Average Rate Speed Scaling*

Nikhil Bansal[1], David P. Bunde[2], Ho-Leung Chan[3], and Kirk Pruhs[3]

[1] IBM T. J. Watson Research Center,
nikhil@us.ibm.com
[2] Computer Science Department, Knox College
dbunde@knox.edu
[3] Computer Science Department, University of Pittsburgh
{hlchan,kirk}@cs.pitt.edu

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. Yao, Demers, and Shenker [8] considered the problem where the QoS constraint is deadline feasibility and the objective is to minimize the energy used. They proposed an online speed scaling algorithm Average Rate (AVR) that runs each job at a constant speed between its release and its deadline. They showed that the competitive ratio of AVR is at most $(2\alpha)^\alpha/2$ if a processor running at speed s uses power s^α. We show the competitive ratio of AVR is at least $((2-\delta)\alpha)^\alpha/2$, where δ is a function of α that approaches zero as α approaches infinity. This shows that the competitive analysis of AVR by Yao, Demers, and Shenker is essentially tight, at least for large α. We also give an alternative proof that the competitive ratio of AVR is at most $(2\alpha)^\alpha/2$ using a potential function argument. We believe that this analysis is significantly simpler and more elementary than the original analysis of AVR in [8].

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. 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. 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.

* D.P. Bunde was supported in part by Howard Hughes Medical Institute grant 52005130. Ho-Leung Chan and Kirk Pruhs were supported in part by NSF grants CNS-0325353, CCF-0514058 and IIS-0534531.

The first theoretical worst-case study of speed scaling algorithms was in the seminal paper [8] by Yao, Demers, and Shenker. Their QoS objective was deadline feasibility and the objective was to minimize the energy used. More precisely, 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. If job i runs at constant speed s, then it completes in w_i/s units of time. In this setting, an optimal job selection policy is Earliest Deadline First (EDF). They assumed a speed to power function $P(s) = s^\alpha$, where $\alpha > 1$ is some constant. If the cube-root rule holds, then $\alpha = 3$. Yao, Demers, and Shenker [8] showed that the optimal energy feasible schedule is found by a simple greedy algorithm that we call YDS.

Yao, Demers, and Shenker [8] also proposed an online speed scaling algorithm, Average Rate (AVR). Conceptually, AVR runs each job i at speed $w_i/(d_i - r_i)$ throughout interval $[r_i, d_i]$, independent of other jobs. This spreads the work of each job as evenly over time as possible. By the convexity of the speed to power function, this even spreading is energy optimal if the instance consists of only one job. The speed of the processor at any time t is then just the sum of the speeds of the jobs active at that time, that is $\sum_{i:t\in[r_i,d_i]} \frac{w_i}{d_i - r_i}$. AVR is an appealing speed scaling algorithm because in some sense it is perfectly fair to all jobs, and each job runs as if it were the only job in the instance.

Yao, Demers, and Shenker [8] showed that the competitive ratio, with respect to energy, of AVR is at least α^α. They also showed that the competitive ratio of AVR, with respect to energy, is at most $(2\alpha)^\alpha/2$. We now outline this upper bound competitive analysis of AVR. A job is defined to be of *type A* if the optimal schedule is always ahead of AVR on this job. A job is defined to be of *type B* if AVR is always ahead of the optimal schedule on this job. A schedule is *bitonic* if every job is of type A or type B. [8] observes that there is a worst-case instance that is bitonic, and that the competitive ratio of AVR is at most $2^{\alpha-1}$ times the competitive ratio of AVR on instances of jobs of just one type (A or B). [8] then considers instances consisting only of type-A jobs. [8] then introduces an auxiliary objective function that is related to, but is not exactly, the energy used. In a somewhat involved reduction, [8] shows that with respect to this auxiliary objective, there is a worst-case instance where the optimal schedule is non-preemptive, each job starts when it is released, and the spans of the jobs are nested (where the *span* of job i is the interval $[r_i, d_i]$). When $\alpha = 2$, [8] then shows that for such instances, optimizing the auxiliary objective function can be represented in terms of the eigenvalues of a particular tree-induced matrix, and shows how to bound the largest eigenvalue for such tree-induced matrices. [8] states that this argument can be readily generalized to an arbitrary α, and using Hölder's inequality, give a bound on the ℓ_p norm of a certain tree-induced matrix that would replace the eigenvalue argument used in the $\alpha = 2$ case.

