive ratio of at least $e^{\alpha-1}/\alpha$. The base of the exponent, e, is the best possible since BKP achieves a ratio of about $2e^{\alpha+1}$. For $\alpha = 3$, this raises the best known lower bound a modest amount, from 1.2 to 2.4. The instance is identical to the one used in [5] to lower bound the competitive ratio with respect to the objective of minimizing the maximum speed. The innovation required to get a lower bound for energy is to categorize the variety of possible speed scaling policies in such a way that one can effectively reason about them.

Given the general lower bound of $e^{\alpha-1}/\alpha$, and that BKP achieves a ratio with base of exponent e, a natural question is whether there is some choice of the parameter q for which the competitive ratio of qOA varies with e as the base of the exponent. Somewhat surprisingly, we show that this is not the case and the base of the exponent cannot be improved beyond 4. In particular, we show that the competitive ratio of qOA is at least $\frac{1}{4\alpha}4^\alpha(1-\frac{2}{\alpha})^{\alpha/2}$. We note that this lower bound is quite close to our upper bound for qOA, especially as α increases.

Our results are summarized in the last two rows of table 1. In particular we give asymptotically matching upper and lower bounds for qOA and reduce the range for the optimal competitive ratio in the case that the cube-root rule holds from $[1.2, 27]$ to $[2.4, 6.7]$ and in the case that $\alpha = 2$ from $[1.1, 4]$ (obtained in [16]) to $[1.3, 2.4]$. Due to the limitation of space, some proofs are omitted and will be given in the full paper.

1.3 Other Related Results

There are now enough speed scaling papers in the literature that it is not practical to survey all such papers here. We limit ourselves to those papers most related to the results presented here.

A naive implementation of YDS runs in time $O(n^3)$. This can be improved to $O(n^2)$ if the intervals have a tree structure [12]. Li, Yao and Yao [13] gave an implementation that runs in $O(n^2 \log n)$ time for the general case. For hard real-time jobs with fixed priorities, Yun and Kim [17] showed that it is NP-hard to compute a minimum-energy schedule. They also gave a fully polynomial time approximation scheme for the problem. Kwon and Kim [11] gave a polynomial time algorithm for the case of a processor with discrete speeds. Li and Yao [14] gave an algorithm with running time $O(d \cdot n \log n)$ where d is the number of speeds. A simpler algorithm with this running time can be found in [13].

Albers, Müller, and Schmelzer [1] consider the problem of finding energy-efficient deadline-feasible schedules on multiprocessors. [1] showed that the offline problem is NP-hard, and gave $O(1)$-approximation algorithms. [1] also gave online algorithms that are $O(1)$-competitive when job deadlines occur in the same order as their release times. Chan et al. [9] considered the more general and realistic speed scaling setting where there is an upper bound on the maximum processor speed. They gave an $O(1)$-competitive algorithm based on OA. Recently, Bansal, Chan and Pruhs [3] investigated speed scaling for deadline feasibility in devices with a regenerative energy source such as a solar cell.

2 Formal Problem Statement

A problem instance consists of n jobs. Job i has a release time r_i, a deadline $d_i > r_i$, and work $w_i > 0$. In the online version of the problem, the scheduler learns about a job only at its release time; at this time, the scheduler also learns the exact work requirement and the deadline of the job. We assume that time is continuous. A schedule specifies for each time a job to be run and a speed at which to run the job. The speed is the amount of work performed on the job per unit time. A job with work w run at a constant speed s thus takes $\frac{w}{s}$ time to complete. More generally, the work done on a job during a time period is the integral over that time period of the speed at which the job is run. A schedule is *feasible* if for each job i, work at least w_i is done on job i during $[r_i, d_i]$. Note that the times at which work is performed on job i do not have to be contiguous. If the processor is run at speed s, then the power is $P(s) = s^\alpha$ for some constant $\alpha > 1$. The energy used during a time period is the integral of the power over that time period. Our objective is to minimize the total energy used by the schedule. An algorithm A is said to be c-competitive if for any job sequence, the energy usage of A is at most c times that of the optimal schedule.

