RC24461 (W0801-032) January 8, 2008
Computer Science

IBM Research Report

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

Nikhil Bansal
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Ho-Leung Chan, Kirk Pruhs
Computer Science Department
University of Pittsburgh

Dmitriy Rogozhnikov-Katz
Operations Research Center
MIT

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

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

Nikhil Bansal*, Ho-Leung Chan**, Kirk Pruhs***, and Dmitriy Rogozhnikov-Katz†

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.

For CMOS based processors, and many other types of devices, $\alpha = 3$, that is, they satisfy the cube-root rule. Thus the most interesting case is when $\alpha = 3$. When $\alpha = 3$, the algorithm with the best known competitive ratio is Optimal Available (OA), which is 27-competitive. We introduce a new algorithm qOA, and show that qOA is 6.7-competitive when $\alpha = 3$. So when the cube-root rule holds, our results reduce the range for the optimal competitive ratio from $[1.8, 27]$ to $[2.4, 6.7]$. We also analyze qOA for general α and give almost matching upper and lower bounds.

1 Introduction

1.1 The Setting

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 the literature assumes a speed to power function $P(s) = s^\alpha$, where $\alpha > 1$ 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

* IBM T. J. Watson Research Center, nikhil@us.ibm.com
** Computer Science Department, University of Pittsburgh, hlchan@cs.pitt.edu
*** Computer Science Department, University of Pittsburgh, kirk@cs.pitt.edu. Supported in part by NSF grants CNS-0325353, CCF-0514058 and IIS-0534531.
† Operations Research Center, MIT, dimdim@mit.edu

to conserve energy, and optimize some Quality of Service (QoS) measure of the resulting schedule.

The first theoretical worst-case study of speed scaling algorithms was in the seminal paper [18] by Yao, Demers, and Shenker. In the problem introduced in [18] 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 [18, 5, 11, 8, 7, 15, 19, 13, 4, 12]. 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.

1.2 The Story to Date

[18] show that the optimal schedule can be efficiently computed offline by a greedy algorithm YDS. [18] 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, if $w(x)$ denotes the amount of work that has arrive thus far, and 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, called BKP, is proposed in [7]. BKP runs at speed $e\ v(t)$ 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. As α approaches infinity, the limiting problem is that of minimizing the maximum, over all times t, of the speed at time t. BKP is e-competitive with respect to min-max speed, and no better competitive ratio is achievable [7]. So intuitively BKP should be close to being optimally competitive with respect to energy for large α.

We summarize previous results, which can also be viewed in Table 1. The competitive ratio of OA is exactly α^α [7], where the upper bound is proved using an amortized local competitiveness argument. The competitive ratio of AVR is at most $2^{\alpha-1}\alpha^\alpha$. This was first shown in [18], and a simpler amortized local competitiveness analysis can be found in [5]. 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 [5]. Thus the competitive ratio of AVR is strictly inferior to the competitive ratio of OA. The competitive ratio of BKP is at most $2(\alpha/(\alpha - 1))^\alpha e^\alpha$ [7], which is about $2e^{\alpha+1}$ for large α . The competitive ratio of BKP is provably better than that of OA only for $\alpha \geq 5$. On the other hand, the lower bounds for any general algorithm are rather weak. Somewhat surprisingly, the best known lower bound instance is essentially the worst-possible instance

consisting of two jobs. [6] shows that one can obtain a lower bound of $\left(\frac{4}{3}\right)^\alpha/2$ on the competitive ratio for a two job instance. If one tries to find the worst 3, 4, ... job instances, the calculations get messy quickly.

1.3 Our Contributions

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 [10]. The power is also roughly proportional to the cube of the speed in many common devices/machines, such as vehicles/automobiles [1], and some types of motors [2]. 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 $\alpha = 3$.

So the goal in this paper is to focus in on the cases that $\alpha = 3$, and to a lesser extent on the case that $\alpha = 2$, and to close the range on the optimal competitive ratio in these cases.