So the natural question left open is, "What is the exact competitive ratio of AVR?" Based on simulation results, [8] conjectured that the competitive ratio of AVR is exactly α^α. That is, that the lower bound in [8] is correct, and intuitively, that AVR can not simultaneously be losing badly on both type-A and type-B jobs. In the case that the cube-root rule holds, $\alpha^\alpha = 3^3 = 27$ is the best known

competitive ratio for any online algorithm. If the conjecture from [8] was true, this would be evidence in favor of adopting the AVR speed scaling policy. Not only would AVR have the best known competitive ratio in the case that the cube-root rule holds, but AVR is appealingly fair to all jobs.

Unfortunately, in section 4, we show that the upper bound on the competitive ratio from [8] is essentially tight, at least for larger α. More precisely, we show that AVR has competitive ratio at least $((2-\delta)\alpha)^\alpha/2$, where δ is a function of α that approaches zero as α approaches infinity. In the case obeying the cube-root rule, we get a lower bound of approximately 48 on the competitive ratio of AVR.

In section 5, we give an alternative proof that the competitive ratio of AVR is at most $(2\alpha)^\alpha/2$. Our analysis uses a potential function argument. We believe that this analysis is significantly simpler and more elementary than the original analysis of AVR in [8]. Our competitive analysis of AVR branches off from the analysis in [8] outlined above after the observation that the competitive ratio of AVR is at most $2^{\alpha-1}$ times the competitive ratio of AVR on jobs of just one type. We give a potential function argument that AVR is α^α-competitive on type-A jobs. We include a complete analysis of AVR in this paper, including the elements of the analysis from [8] that we use. In principle, verifying this analysis requires only basic algebra, except that some basic calculus is used to verify the positivity/negativity of certain polynomials over particular ranges.

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.

Yao, Demers, and Shenker [8] also proposed another online speed scaling algorithm, Optimal Available (OA). The algorithm OA runs at the optimal speed (which can be computed using the YDS algorithm) assuming the current state and that no more jobs will be released in the future. [8] showed that the competitive ratio of OA is at least α^α. Using a potential function analysis, Bansal, Kimbrel, and Pruhs [2] showed that OA is actually α^α-competitive.

Bansal, Kimbrel, and Pruhs [2] also introduced an online speed scaling algorithm that we call BKP. Intuitively, BKP tries to mimic the offline YDS schedule in some way. Formally, at time t BKP runs at speed $e\ v(t)$ where $v(t) = \max_{t'>t} \frac{w(t,et-(e-1)t',t')}{e(t'-t)}$ 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. [2] showed that BKP is simultaneously $O(1)$-competitive for total energy, maximum temperature (assuming cooling obeys Fourier's law), maximum power, and maximum speed. Specifically, [2] showed that the competitive ratio of BKP with respect to energy is at most $2(\alpha/(\alpha-1))^\alpha e^\alpha$. With respect to maximum speed, [2] showed that BKP is e-competitive and that this competitive ratio is optimal among randomized algorithms.

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 [4]. Li, Yao and Yao [5] 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 [9] 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 [3] gave a polynomial time algorithm to schedule a processor with discrete speeds. Li and Yao [6] gave an algorithm with running time $O(d \cdot n \log n)$ where d is the number of speeds.

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.

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

For a schedule T, let $s_{T,j}(t)$ denote the speed job j runs at time t in the schedule T, and let $s_T(t) = \sum_j s_{T,j}(t)$ denote the speed of the processor at time t in schedule T. If U is a subcollection of jobs, let $s_{T,U}(t)$ denote the sum of the speeds of the jobs in U at time t in the schedule T. We will also substitute an algorithm for a schedule in this notation. So for example, $s_{AVR}(t)$ is the speed of the algorithm AVR at time t. We use OPT to denote a particular optimal schedule. We say that job i is *active* between its release time and its deadline. We call $w_i/(d_i - r_i)$ the *density* of job i since this is the job's work divided by the length of the interval in which it is active.