3 Upper Bound Analysis of qOA

Our goal in this section is to show that qOA is about $4^\alpha/(2\sqrt{e\alpha})$-competitive when $q = 2 - (1/\alpha)$. We wish to point out that $q = 2 - 1/\alpha$ is not necessarily the optimum value of q. For general α it is not clear how to obtain the optimum choice of q since it involves solving a system of high degree algebraic inequalities. However, the lower bound for qOA will imply that the choice $q = 2 - 1/\alpha$ is close to optimum. For the case of $\alpha = 3$ and that of $\alpha = 2$, we can explicitly determine the optimum choice of q which gives better competitive ratios for these cases.

We use an amortized local competitiveness analysis, and use a potential function $\Phi(t)$ that is a function of time. In this setting, the value of $\Phi(t)$ will be energy, and thus, the derivative of $\Phi(t)$ with respect to time will be power. We need that Φ is initially and finally zero. Let s_a and s_o be the current speed of the online algorithm (qOA in our case) and the optimal algorithm OPT respectively. Then in order to establish that the online algorithm is c-competitive, it is sufficient to show that the following key equation holds at all times:

$$s_a^\alpha + \frac{d\Phi}{dt} \leq c \cdot s_o^\alpha \tag{1}$$

The fact that equation (1) establishes c-competitiveness follows by integrating this equation over time, and from the fact that Φ is initially and finally 0. For more information on amortized local competitiveness arguments see [15].

Before defining the potential function Φ that we use, we need to introduce some notation. We always denote the current time as t_0. For any $t_0 \leq t' \leq t''$, let $w_a(t', t'')$ denote the total amount of work remaining in qOA at t_0 with deadline in $(t', t'']$. Define $w_o(t', t'')$ similarly for OPT. Recall that qOA runs at speed $q \cdot \max_t w_a(t_0, t)/(t - t_0)$, which is q times the speed that OA would run. Let $d(t', t'') = \max\{0, w_a(t', t'') - w_o(t', t'')\}$, denote the amount of additional work left under the online algorithm that has deadline in $(t', t'']$. We define a sequence of time points $t_1 < t_2 < \ldots$ iteratively as follows: Let t_1 be the time such that $d(t_0, t_1)/(t_1 - t_0)$ is maximized. If there are several such points, we choose the furthest one. Given t_i, let $t_{i+1} > t_i$ be the furthest point that maximizes $d(t_i, t_{i+1})/(t_{i+1} - t_i)$. We use g_i to denote $d(t_i, t_{i+1})/(t_{i+1} - t_i)$. Note that g_i is a non-negative monotonically decreasing sequence.

We first bound the offline and online speed, which will be useful in our analysis:

Lemma 1. **(i)** *$s_o \geq \max_t w_o(t_0, t)/(t - t_0)$.* **(ii)** *$s_a \geq qg_0$ and $s_a \leq qg_0 + qs_o$.*

We are now ready to define the potential function Φ that we use in our analysis of qOA:

$$\Phi = \beta \sum_{i=0}^{\infty} \left((t_{i+1} - t_i) \cdot g_i^\alpha\right) \quad ,$$

where β is some constant (which will be set to $q^\alpha(1 + \alpha^{-1/(\alpha-1)})^{\alpha-1}$).

We now make some observations about the potential function Φ. Φ is obviously zero before any jobs are released, and after the last deadline. Job arrivals do not affect Φ since $d(t', t'')$ does not change upon a job arrival for any t' and t''. Similarly, job completions by either qOA or optimal do not change Φ since it is a continuous function of the unfinished work, and the unfinished work on a job continuously decreases to 0 as it completes. Finally, structural changes in the t_i and g_i do not change the value of Φ. In particular, if g_0 decreases (for instance if online is working faster than offline on jobs with deadline in $[t_0, t_1]$), then at some point g_0 becomes equal to g_1, and the intervals $[t_0, t_1]$ and $[t_1, t_2]$ merge together. Upon this merge, the potential does not change as $g_0 = g_1$ at this point. Similarly, if offline works too fast, the interval $[t_k, t_{k+1}]$ (which contains the earliest deadline among the unfinished jobs under offline) might split into two critical intervals, $[t_k, t']$ and $[t', t_{k+1}]$, but again this change does not affect Φ since at the time of splitting, the value of g for the newly formed intervals is identical to the value of the interval $[t_k, t_{k+1}]$

Thus to complete our analysis, we are left to show the following lemma:

Lemma 2. *For general $\alpha > 1$, set $q = 2 - (1/\alpha)$, $\beta = c = (2 - (1/\alpha))^\alpha (1 + \alpha^{-1/(\alpha-1)})^{\alpha-1}$. Consider a time t where no jobs are released, no jobs are completed by* qOA *or optimal, and there are no structural changes to the t_i's nor g_i's. Then equation (1), $s_a^\alpha + d\Phi/dt - c \cdot s_o^\alpha \le 0$, holds at time t.*

Proof. Suppose first that $w_a(t_0, t_1) < w_o(t_0, t_1)$. In this case, $d(t_0, t_1) = 0$, $g_0 = 0$ and t_1 is basically infinity. Note that $d\Phi/dt = 0$ since Φ remains zero until $w_a(t_0, t_1) \ge w_o(t_0, t_1)$. Therefore, $s_a^\alpha + d\Phi/dt - c \cdot s_o^\alpha \le 0$ because $s_a \le q s_o$ and $c = q^\alpha (1 + \alpha^{-1/(\alpha-1)})^{\alpha-1} > q^\alpha$.

Hence we assume $w_a(t_0, t_1) \ge w_o(t_0, t_1)$ in the following. Without loss of generality, both OPT and qOA schedule jobs according to Earliest Deadline First, and hence qOA is working on a job with deadline at most t_1. Let t' be deadline of the job that OPT is working on, and let k be such that $t_k < t' \le t_{k+1}$.

First consider the case that $k > 0$. When both qOA and OPT work, g_0 decreases, the quantities $g_1, \ldots, g_{k-1}$, and $g_{k+1}, \ldots$ stay unchanged, and g_k increases. Note that $(t_1 - t_0)$ is decreasing, and the rate of decrease is the same as the rate that time passes. Therefore, the rate of change of $(t_1 - t_0) \cdot g_0^\alpha$ is

$$\begin{aligned}\frac{d}{dt_0}((t_1 - t_0) \cdot g_0^\alpha) &= (t_1 - t_0) \cdot \alpha g_0^{\alpha-1} \left(\frac{(t_1 - t_0)(-s_a) + d(t_0, t_1)}{(t_1 - t_0)^2} \right) - g_0^\alpha \\ &= \alpha g_0^{\alpha-1}(-s_a) + (\alpha - 1) g_0^\alpha\end{aligned}$$

For the rate of change of $(t_{k+1} - t_k) \cdot g_k^\alpha$, we note that $t_{k+1} - t_k$ stays unchanged. Also, the rate of change of $d(t_k, t_{k+1})$ may be $-s_o$ or 0, depending on whether $w_a(t_k, t_{k+1})$ is greater than $w_o(k_k, t_{k+1})$. Therefore,

$$\begin{aligned}\frac{d}{dt_0}((t_{k+1} - t_k) \cdot g_k^\alpha) &\le (t_{k+1} - t_k) \cdot \alpha g_k^{\alpha-1} \left(\frac{(t_{k+1} - t_k)(s_o)}{(t_{k+1} - t_k)^2} \right) \\ &= \alpha g_k^{\alpha-1}(s_o) \quad \le \quad \alpha g_0^{\alpha-1}(s_o)\end{aligned}$$

Thus to show $s_a^\alpha + d\Phi/dt - c \cdot s_o^\alpha \le 0$, it suffices to show that

$$s_a^\alpha + \beta(\alpha g_0^{\alpha-1}(-s_a + s_o) + (\alpha - 1) g_0^\alpha) - c \cdot s_o^\alpha \le 0. \tag{2}$$

Now consider the case that $k = 0$. Note that for $i \ge 1$, neither g_i nor $t_{i+1} - t_i$ changes, so we need not consider these terms in the potential function. The rate of change of $(t_1 - t_0) \cdot g_o^\alpha$ is

$$\begin{aligned}\frac{d}{dt_0}((t_1 - t_0) \cdot g_0^\alpha) &= (t_1 - t_0) \cdot \alpha g_0^{\alpha-1} \cdot \left(\frac{(t_1 - t_0)(-s_a + s_o) + d(t_0, t_1)}{(t_1 - t_0)^2} \right) - g_0^\alpha \\ &= \alpha g_0^{\alpha-1}(-s_a + s_o) + (\alpha - 1) g_0^\alpha\end{aligned}$$

which leads to the same inequality as equation (2).