Previous Results

Algorithm	General α		$\alpha = 2$		$\alpha = 3$	
	Upper	Lower	Upper	Lower	Upper	Lower
General		$\left(\frac{4}{3}\right)^\alpha/2$				1.8
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	$4^\alpha/(2\sqrt{e\alpha})$	$\frac{1}{2q\alpha}4^\alpha(1-\frac{2}{\alpha})^{\alpha/2}$	2.3		6.7	

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

In section 4, we give the first non-trivial lower bound on the competitive ratio. We show that every deterministic algorithm must have a competitive ratio of at least $e^{\alpha-1}/\alpha$. For $\alpha = 3$, this raises the best known lower bound a modest amount, from 1.8 to 2.4. The instance is identical to one used in [7] 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.

We introduce some informal concepts to facilitate our discussion of improving the upper bound on the competitive ratio. Let A be a generic speed scaling

algorithm. We say that A is *conservative* if A is presuming that no more jobs will arrive in the future, and is running at a minimal speed. So OA may be viewed as the most natural conservative algorithm. Alternatively A is *aggressive* if A is speculating that more jobs will arrive in the future and is running more quickly than the current state might suggest. BKP is an aggressive algorithm. We say that A is a *local-state* algorithm if the speed at any time t depends only upon information about jobs that have been released, but not finished by A, before time t. OA and AVR are local-state algorithms. BKP is not a local-state algorithm since it speed may depend upon work already completed by BKP in the past. Local state algorithms are more amenable to locally amortized competitive analysis using a potential function. Generally speaking, the tightest upper bound analyses of competitive ratios in the literature use amortized local competitiveness arguments [7, 5, 9]. Analyses that resort to more global arguments, such as the analysis of BKP in [7] and the analysis in [3], generally obtain competitive ratios that are presumably much larger than the actual competitive ratios.

AVR is provably inferior to OA, which is the optimal conservative algorithm. In order to be more competitive than OA, one needs an aggressive algorithm. In order to be able to reasonably tightly bound the competitive ratio using current techniques, one needs a local-state algorithm (this requirement rules out the lone remaining previous candidate algorithm BKP). In section 5 we introduce an aggressive local-state algorithm, that we call qOA. qOA runs at speed equal to some constant q times the speed that OA would run in the current state. 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 $4^\alpha/(2\sqrt{e\alpha})$. The potential function that we used is quite different than the one used in the analysis of OA in [7] (and the potential function used to analyze AVR in [5]). The potential function we use is more similar to the one used in [9] to analyze a speed scaling algorithm for the objective of flow time plus energy. One fundamental step in the analysis of the potential function used in [9] applied Young's inequality. However, Young's inequality gives a bound that is too weak to be useful when analyzing qOA. Thus the analysis for qOA is necessarily different than the analysis of the similar potential function in [9]. A key feature of our analysis is the use of convexity, which allows us to reduce the analysis of the general case down to just two extreme cases.

The upper bound of $4^\alpha/(2\sqrt{e\alpha})$ on the competitive ratio of qOA 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 2.3-competitive when $\alpha = 2$, and at worst 6.7-competitive when $\alpha = 3$.

Given the general lower bound of $e^{\alpha-1}/\alpha$, a natural question is whether there is some choice of q for which the competitive ratio of qOA varies with exponential with e as the base of the exponent. Somewhat surprisingly, we show that this is not the case. In particular, in section 6 we show that the competitive ratio of qOA is at least $\frac{1}{2q\alpha}4^\alpha(1 - \frac{2}{\alpha})^{\alpha/2}$-competitive[1] which is about $4^\alpha/(2eq\alpha)$ for

[1] Note that $q \leq 4$, otherwise ratio of qOA is trivially $\Omega(4^\alpha)$.

large α. Thus our upper bound on the competitive ratio of qOA has the right asymptotic growth rate as a function of α.

Our results presented in this paper are summarized in the last two rows of table 1. In particular we have reduced the range for the optimal competitive ratio in the case that the cube-root rule holds from $[1.8, 27]$ to $[2.4, 6.7]$ and in the case that $\alpha = 2$ from $[1, 4]$ to $[1.3, 2.3]$.

2 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 [13]. Li, Yao and Yao [14] 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 [19] 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 [12] gave a polynomial time algorithm for the case of a processor with discrete speeds. Li and Yao [15] gave an algorithm with running time $O(d \cdot n \log n)$ where d is the number of speeds.