Algorithm AVR: At all times t, run the earliest-deadline job at speed $s_{AVR}(t) = \sum_i \frac{w_i}{d_i - r_i}$, where the sum is over jobs i active at time t.

Consider a fixed optimum schedule OPT. A job is said to be of *type A* if

$$\int_{r_j}^{t} s_{OPT,j}(t)dt \geq \int_{r_j}^{t} \frac{w_j}{d_i - r_i}dt \qquad \text{for all } r_j \leq t \leq d_j$$

Intuitively, these are the jobs that OPT runs consistently ahead of their density. Similarly, the jobs of *type B* are those that OPT runs consistently behind their density, meaning they satisfy

$$\int_{r_j}^{t} s_{OPT,j}(t)dt \leq \int_{r_j}^{t} \frac{w_j}{d_i - r_i}dt \qquad \text{for all } r_j \leq t \leq d_j.$$

In general, a job need not be of either type (or it can also be of both types, in which case OPT executes exactly as in AVR). We say an instance is *bitonic* if every job is of type A, type B, or both (in which case it is arbitrarily assigned one of the types). A simple observation (Lemma 5) shows that if AVR is c-competitive for bitonic instances, then it is also c-competitive in general.

4 The Lower Bound

We give an instance on which AVR uses up at least $((2-\delta)\alpha)^\alpha/2$ times the energy used by an energy optimum solution, where δ is a function of α that tends to zero as α increases.

Instance Description: For convenience we will work with a continuous version of the job instance. We say that work arrives at rate $a(t)$ at time t to mean that $a(t)dt$ units of work arrive during the infinitesimally small interval $[t, t+dt]$.

The instance consists of two sets of jobs A and B. The work in A arrives during the time interval $[0, 1-\epsilon]$, at rate

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

and all the work in A has deadline 1. Here $\epsilon > 0$ is an arbitrarily small but fixed constant. The work in B arrives during the interval $[1-1/c, 1-\epsilon/c]$ (where c is a constant that will be set to $\alpha - 1$ later) at rate

$$b(t) = \frac{c}{c^{1/\alpha}(1-t)^{1/\alpha}}$$

and the work in B arriving at time t has deadline $1 + c(1-t)$.

Lemma 1. *On the instance above, the optimal algorithm uses total energy at most* $2\ln(1/\epsilon)$.

Proof: It suffices to give some feasible schedule that uses energy $2\ln(1/\epsilon)$. Consider the schedule that completes all jobs in A by running at speed $a(t)$ during $[0, 1-\epsilon]$. The energy usage is

$$\int_0^{1-\epsilon} (a(t))^\alpha dt = [-\ln(1-t)]_0^{1-\epsilon} = \ln(1/\epsilon)$$

For jobs in B, note that they are released before time 1 and have deadlines in $[1+\epsilon, 2]$. Consider any time $x \in [1+\epsilon, 2]$. The jobs with deadline in $[1+\epsilon, x]$ are released during $[1-\frac{x-1}{c}, 1-\frac{\epsilon}{c}]$. Their total amount of work is

$$\int_{1-(x-1)/c}^{1-\epsilon/c} b(t)dt = \int_{1-(x-1)/c}^{1-\epsilon/c} \frac{c}{c^{1/\alpha}(1-t)^{1/\alpha}}dt$$

Let $y = 1 + c(1-t)$. Then $dy = -c \cdot dt$, and the amount of work equals

$$\int_{1-(x-1)/c}^{1-\epsilon/c} \frac{c}{c^{1/\alpha}(1-t)^{1/\alpha}}dt = \int_{x}^{1+\epsilon} \frac{-1}{(y-1)^{1/\alpha}}dy = \int_{1+\epsilon}^{x} \frac{1}{(y-1)^{1/\alpha}}dy$$

Therefore, consider the schedule that processes jobs in B at speed $\hat{b}(y) = \frac{1}{(y-1)^{1/\alpha}}$ continuously during $[1+\epsilon, 2]$. For any $x \in [1+\epsilon, 2]$, the amount of work done by time x equals the amount work with deadline by x. So the schedule completes each job in B by its deadline. The energy usage to complete all jobs in B is

$$\int_{1+\epsilon}^{2} (\hat{b}(t))^{\alpha} dt = [\ln(y-1)]_{1+\epsilon}^{2} = \ln(1/\epsilon)$$