Hence, we will focus on equation (2), and show that it is true for the stated values of q, c and β. We consider the left hand side of equation (2) as a function of s_a while g and s_o are fixed. Note that it is a convex function of s_a. Since $s_a \in [qg_0, qg_0 + qs_o]$, it suffices to show that equation (2) holds at the endpoints $s_a = qg_0$ and $s_a = qg_0 + qs_o$.

If $s_a = qg_0$, the left hand side of equation (2) becomes

$$\begin{gathered} q^\alpha g_0^\alpha - \beta q \alpha g_0^\alpha + \beta \alpha g_0^{\alpha-1} s_o + \beta(\alpha - 1) g_0^\alpha - c s_o^\alpha = \\ (q^\alpha - \beta \alpha q + \beta(\alpha - 1)) g_0^\alpha + \beta \alpha g_0^{\alpha-1} s_o - c s_o^\alpha \end{gathered}$$

Taking derivative with respect to s_o, we get that this is maximized at s_o satisfying $c s_o^{\alpha-1} = \beta g_0^{\alpha-1}$, and hence $s_o = \left(\frac{\beta}{c}\right)^{1/(\alpha-1)} g_0$. Substituting this for s_o and canceling g_0^α, it follows that we need to satisfy the following equation:

$$(q^\alpha - \beta \alpha q + \beta(\alpha - 1)) + \beta(\alpha - 1) \left(\frac{\beta}{c}\right)^{1/(\alpha-1)} \leq 0. \tag{3}$$

If $s_a = qg_0 + qs_o$, the left hand side of equation (2) becomes

$$\begin{aligned} &q^\alpha (g_0 + s_o)^\alpha - \beta q \alpha (g_0 + s_o) g_0^{\alpha-1} + \beta \alpha g_0^{\alpha-1} s_o + \beta(\alpha - 1) g_0^\alpha - c s_o^\alpha \\ &= q^\alpha (g_0 + s_o)^\alpha - \beta(q\alpha - (\alpha - 1)) g_0^\alpha - \beta \alpha (q - 1) g_0^{\alpha-1} s_o - c s_o^\alpha \end{aligned}$$

Setting $s_o = x \cdot g_0$ and canceling g_0^α, it follows that we need to satisfy

$$q^\alpha (1 + x)^\alpha - \beta(q\alpha - (\alpha - 1)) - \beta \alpha (q - 1) x - c x^\alpha \leq 0. \tag{4}$$

We set $q = 2 - (1/\alpha)$ and $\beta = c = q^\alpha \eta^{\alpha-1}$ where $\eta = 1 + \alpha^{-1/(\alpha-1)}$. With these choices of q, β and c, $\alpha q = 2\alpha - 1$, and to establish equation (3) it is sufficient to show that $q^\alpha - \beta \leq 0$, which is trivially true since $\eta > 1$. Similarly, equation (4) is equivalent to $(1 + x)^\alpha - \alpha \eta^{\alpha-1} - \eta^{\alpha-1}(\alpha - 1)x - \eta^{\alpha-1} x^\alpha \leq 0$ for all $x \geq 0$. Since $\alpha \geq 1$, it suffices to show that

$$(1 + x)^\alpha - \alpha \eta^{\alpha-1} - \eta^{\alpha-1} x^\alpha \leq 0. \tag{5}$$

To see this, note that if we take the derivative of the left side of equation (5), we obtain that the maximum is attained at x such that $(1 + x)^{\alpha-1} - \eta^{\alpha-1} x^{\alpha-1} = 0$ and hence $x = 1/(\eta - 1)$. For this value of x, the left side of equation (5) evaluates to 0 and hence the result follows. Hence equation (2) is satisfied and the lemma follows. □

Now our main theorem follows as a direct consequence.

Theorem 1. qOA *is* $(2-\frac{1}{\alpha})^\alpha(1+\alpha^{-1/(\alpha-1)})^{\alpha-1}$*-competitive for general* $\alpha > 1$*.*

Note that for large values of α, this bound on the competitive ratio of qOA is approximately $4^\alpha/(2\sqrt{e\alpha})$. For $\alpha = 3$ this bound on the competitive ratio of qOA evaluates to $(5/3)^3(1+1/\sqrt{3})^2 \approx 11.52$ (which is already better than the best known bound of 27). However, for the cases of $\alpha = 2$ and $\alpha = 3$, we can determine the optimum values of q and β to obtain Theorems 2 and 3.