Albers, Müller, and Schmelzer [4] consider the problem of finding energy-efficient deadline-feasible schedules on multiprocessors. [4] showed that the offline problem is NP-hard, and gave $O(1)$-approximation algorithms. [4] also gave online algorithms that are $O(1)$-competitive when job deadlines occur in the same order as their release times. Chan et al. [11] 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 [16] investigated speed scaling for deadline feasibility in devices with a regenerative energy source such as a solar cell.

Results on finding low-temperature deadline-feasible schedules can be found in [7]. In general, temperature is more closely related to instantaneous power than cumulative power, which is energy.

3 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 a job 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.

If A is a scheduling algorithm, then $A(I)$ denotes the schedule output by A on input I. A schedule is c-competitive for a particular objective function if the value of that objective function on the schedule is at most c times the value of the objective function on an optimal schedule. An online scheduling algorithm A is c-competitive, or has competitive ratio c, if $A(I)$ is c-competitive for all instances.

4 Lower Bound for General Algorithms

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 give an adversarial strategy for constructing a job instance such that any algorithm uses at least $\frac{1}{\alpha}e^{\alpha-1}$ times more energy than the optimal algorithm.

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 $0 < h \leq 1 - \epsilon$ 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 1 we show that if the adversary ends the arrival of work at some time $0 < h \leq 1 - \epsilon$ 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 5, 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 from Lemma 1 and Lemma 5.

Lemma 1. *If there is a time $0 < h \leq 1-\epsilon$ 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,

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

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

(i) If $1-\frac{1}{e} < h \leq 1-\epsilon$, 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,

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

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

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

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

where the inequality comes from Lemma 2. The competitive ratio is $\frac{E_A}{E_{opt}} \geq (1+\frac{b}{\ln\epsilon})^\alpha \frac{1}{\alpha}e^{\alpha-1}$, which is again at least $\frac{1}{\alpha}e^{\alpha-1}$ when ϵ tends to 0.

Lemma 2 is a technical result used in the proof of Lemma 1.

Lemma 2. *For any $0 < h \leq 1-\frac{1}{e}$, $(-\ln(1-h))^\alpha \leq \frac{\alpha}{e^{\alpha-1}}(\frac{1}{(\alpha-1)(1-h)^{\alpha-1}} - \frac{1}{\alpha-1})$*
.

Proof. Let $f(h) = (-\ln(1-h))^\alpha - \frac{\alpha}{e^{\alpha-1}}(\frac{1}{(\alpha-1)(1-h)^{\alpha-1}} - \frac{1}{\alpha-1})$. Differentiating $f(h)$, we have

$$\begin{aligned} f'(h) &= \alpha(-\ln(1-h))^{\alpha-1}\frac{1}{1-h} - \frac{\alpha}{e^{\alpha-1}}\frac{1}{\alpha-1}(\alpha-1)\frac{1}{(1-h)^\alpha} \\ &= \frac{\alpha}{1-h}\left((-\ln(1-h))^{\alpha-1} - (\frac{1}{e(1-h)})^{\alpha-1}\right) \end{aligned}$$

We can check easily by differentiation that $-\ln(1-h) = \ln\frac{1}{1-h} \leq \frac{1}{e(1-h)}$ for all $h > 0$, and the equality holds only at $h = 1-\frac{1}{e}$. Therefore, $f'(h)$ is non-positive, and $f(h) \leq f(0) = 0$. The lemma then follows.

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 3 that A cannot complete too much work by time $1 - \epsilon$.

Lemma 3. *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.

Lemma 4 is a technical result used in the proof of Lemma 3.

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 $0 < y \leq x$, then $\int_0^x (s_1(t) - s_2(t)) dt < 0$.*

Proof. Define $F(y) = \int_0^y (s_1(t)^\alpha - s_2(t)^\alpha) dt$ and $G(y) = \alpha \int_0^y s_2(t)^{\alpha-1} (s_1(t) - s_2(t)) dt$. By Bernoulli Inequality, $(1+z)^\alpha \geq 1 + \alpha z$ for all $\alpha > 1$ and $z \in [-1, \infty)$. Hence

$$\begin{aligned} F(y) &= \int_0^y \left(s_2(t)^\alpha ((1 + \frac{s_1(t) - s_2(t)}{s_2(t)})^\alpha - 1) \right) dt \\ &\geq \alpha \int_0^y s_2(t)^{\alpha-1} (s_1(t) - s_1(t)) dt = G(y) \end{aligned}$$