Since the intervals of execution of work in A and B do not overlap, the total energy used is $2\ln(1/\epsilon)$ and the lemma follows. □

Lemma 2. *On the instance above, AVR uses total energy at least* $\alpha^{\alpha}(1 + \frac{c}{c^{1/\alpha}(c+1)})^{\alpha} \ln(1/\epsilon) + K$*, where* K *is a constant independent of* ϵ*.*

Proof: Consider the work in A. The work released at time t is scheduled by AVR uniformly during the interval $[t, 1]$. Thus, at any time $x \in [0, 1]$, the density due to work in A is

$$\text{den}_a(x) = \int_0^x a(t) \cdot \frac{1}{1-t}dt = \int_0^x \frac{1}{(1-t)^{1/\alpha}} \cdot \frac{1}{1-t}dt = \alpha\left(\frac{1}{(1-x)^{1/\alpha}} - 1\right)$$

Now consider the work in B. Note that for work released at time t, the duration between its release time and deadline is $1 + c(1-t) - t = (c+1)(1-t)$. Thus, at any time $x \in [1-\frac{1}{c}, 1-\frac{\epsilon}{c}]$, the density due to work in B is

$$\begin{aligned}\text{den}_b(x) &= \int_{1-1/c}^{x} \frac{c}{c^{1/\alpha}(1-t)^{1/\alpha}} \cdot \frac{1}{(c+1)(1-t)}dt \\ &= \frac{c}{c^{1/\alpha}(c+1)} \cdot \alpha\left(\frac{1}{(1-x)^{1/\alpha}} - c^{1/\alpha}\right)\end{aligned}$$

During the interval $[1-\frac{1}{c}, 1-\epsilon]$, AVR runs at speed equal to the total density due to work in A and B. Therefore, the energy usage of AVR is at least

$$\begin{aligned}&\int_{1-1/c}^{1-\epsilon} (\text{den}_a(t) + \text{den}_b(t))^{\alpha}\, dt \\ &\quad = \int_{1-1/c}^{1-\epsilon} \left(\alpha\left(1 + \frac{c}{c^{1/\alpha}(c+1)}\right) \cdot \frac{1}{(1-t)^{1/\alpha}} - \alpha\frac{2c+1}{c+1}\right)^{\alpha} dt \qquad (1)\end{aligned}$$

Let $Y = 1 + \frac{c}{c^{1/\alpha}(c+1)}$. Note that for all $t \in [1-\frac{1}{c}, 1-\epsilon]$, we have that $1-t \le 1/c$ and hence

$$\frac{2c+1}{c+1} \cdot \frac{(1-t)^{1/\alpha}}{Y} \le \frac{2c+1}{c+1} \cdot \frac{1}{c^{1/\alpha}} \cdot \frac{c^{1/\alpha}(c+1)}{c^{1/\alpha}(c+1)+c} \le \frac{2c+1}{(c+1)+c} = 1$$

Then, by factoring $\alpha Y \frac{1}{(1-t)^{1/\alpha}}$, the right side of (1) can be written as

$$\begin{aligned}
&\int_{1-1/c}^{1-\epsilon} \alpha^\alpha Y^\alpha \frac{1}{1-t} \left(1 - \frac{2c+1}{c+1} \cdot \frac{(1-t)^{1/\alpha}}{Y}\right)^\alpha dt \\
&\ge \int_{1-1/c}^{1-\epsilon} \frac{\alpha^\alpha Y^\alpha}{1-t} \left(1 - \alpha \frac{2c+1}{c+1} \cdot \frac{(1-t)^{1/\alpha}}{Y}\right) dt \quad \text{as } 1-\alpha x \le (1-x)^\alpha \text{ for } x \le 1 \\
&= \int_{1-1/c}^{1-\epsilon} \alpha^\alpha Y^\alpha \left(\frac{1}{1-t} - Z(1-t)^{(1/\alpha)-1}\right) dt \qquad \text{where } Z = \tfrac{\alpha(2c+1)}{Y(c+1)} \\
&= \alpha^\alpha Y^\alpha \left[-\ln(1-t) + \alpha Z (1-t)^{1/\alpha}\right]_{1-1/c}^{1-\epsilon} \\
&= \alpha^\alpha Y^\alpha \left(-\ln \epsilon + \alpha Z \epsilon^{1/\alpha} + \ln \frac{1}{c} - \alpha Z (\frac{1}{c})^{1/\alpha}\right) \\
&\ge \alpha^\alpha Y^\alpha \ln(1/\epsilon) + \alpha^\alpha Y^\alpha \left(\ln \frac{1}{c} - \alpha Z (\frac{1}{c})^{1/\alpha}\right) \qquad \text{since } \epsilon > 0
\end{aligned}$$