Theorem 2. *If* $q = 1.54$*, then* qOA *is 6.73-competitive for* $\alpha = 3$*.*

Proof. We follow the same proof structure as that for Lemma 2 to obtain the inequalities (3) and (4). By putting $\alpha = 3$, it follows that we need to satisfy:

$$(q^3 - 3\beta q + 2\beta) + 2\beta\left(\frac{\beta}{c}\right)^{1/2} \leq 0$$
$$q^3(1+x)^3 - \beta(3q-2) - 3\beta(q-1)x - cx^3 \leq 0$$

We wrote a program to determine the values of q and β that minimize c. The best values we obtained are $q = 1.54, \beta = 7.78$ and $c = 6.73$. It is easy to check that the first inequality is satisfied. The left hand side of the second inequality becomes $-3.08x^3 + 10.96x^2 - 1.65x - 16.73$, which can be shown to be negative by differentiation. Hence (3) and (4) are satisfied and the theorem follows. □

Theorem 3. *If* $q = 1.46$ *and* $\beta = 2.7$*, then* qOA *is 2.391-competitive for* $\alpha = 2$*.*

4 Lower Bounds

In this section, we show that any algorithm is at least $\frac{1}{\alpha}e^{\alpha-1}$-competitive. Note that we assume α is fixed and is known to the algorithm. We first give an adversarial strategy for constructing a job instance such that any algorithm uses at least $\frac{1}{\alpha}e^{\alpha-1}$ times the energy of the optimal.

Adversarial Strategy: Let $\epsilon > 0$ be some small fixed constant. Work is arriving during $[0,\ell]$, where $0 < \ell \leq 1-\epsilon$. The rate of work arriving at time $t \in [0,\ell]$ is

$$a(t) = \frac{1}{1-t}$$

So the work that arrives during any time interval $[u,v]$ is $\int_u^v a(t)dt$. All work has deadline 1. Let A be any online algorithm. The value of ℓ will be set by the adversary according to the action of A. Intuitively, if A spends too much energy initially, then ℓ will be set to be small. If A doesn't spend enough energy early on, then ℓ will be set to $1-\epsilon$. In this case, A will have a lot of work left toward the end and will have to spend too much energy finishing this work off. To make this more formal, consider the function

$$E(t) = \int_0^t \left((1+\frac{b}{\ln\epsilon})\frac{1}{1-x}\right)^\alpha dx \ ,$$

where b is a constant (set to $\frac{1}{(\alpha-1)^{1/\alpha}}$ later). This is the total energy usage up to time t if A runs at speed $s(t) = (1 + \frac{b}{\ln \epsilon})\frac{1}{1-t}$. Of course, A may run at speed other than $s(t)$. If there is a first time $0 < h \le 1-\epsilon$ such that total energy usage of A up to h is at least $E(h)$, then the value of ℓ is set to h. If no such event occurs, then $\ell = 1 - \epsilon$. ■

In Lemma 5 we show that if the adversary ends the arrival of work at some time $0 < h \le 1 - \epsilon$ because the total energy usage of A is at least $E(h)$, then A must have used at least $\frac{1}{\alpha}e^{\alpha-1}$ times as much energy as optimal. Similarly, in Lemma 7, we show that if the adversary doesn't end the arrival of work until the time $1-\epsilon$, then the online algorithm uses at least $\frac{1}{\alpha}e^{\alpha-1}$ times as much energy as optimal. Then our main result, that any algorithm is at least $\frac{1}{\alpha}e^{\alpha-1}$-competitive, follows immediately. We start with two technical lemmas.

Lemma 3. *For any* $0 < h \le 1-\frac{1}{e}$, $(-\ln(1-h))^\alpha \le \frac{\alpha}{e^{\alpha-1}}(\frac{1}{(\alpha-1)(1-h)^{\alpha-1}} - \frac{1}{\alpha-1})$.

Lemma 4. *Let* $s_1(t)$ *and* $s_2(t)$ *be non-negative functions, and let* $\alpha > 1$ *and* $x > 0$ *be some real numbers. If* $s_2(t)$ *is continuous and monotonically increasing and if* $\int_0^y (s_1(t)^\alpha - s_2(t)^\alpha)dt < 0$ *for all* $0 < y \le x$*, then* $\int_0^x (s_1(t) - s_2(t))dt < 0$.