Since $F(y) < 0$ for all $y \in (0, x]$, it follows that $G(y) < 0$ for all $y \in (0, x]$. As s_2 is monotonically increasing and non-negative, it follows that $G(y) s_2'(y) / s_2(y)^\alpha < 0$ for all $y \in (0, x]$ and hence that

$$\int_0^x G(y) \frac{s_2'(y)}{s_2(y)^\alpha} dy < 0.$$

Applying integration by parts and noting that $G'(y) = \alpha s_2(y)^{\alpha-1} (s_1(y) - s_2(y))$, we obtain that

$$\begin{aligned} 0 &> \int_0^x G(y) \frac{s_2'(y)}{s_2(y)^\alpha} dy \\ &= \left[G(y) \frac{s_2(y)^{1-\alpha}}{1-\alpha} \right]_0^x - \int_0^x \alpha \frac{s_1(y) - s_2(y)}{1-\alpha} dy \\ &= -G(x) \frac{s_2(x)^{1-\alpha}}{\alpha - 1} + \frac{\alpha}{\alpha - 1} \int_0^x (s_1(y) - s_2(y)) dy \end{aligned}$$

The last equality follows from $G(0) = 0$. Since $G(x) < 0$, we obtain the desired result.

We are now ready to show, in Lemma 5, 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 5. *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 3, 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 $b = \frac{1}{(\alpha-1)^{1/\alpha}}$. Simple calculation shows 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 $\frac{1}{\alpha}e^{\alpha-1}$.

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

5 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, as we shall later in section 6 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. Such an analysis needs 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 [17].

Before defining the potential function Φ that we use, we need to introduce a fair amount of notation. We always denote the current time as t_0. For any

$t_0 \le t' \le t''$, 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 $d(t', t'') = \max\{0, w_a(t', t'') - w_o(t', t'')\}$. Note that $d(t', t'')$ 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.

The following observations will be useful in our analysis:

Observation 2 **(i)** $s_o \ge \max_t w_o(t_0, t)/(t - t_0)$. **(ii)** $s_a \ge q g_0$ *and* $s_a \le q g_0 + q s_o$.

Proof. (i) is true because OPT needs to complete at least $w_o(t_0, t)$ units of work by the corresponding time t. For the first inequality of (ii), $s_a = q \cdot \max_t w_a(t_0, t)/(t - t_0) \ge q w_a(t_0, t_1)/(t_1 - t_0) = q g_0$. For the second inequality,

$$\begin{aligned} s_a &= q \cdot \max_t w_a(t_0, t)/(t - t_0) \le q \cdot \max_t (w_o(t_0, t) + d(t_0, t))/(t - t_0) \\ &\le q \cdot \max_t w_o(t_0, t)/(t - t_0) + q \cdot \max_t d(t_0, t)/(t - t_0) \le q s_o + q g_0 \end{aligned}$$

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

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

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, and job completions by either qOA or optimal, do not change the value Φ since they do not change the value of $d(t', t'')$ for any t', t''. Structural changes in the t_i and g_i do not change the value of Φ. For example, suppose 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 these two merge together. Upon this merge, the potential does not change as can be easily verified. Similarly, if offline works too fast, the interval $[t_k, t_{k+1}]$ might split into two critical intervals, but again this change is continuous.

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

Lemma 6. *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 $w_a(t_0,t_1) \geq w_o(t_0,t_1)$. Therefore, $s_a^\alpha + d\Phi/dt - c \cdot s_o^\alpha \leq 0$ because $s_a \leq q s_o$ and $c = q^\alpha(1+\alpha^{-1/(\alpha-1)})^{\alpha-1} > q^\alpha$.

Hence we assume $w_a(t_0,t_1) \geq w_o(t_0,t_1)$ 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 $t_k < t' \leq t_{k+1}$.