Since α, c, Y and Z are independent of ϵ the lemma follows. □

Theorem 3. *The competitive ratio of AVR is at least $((2-\delta)\alpha)^\alpha/2$, where δ is a function of α that tends to zero as α increases.*

Proof: By Lemma 1 and 2, when ϵ tends to zero, the competitive ratio of AVR is at least $((1 + \frac{c^{1-1/\alpha}}{c+1})\alpha)^\alpha/2$. Putting $c = \alpha - 1$, the competitive ratio is at least $((1 + \frac{(\alpha-1)^{1-1/\alpha}}{\alpha})\alpha)^\alpha/2$, which equals $((2-\delta)\alpha)^\alpha/2$ where $\delta = 1 - \frac{(\alpha-1)^{1-1/\alpha}}{\alpha}$.

Note that for large α (in particular for $\alpha \ge 2$, we have that

$$\begin{aligned}
\delta &= 1 - (\alpha-1)^{-1/\alpha} \frac{\alpha-1}{\alpha} \\
&= 1 - e^{(-1/\alpha)\ln(\alpha-1)}(1 - \frac{1}{\alpha}) \\
&\le 1 - \left(1 - \frac{1}{\alpha}\ln(\alpha-1)\right)(1 - \frac{1}{\alpha}) \qquad \text{using } e^x \ge 1 + x \text{ for } x < 0 \\
&= \frac{\ln(\alpha-1)}{\alpha} + \frac{1}{\alpha} - \frac{\ln(\alpha-1)}{\alpha^2} \qquad (2)
\end{aligned}$$

Hence δ approaches zero as α approaches infinity. □

We remark that our bound $((2-\delta)\alpha)^\alpha/2$ is asymptotically $2^{\alpha-1}\alpha^{\alpha-1/2-o(1)}$ for large α, and hence within $\alpha^{1/2+o(1)}$ of the best known upper bound. To see this, by (2), we obtain that

$$\lim_{\alpha\to\infty} \left(\frac{\alpha}{\ln \alpha}\right)\delta \le \lim_{\alpha\to\infty} \left(\frac{\ln(\alpha-1)}{\ln \alpha} + \frac{1}{\ln \alpha} - \frac{\ln(\alpha-1)}{\alpha \ln \alpha}\right) = 1.$$

Similarly,

$$\delta \geq 1 - \frac{\alpha^{1-1/\alpha}}{\alpha} = 1 - \frac{1}{e^{(\ln \alpha/\alpha)}} \geq 1 - \frac{1}{1 + \frac{1}{\alpha}\ln \alpha} = \frac{\ln \alpha}{\alpha + \ln \alpha},$$

and hence

$$\lim_{\alpha \to \infty} \left(\frac{\alpha}{\ln \alpha}\right) \delta \geq \lim_{\alpha \to \infty} \frac{\alpha}{\alpha + \ln \alpha} = 1.$$

Thus the expression $(2-\delta)^\alpha \alpha^\alpha / 2 = 2^{\alpha-1}\alpha^\alpha (1-\delta/2)^\alpha \approx 2^{\alpha-1}\alpha^\alpha \alpha^{-\delta\alpha/(2\ln\alpha)} = 2^{\alpha-1}\alpha^\alpha \alpha^{-1/2-o(1)}$.

5 An Elementary Proof that AVR is $2^{\alpha-1}\alpha^\alpha$-competitive

This section gives a complete elementary proof that AVR is $2^{\alpha-1}\alpha^\alpha$-competitive. This proof uses some elements of the analysis of AVR in [8] and some variations on elements of the analysis of OA in [2]. We start with the analysis of AVR on instances consisting of only type-A jobs. The analysis for general instances then follows along the same lines as in [8], and is included here for completeness.