Lemma 5. *If there is a time* $0 < h \le 1 - \epsilon$ *such that the total energy usage of* A *is at least* $E(h)$*, then* A *is at least* $\frac{1}{\alpha}e^{\alpha-1}$*-competitive.*

Proof. Let E_A be the total energy usage of A. Then,

$$E_A \ge E(h) = \int_0^h \left((1+\frac{b}{\ln\epsilon})\frac{1}{1-x}\right)^\alpha dx = (1+\frac{b}{\ln\epsilon})^\alpha(\frac{1}{(\alpha-1)(1-h)^{\alpha-1}} - \frac{1}{\alpha-1})$$

Let E_{opt} be the energy usage of the optimal algorithm. There are two cases for the value of E_{opt}: (i) $0 < h \le 1 - \frac{1}{e}$ and (ii) $1 - \frac{1}{e} < h \le 1 - \epsilon$.

(i) If $0 < h \le 1 - \frac{1}{e}$, the total amount of work released is $\int_0^h \frac{1}{1-x}dx = -\ln(1-h) \le 1$. Thus, the optimal algorithm can run at speed $-\ln(1-h)$ throughout $[0, 1]$ to completes all work. Then

$$E_{opt} = \big(-\ln(1-h)\big)^\alpha \le \frac{\alpha}{e^{\alpha-1}}(\frac{1}{(\alpha-1)(1-h)^{\alpha-1}} - \frac{1}{\alpha-1})$$

where the inequality comes from Lemma 3. The competitive ratio is $E_A/E_{opt} \ge (1 + \frac{b}{\ln\epsilon})^\alpha \frac{1}{\alpha}e^{\alpha-1}$, which is again at least $\frac{1}{\alpha}e^{\alpha-1}$ when ϵ tends to 0.

(ii) If $1 - \frac{1}{e} < h \le 1 - \epsilon$, the optimal algorithm runs at speed $a(t)$ for $t \in [0, 1 - e(1-h)]$ and run at speed $\frac{1}{e(1-h)}$ for $t \in [1 - e(1-h), 1]$. It is easy to check that this schedule completes all work. Then,

$$E_{opt} = \int_0^{1-e(1-h)} (\frac{1}{1-x})^\alpha dx + (\frac{1}{e(1-h)})^\alpha \cdot e(1-h) \tag{6}$$

$$= \frac{\alpha}{e^{\alpha-1}}\frac{1}{(\alpha-1)(1-h)^{\alpha-1}} - \frac{1}{\alpha-1} \tag{7}$$

The competitive ratio is $\frac{E_A}{E_{opt}} \ge (1 + \frac{b}{\ln\epsilon})^\alpha \frac{1}{\alpha}e^{\alpha-1}$, which is at least $\frac{1}{\alpha}e^{\alpha-1}$ when ϵ tends to 0. □

We now turn attention to the case that the energy usage of A is less than $E(t)$ for all $0 < t \leq 1-\epsilon$. We first show in Lemma 6 that A cannot complete too much work by time $1-\epsilon$.

Lemma 6. *Assume at any time $0 < t \leq 1-\epsilon$, the energy usage of A up to time t is less than $E(t)$. Then, the amount of work done by A up to time $1-\epsilon$ is less than $\int_0^{1-\epsilon}(1+\frac{b}{\ln\epsilon})\frac{1}{1-x}dx$.*

Proof. Let $s_1(y)$ be the speed of the algorithm A and consider the algorithm B that works at speed $s_2(t) = (1+\frac{b}{\ln\epsilon})\frac{1}{1-t}$. The energy consumed by B by time t is exactly $\int_0^t s_2(y)^\alpha dy = E(t)$. The result now follows by applying Lemma 4 with $x = 1-\epsilon$ and observing that $s_2(t)$ is monotonically increasing. $\square$

We are now ready to show, in Lemma 7, that if the adversary doesn't end the arrival of work until time $1-\epsilon$ then the online algorithm uses at least $\frac{1}{\alpha}e^{\alpha-1}$ times as much energy as optimal.