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

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

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. We also notice that the rate of change of $d(t_{k+1}, t_k)$ may be s_o or 0, depending on whether $w_a(t_{k+1}, t_k)$ is greater than $w_o(k_{k+1}, t_k)$. Therefore,

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

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

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

Now consider the case that $k = 0$. Note that for $i \geq 1$, 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

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

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

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

Taking derivative with respect to s_o, we get that this is maximized at s_o satisfying $cs_o^{\alpha-1} = \beta g_0^{\alpha-1}$, 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

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

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

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

We set $q = 2 - (1/\alpha)$ and $\beta = c = q^\alpha\eta^{\alpha-1}$ where $\eta = 1 + \alpha^{-1/(\alpha-1)}$. With this choices of q, β and c, $\alpha q = 2\alpha - 1$ and equation (3) is equivalent to $q^\alpha - \beta \leq 0$, 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 3. qOA *is* $(2-(1/\alpha))^\alpha(1+\alpha^{-1/(\alpha-1)})^{\alpha-1}$*-competitive for general* $\alpha > 1$.

For $\alpha = 3$ this bound on the competitive ratio of qOA evaluates to $(5/3)^3(1+1/\sqrt{3})^2 \approx 11.52$. This is substantially better than the previous best known bound of 27 on the competitive ratio (which was for the algorithm OA). Note that for large values of α, this bound on the competitive ratio of qOA is approximately $4^\alpha/(2\sqrt{e\alpha})$.

We now show that even better bounds on the competitive ratio of qOA can be achieved in the cases that $\alpha = 2$ and $\alpha = 3$ by a careful choice of q and β.

Theorem 4. *If* $q = 1.54$*, then* qOA *is 6.73-competitive for* $\alpha = 3$.

Proof. We follow the same proof structure as that for Lemma 6 to obtain the inequalities (3) and (4). By putting $\alpha = 3$, it follows that we need to satisfy:

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

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 $-3.08x^3 + 10.96x^2 - 1.65x - 16.73$. If we differentiate with respect to x, and solve $-9.2x^2 + 21.9x - 1.7 = 0$, we obtain that $x = 0.1$ or $x = 2.3$. Therefore, the left hand side of the second inequality is maximized at $x = 0.1$ or $x = 2.3$, which are negative in both cases. Hence (3) and (4) are both satisfied and the lemma follows.

Theorem 5. *If $q = 1.5$, then,* qOA *is 2.252-competitive for $\alpha = 2$.*

Proof. Set $\beta = 2.2$ and follow the same lines as the proof of Theorem 4.

6 Lower Bound for qOA

In this section, we show that qOA is at least $\frac{1}{2q\alpha}4^\alpha(1-\frac{2}{\alpha})^{\alpha/2}$ competitive. We assume α is known and fixed, and qOA can optimize the choice of q based on α. When α is large, this lower bound is about $\frac{1}{2q\alpha e}4^\alpha$. Hence, our analysis of the competitive ratio of qOA in the previous section at least has the right growth rate as a function of α. We consider the following job instance:

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. Furthermore, there is a job released at time $1-\epsilon$ with work ϵ^{1-x} and deadline 1. □

We first bound the energy used in the optimal schedule.

Lemma 7. *For the above instance, the optimal algorithm uses total energy at most $\frac{x\alpha}{x\alpha-1}\frac{1}{\epsilon^{x\alpha-1}}$.*

Proof. It suffices to give a schedule that completes all work with energy at most $\frac{x\alpha}{x\alpha-1}\frac{1}{\epsilon^{x\alpha-1}}$. Consider the schedule that runs at speed $a(t)$ during $[0, 1-\epsilon]$ and then runs at speed $\frac{1}{\epsilon^x}$ during $[1-\epsilon, 1]$. We can easily see that this schedule completes all work by deadline 1 and the energy usage of the schedule is

$$\begin{aligned}
\int_0^{1-\epsilon}(a(t))^\alpha dt + \left(\frac{1}{\epsilon^x}\right)^\alpha \cdot \epsilon &= \int_0^{1-\epsilon}\frac{1}{(1-t)^{x\alpha}}dt + \frac{1}{\epsilon^{x\alpha-1}} \\
&= \left[\frac{1}{x\alpha-1}\frac{1}{(1-t)^{x\alpha-1}}\right]_0^{1-\epsilon} + \frac{1}{\epsilon^{x\alpha-1}} \\
&= \frac{1}{x\alpha-1}\frac{1}{\epsilon^{x\alpha-1}} - \frac{1}{x\alpha-1} + \frac{1}{\epsilon^{x\alpha-1}} \\
&\leq \frac{x\alpha}{x\alpha-1}\frac{1}{\epsilon^{x\alpha-1}}
\end{aligned}$$