Lemma 4. *For instances consisting of only type-A jobs,* AVR *is α^α-competitive with respect to energy.*

Proof: We use an amortized local competitiveness argument (for more information on such arguments in scheduling problems, see [7]). At any time t, either a task arrives or finishes, or else an infinitesimal interval of time dt elapses and AVR consumes $s_{AVR}(t)^\alpha dt$ units of energy. We will define a potential function $\phi(t)$ that satisfies the following properties:

- The potential function $\phi(t)$ has value 0 before any jobs arrive and after the last deadline.
- The potential function $\phi(t)$ does not increase as a result of AVR completing a job, OPT completing a job, or the release of a job.
- At any time t,

$$s_{AVR}(t)^\alpha + \frac{d\phi(t)}{dt} \leq \alpha^\alpha s_{OPT}(t)^\alpha. \tag{3}$$

Integrating equation 3 over time and using the other two stated properties, we can conclude the desired result.

Before we can define the potential function we need to introduce some notation. Let t_0 denote the current time and t_i denote the time of the i^{th} deadline occurring after t_0. Then let I_i denote the interval of time $[t_i, t_{i+1})$. Let $\tau_i = t_{i+1} - t_i$ be the length of interval I_i. Let s_i denote the speed at which AVR will work during interval I_i if no new jobs arrive. This can be computed by summing the densities of active jobs whose deadline is at or after time t_{i+1}. Let $w_{AVR,i} = s_i \tau_i$ denote the amount of work that AVR plans to complete during interval I_i. Let $w_{OPT,i}$ be the portion of the work AVR allocates to interval I_i that OPT has not yet completed. Because all jobs are of type A, all work that

is unfinished by OPT is also unfinished by AVR. Without loss of generality, we assume that when OPT is working on a job j, work is removed from the term $w_{OPT,i}$ that contains work from job j with the smallest index i. That is, OPT removes work from the earlier intervals first.

We define the potential function $\phi(t)$ as follows:

$$\phi(t) = \alpha \sum_{i \geq 0} s_i^{\alpha-1}(w_{AVR,i} - \alpha w_{OPT,i}) \tag{4}$$

This potential function is a slight modification of the potential function used in [2] to analyze the algorithm OA. The difference is that their potential function uses $w_{OPT,i}$ to denote the work of jobs unfinished for OPT with deadline in I_i.

Now we show that ϕ has the claimed properties. This function is clearly 0 when there are no active jobs. The completion of a job by OPT also has no effect since the potential is a continuous function of $w_{OPT,i}$. The situation when AVR completes a job is slightly more complicated. Observe that a job completes under AVR if and only if the size of the interval I_0 shrinks to 0, i.e. when the current time t_0 becomes equal to t_1, which shifts all the indices. At the moment this happens AVR has completed all the work allocated to I_0 and hence $w_{AVR,0} = 0$. Because all jobs are of type A, OPT has also completed the work allocated to I_1 so $w_{\mathrm{OPT},0} = 0$. Thus, the potential is continuous even in this case. (This is the only time we use that all the jobs are of type A.)

Arrival Case: The next case to consider is when a new job j arrives. First observe that adding a zero work job with deadline d_j does not change the value of the potential function ϕ. Thus, we may assume that the new job's deadline is t_k for some k. Let y be the density of the new job. Then the release increases the density of intervals $I_0, I_1, \ldots, I_{k-1}$ by y, increasing the weight of interval I_i by $y\tau_i$ for $0 \leq i \leq k-1$. This changes the potential function by

$$\begin{aligned} \Delta\phi = \alpha \sum_{i=0}^{k-1} \left(\frac{w_{AVR,i} + y\tau_i}{\tau_i} \right)^{\alpha-1} ((w_{AVR,i} + y\tau_i) - \alpha(w_{OPT,i} + y\tau_i)) \\ -\alpha \sum_{i=0}^{k-1} \left(\frac{w_{AVR,i}}{\tau_i} \right)^{\alpha-1} (w_{AVR,i} - \alpha w_{OPT,i}). \end{aligned} \tag{5}$$