Lemma 7. *If at any time $0 < t \leq 1-\epsilon$, the total energy usage of A is less than $E(t)$, then A is at least $\frac{1}{\alpha}e^{\alpha-1}$-competitive.*

Proof. Note that the adversary ends the arrival of work at time $1-\epsilon$ and the total amount of work arrived is $\int_0^{1-\epsilon}\frac{1}{1-x}dx = -\ln\epsilon$. By Lemma 6, the maximum amount of work completed by A up to time $1-\epsilon$ is

$$\int_0^{1-\epsilon}(1+\frac{b}{\ln\epsilon})\frac{1}{1-x}dx = (1+\frac{b}{\ln\epsilon})[-\ln(1-x)]_0^{1-\epsilon} = -\ln\epsilon - b$$

Hence, A has at least b units of work remaining at time $1-\epsilon$. To finish it, the total energy usage of A is at least $\frac{b^\alpha}{\epsilon^{\alpha-1}}$, which equals $\frac{1}{(\alpha-1)\epsilon^{\alpha-1}}$ by setting $b = \frac{1}{(\alpha-1)^{1/\alpha}}$. Using equation 7 we find that the energy usage of the optimal algorithm is at most $\frac{\alpha}{e^{\alpha-1}}\frac{1}{(\alpha-1)\epsilon^{\alpha-1}}$. Thus, the competitive ratio is at least $\frac{1}{\alpha}e^{\alpha-1}$. $\square$

Theorem 4. *Any algorithm is at least $\frac{1}{\alpha}e^{\alpha-1}$-competitive.*

Lower bound for qOA**.** Finally, we give a job instance to show that qOA is at least $\frac{1}{4\alpha}4^\alpha(1-\frac{2}{\alpha})^{\alpha/2}$ competitive. The analysis is left to the full paper.

Job Instance: Let $1 > \epsilon > 0$ be some small fixed constant. Consider the input job sequence where work is arriving during $[0, 1-\epsilon]$ and the rate of arrival at time t is

$$a(t) = \frac{1}{(1-t)^x} \ ,$$

where $x > \frac{1}{\alpha}$ is a constant (which will be set to $\frac{2}{\alpha}$ later). All work has deadline 1. Finally, a job is released at time $1-\epsilon$ with work ϵ^{1-x} and deadline 1. $\blacksquare$

Theorem 5. *Let α be a known constant. For any choice of q,* qOA *is at least $\frac{1}{4\alpha}4^\alpha(1-\frac{2}{\alpha})^{\alpha/2}$ competitive.*

References

1. S. Albers, F. Müller, and S. Schmelzer. Speed scaling on parallel processors. In *Proc. ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 289–298, 2007.
2. N. Bansal, D. Bunde, H. L. Chan, and K. Pruhs. Average rate speed scaling. In *Latin American Theoretical Informatics Symposium*, 2008.
3. N. Bansal, H. Chan, and K. Pruhs. Speed scaling with a solar cell, submitted. In *International Conference on Algorithmic Aspects in Information and Management*, 2008.
4. N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 520–529, 2004.
5. N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *JACM*, 54(1), 2007.
6. N. Bansal and K. Pruhs. Speed scaling to manage temperature. In *STACS*, pages 460–471, 2005.
7. N. Bansal, K. Pruhs, and C. 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.
8. D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.
9. H. L. Chan, W.-T. Chan, T.-W. Lam, L.-K. Lee, K.-S. Mak, and P. W. H. Wong. Energy efficient online deadline scheduling. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 795–804, 2007.
10. H.-L. Chan, J. Edmonds, T.-W. Lam, L.-K. Lee, A. Marchetti-Spaccamela, and K. Pruhs. Nonclairvoyant speed scaling for flow and energy. In *STACS*, 2009.
11. W.-C. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. In *Proc. ACM-IEEE Design Automation Conf.*, pages 125–130, 2003.
12. M. Li, B. J. Liu, and F. F. Yao. Min-energy voltage allocation for tree-structured tasks. *Journal of Combinatorial Optimization*, 11(3):305–319, 2006.
13. M. Li, A. C. Yao, and F. F. Yao. Discrete and continuous min-energy schedules for variable voltage processors. In *Proc. of the National Academy of Sciences USA*, volume 103, pages 3983–3987, 2006.
14. M. Li and F. F. Yao. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM J. on Computing*, 35:658–671, 2005.
15. K. Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.
16. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 374–382, 1995.
17. H. Yun and J. Kim. On energy-optimal voltage scheduling for fixed priority hard real-time systems. *ACM Trans. on Embedded Computing Systems*, 2(3):393–430, 2003.