We now analyze the energy usage of qOA. We will lower bound the energy used by qOA by the energy used by qOA during the time interval $[1-\epsilon, 1]$. In Lemma 8 we bound the amount of work w remaining for qOA at time $1-\epsilon$. During the time interval $[1-\epsilon, 1]$, qOA behaves exactly how it would have if the instance consisted only of one job with work w, release time $1-\epsilon$, and deadline 1. Lemma 9 bounds the energy used by qOA on such a one job instance. Lemma 10 then gives our lower bound on the energy used by qOA.

Lemma 8. *For the above instance,* qOA *has* $\frac{x+q}{x+q-1}\frac{1}{\epsilon^{x-1}} - \frac{1}{x+q-1}\epsilon^q$ *units of work remaining at time* $1-\epsilon$ *after the last job is released.*

Proof. Let $s(t)$ be the speed of qOA at time $t \in [0, 1-\epsilon]$. We note that $s(t)$ satisfies

$$s(t) = q \cdot \frac{\int_0^t a(y)dy - \int_0^t s(y)dy}{1-t} \tag{6}$$

Solving the equation, we obtain

$$s(t) = \frac{q}{x+q-1}\frac{1}{(1-t)^x} - \frac{q}{x+q-1}(1-t)^{q-1}$$

We can verify it easily by showing that it satisfies (6). We also remark that this is the unique solution for $s(t)$. Then, the amount of work remaining at time $1-\epsilon$ after the last job is released is

$$\begin{aligned}
&\int_0^{1-\epsilon} a(t)dt - \int_0^{1-\epsilon} s(t)dy + \epsilon^{1-x} \\
&= \int_0^{1-\epsilon} (1 - \frac{q}{x+q-1})\frac{1}{(1-y)^x} + \frac{q}{x+q-1}(1-y)^{q-1}dy + \epsilon^{1-x} \\
&= \left[\frac{1}{x+q-1}(1-y)^{-x+1} - \frac{1}{x+q-1}(1-y)^q\right]_0^{1-\epsilon} + \epsilon^{1-x} \\
&= \frac{x+q}{x+q-1}\frac{1}{\epsilon^{x-1}} - \frac{1}{x+q-1}\epsilon^q
\end{aligned}$$

Lemma 9. *Consider a job instance consisting of a single job with work* w*, and deadline minus release time of* ℓ*. The energy usage of* qOA *is* $\frac{q^\alpha}{q\alpha+\alpha-1}\left(\frac{w}{\ell}\right)^\alpha \ell$*.*

Proof. Without loss of generality, we may assume that the release time is 0, and the deadline is ℓ. Let $g(t)$ be the speed of qOA at time t for this single job sequence. Then,

$$g(t) = q \cdot \frac{w - \int_0^t g(y)dy}{\ell - t}$$

Solving the equation, we can obtain that

$$g(t) = q\left(\frac{\ell}{\ell - t}\right)^{1-q}\left(\frac{w}{\ell}\right)$$

Then, the energy usage of qOA to complete the job is

$$\begin{aligned}\int_0^\ell (g(t))^\alpha dt &= q^\alpha \ell^{(1-q)\alpha} \left(\frac{w}{\ell}\right)^\alpha \int_0^\ell \frac{1}{(\ell-t)^{(1-q)\alpha}} dt \\ &= q^\alpha \ell^{(1-q)\alpha} \left(\frac{w}{\ell}\right)^\alpha \left[\frac{-1}{q\alpha-\alpha+1}(\ell-t)^{q\alpha-\alpha+1}\right]_0^\ell \\ &= \frac{q^\alpha}{q\alpha-\alpha+1}\left(\frac{w}{\ell}\right)^\alpha \ell\end{aligned}$$

We remark that for this single job instance, the energy usage of qOA is $\frac{q^\alpha}{q\alpha-\alpha+1}$ times that of the minimum possible one.