This expression can be rearranged into

$$\begin{aligned} \sum_{i=0}^{k-1} \frac{\alpha}{\tau_i^{\alpha-1}} \Big((w_{AVR,i} + y\tau_i)^{\alpha-1}(w_{AVR,i} - \alpha w_{OPT,i} - (\alpha-1)y\tau_i) \\ -w_{AVR,i}^{\alpha-1}(w_{AVR,i} - \alpha w_{OPT,i}) \Big) \end{aligned}$$

By making the substitutions $q = w_{AVR,i}$, $\delta = y\tau_i$ and $r = w_{OPT,i}$ each term of this sum becomes a quantity shown to be at most 0 by Lemma 8.

Working case: We now consider times when no job arrives, and no jobs complete. Each s_i, including s_0, remains fixed during this time. We have to show

$$s_{AVR}(t_0)^\alpha - \alpha^\alpha s_{OPT}(t_0)^\alpha + \frac{d\phi(t)}{dt} \leq 0 \tag{6}$$

or equivalently,

$$s_0^\alpha - \alpha^\alpha s_{OPT}(t_0)^\alpha + \frac{d}{dt}(\alpha \sum_{i \geq 0} s_i^{\alpha-1}(w_{AVR,i} - \alpha w_{OPT,i})) \leq 0 \tag{7}$$

As AVR works, $w_{AVR,0}$ is decreasing at rate s_0, and $w_{AVR,i}$ remains fixed for all $i \geq 1$. Since OPT takes work from a single interval I_i, only one of the $w_{OPT,i}$ changes; let it be $w_{OPT,k}$. Then equation (7) is equivalent to

$$s_0^\alpha - \alpha^\alpha s_{OPT}(t_0)^\alpha + (-\alpha s_0^{\alpha-1} s_0 + \alpha^2 s_k^{\alpha-1} s_{OPT}(t_0)) \leq 0$$

Since a job active during one interval is also active in all earlier intervals, $s_k \leq s_0$ and it suffices to show that

$$(1-\alpha)s_0^\alpha + \alpha^2 s_0^{\alpha-1} s_{OPT}(t_0) - \alpha^\alpha s_{OPT}(t_0)^\alpha \leq 0$$

Substituting $z = s_0/s_{OPT}(t_0)$ gives

$$(1-\alpha)z^\alpha + \alpha^2 z^{\alpha-1} - \alpha^\alpha \leq 0 \tag{8}$$

Let $u(z)$ be the polynomial on the left hand side of inequality 8. Note that $u(0) = -\alpha^\alpha$ and $u(+\infty) = -\infty$. In addition, the derivative of $u(z)$ is 0 at only the point $z = \alpha$. Since $u(\alpha) = 0$, we conclude that $u(z)$ is non-positive for $z \geq 0$, which holds because of the definition of z. This establishes inequality 6. □

Lemma 4 and the argument of Yao, Demers, and Shenker [8] proves the $2^{\alpha-1}\alpha^\alpha$-competitiveness of AVR. We now give their argument for completeness.

Lemma 5. *[8] Among those instances on which* AVR *has it worst-case competitive ratio, there is a bitonic instance.*

Proof: Consider a worst-case instance I that is not bitonic. We explain how to transform I into another worst-case instance that is bitonic. There must be a job i that is of neither type A nor type B. By the definition of the types, there has to be some times s, u, with $s < u$, for which one of AVR or OPT is ahead of the other on job i at time s, but behind at time u. By the intermediate value theorem, there must be a time $t \in (s, u)$ where AVR and OPT have completed an equal amount of work w on job i. We say that the *lead changes* at such a time t. We now create a new instance I' from I by replacing job i with two jobs: one with work w released at time r_i with deadline t, and one with work $w_i - w$ released at time t with deadline d_i. It is easy to see that both AVR and OPT always run at the same speed in I' that they did in I. This transformation however reduces the number of lead changes by one. Since there can only be a bounded number of lead changes between YDS = OPT and AVR, a bounded number of applications of this transformation leads to a bitonic instance. □

Lemma 6. *[8]* AVR *is* $2^{\alpha-1}\alpha^\alpha$*-competitive on bitonic instances.*

Proof Sketch: Given a bitonic instance, let A be the set of type-A jobs and B be the others. Let AVR_A and AVR_B denote the energy attributable to A and B in the AVR schedule, respectively. Define OPT_A and OPT_B similarly with reference to the schedule OPT.