Lemma 8 and Lemma 9 we get a lower bound on the energy used by qOA during the time interval $[1-\epsilon, 1]$.

Lemma 10. *The energy usage of* qOA *during* $[1-\epsilon, 1]$ *is*

$$\frac{q^\alpha}{q\alpha-\alpha+1}\left(\frac{x+q}{x+q-1}\frac{1}{\epsilon^x} - \frac{1}{x+q-1}\epsilon^{q-1}\right)^\alpha \epsilon.$$

Finally, we have the main theorem of this section.

Theorem 6. *Let* α *be a known constant. For any choice of* q, qOA *is at least* $\frac{1}{2q\alpha}4^\alpha(1-\frac{2}{\alpha})^{\alpha/2}$ *competitive.*

Proof. By Lemma 10, when ϵ tends to zero, the energy usage of qOA is at least $\frac{q^\alpha}{q\alpha-\alpha+1}\left(\frac{x+q}{x+q-1}\right)^\alpha \frac{1}{\epsilon^{x\alpha-1}}$. Together with Lemma 7, the competitive ratio of qOA is at least

$$\frac{q^\alpha}{q\alpha-\alpha+1}\left(\frac{x+q}{x+q-1}\right)^\alpha \cdot \frac{x\alpha-1}{x\alpha} \geq \frac{q^\alpha}{q\alpha}\left(\frac{x+q}{x+q-1}\right)^\alpha \cdot \frac{x\alpha-1}{x\alpha}$$

We put $x = \frac{2}{\alpha}$ and the right hand side becomes $\frac{1}{2q\alpha}\left(\frac{q(q+\frac{2}{\alpha})}{q+\frac{2}{\alpha}-1}\right)^\alpha$. To minimize it, we note that

$$\frac{d}{dq}\frac{q(q+\frac{2}{\alpha})}{q+\frac{2}{\alpha}-1} = \frac{(q+\frac{2}{\alpha}-1)(2q+\frac{2}{\alpha}) - (q^2+\frac{2q}{\alpha})}{(q+\frac{2}{\alpha}-1)^2}$$

Setting it to zero, we obtain that $q^2 + (\frac{4}{\alpha}-2)q + (\frac{4}{\alpha^2}-\frac{2}{\alpha}) = 0$ and $q = 1-\frac{2}{\alpha}+\sqrt{1-\frac{2}{\alpha}}$ (the other solution for q is negative and hence does not apply). Putting it back, we obtain that the competitive ratio is at least

$$\begin{aligned}&\frac{1}{2q\alpha}\left(\frac{(1-\frac{2}{\alpha}+\sqrt{1-\frac{2}{\alpha}})(1+\sqrt{1-\frac{2}{\alpha}})}{\sqrt{1-\frac{2}{\alpha}}}\right)^\alpha \\ &= \frac{1}{2q\alpha}\left(\left(1+\sqrt{1-\frac{2}{\alpha}}\right)^2\right)^\alpha \geq \frac{1}{2q\alpha}\left(4\sqrt{(1-\frac{2}{\alpha})}\right)^\alpha = \frac{1}{2q\alpha}4^\alpha(1-\frac{2}{\alpha})^{\alpha/2}\end{aligned}$$

The inequality in the second step follows by using $(a+b)^2 \geq 4ab$. This completes the proof.

References

1. http://auto.howstuffworks.com/question477.htm.
2. Grundfos Motor Book.
3. S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. In *Lecture Notes in Computer Science (STACS)*, volume 3884, pages 621 – 633, 2006.
4. 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.
5. N. Bansal, D. Bunde, H.-L. Chan, and K. Pruhs. Average rate speed scaling. In *LATIN 2008, to appear*.
6. 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.
7. N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *JACM*, 54(1), 2007.
8. N. Bansal and K. Pruhs. Speed scaling to manage temperature. In *STACS*, pages 460–471, 2005.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. M. Li and F. F. Yao. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM J. on Computing*, 35:658–671, 2005.
16. H. C. N. Bansal, K. Pruhs. Speed scaling with a solar cell, submitted.
17. K. Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.
18. 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.
19. 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.