Next observe that the roles of type-A jobs and type-B jobs can be swapped by reversing time and swapping the release time and deadline for each job. Both YDS and AVR give the same schedule to the forward and backwards versions so Lemma 4 implies that AVR is simultaneously α^α-competitive with respect to energy attributable to type-A jobs and energy attributable to type-B jobs.

The proof follows by combining the schedules for the jobs of different types. The optimal cost is clearly at least $\text{OPT}_A + \text{OPT}_B$. To bound the cost of AVR, define $s_{AVR,A}(t)$ and $s_{AVR,B}(t)$ as the speed of AVR on type-A and type-B jobs respectively. Then the cost of AVR is at most

$$\begin{aligned}\int s_{AVR}(t)^\alpha dt &= \int \left(s_{AVR,A}(t) + s_{AVR,B}(t)\right)^\alpha dt \\ &\leq \int 2^{\alpha-1}\left(s_{AVR,A}(t)^\alpha + s_{AVR,B}(t)^\alpha\right) dt \\ &= 2^{\alpha-1}\left(\text{AVR}_A + \text{AVR}_B\right) \\ &\leq 2^{\alpha-1}\alpha^\alpha(\text{OPT}_A + \text{OPT}_B),\end{aligned}$$

which gives the desired ratio. □

Thus we reach our final theorem, which is an immediate consequence of Lemma 4, Lemma 5, and Lemma 6.

Theorem 7. AVR *is* $2^{\alpha-1}\alpha^\alpha$*-competitive.*

The following lemma from [2] was used in the proof of Lemma 4:

Lemma 8. *[2] Let* $q, r, \delta \geq 0$ *and* $\alpha \geq 1$*. Then* $(q+\delta)^{\alpha-1}(q - \alpha r - (\alpha-1)\delta) - q^{\alpha-1}(q - \alpha r) \leq 0$*.*

Proof: The lemma is equivalent to showing that

$$(q - \alpha r)[(q+\delta)^{\alpha-1} - q^{\alpha-1}] - (q+\delta)^{\alpha-1}(\alpha-1)\delta \leq 0$$

Since $[(q+\delta)^{\alpha-1} - q^{\alpha-1}] \geq 0$, it suffices to show that

$$q[(q+\delta)^{\alpha-1} - q^{\alpha-1}] - (q+\delta)^{\alpha-1}(\alpha-1)\delta \leq 0$$

Let $\delta = zq$, which implies $z \geq 0$. The left hand side of the above becomes

$$q^\alpha[(1+z)^{\alpha-1} - 1] - q^\alpha[(1+z)^{\alpha-1}(\alpha-1)z]$$

Factoring out q^α and differentiating the rest with respect to z gives

$$\begin{aligned}&((\alpha-1)(1+z)^{\alpha-2}[1-(\alpha-1)z] + (1+z)^{\alpha-1}(-\alpha+1)) \\ &= ((\alpha-1)(1+z)^{\alpha-2}[1-(\alpha-1)z - (1+z)] \\ &= -\alpha(\alpha-1)z(1+z)^{\alpha-2}\end{aligned}$$

This is non-positive since $\alpha > 1$ and $z \geq 0$. Thus, the expression is maximized at $z = 0$, where it has value 0. This implies the result. □

6 Conclusion

Even though AVR is not optimally competitive, one could imagine situations where a system designer might still adopt AVR because AVR is in some sense fair to each job. This is analogous to the reason that Processor Sharing (Round Robin) is adopted in some systems even though Processor Sharing is known not to have the best competitive ratio for the standard QoS measures.

Acknowledgments: We thank Don Coppersmith for helpful discussions.

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, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *JACM*, 54(1), 2007.
3. 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.
4. 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.
5. 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.
6. M. Li and F. F. Yao. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM J. on Computing*, 35:658–671, 2005.
7. K. Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.
8. 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.
9. 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.