Race to Idle: New Algorithms for Speed Scaling with a Sleep State

SUSANNE ALBERS and ANTONIOS ANTONIADIS, Humboldt-Universität zu Berlin

We study an energy conservation problem where a variable-speed processor is equipped with a sleep state. Executing jobs at high speeds and then setting the processor asleep is an approach that can lead to further energy savings compared to standard dynamic speed scaling. We consider classical deadline-based scheduling, that is, each job is specified by a release time, a deadline and a processing volume. For general convex power functions, Irani et al. [2007] devised an offline 2-approximation algorithm. Roughly speaking, the algorithm schedules jobs at a critical speed s_{crit} that yields the smallest energy consumption while jobs are processed. For power functions $P(s) = s^\alpha + \gamma$, where s is the processor speed, Han et al. [2010] gave an $(\alpha^\alpha + 2)$-competitive online algorithm.

We investigate the offline setting of speed scaling with a sleep state. First, we prove NP-hardness of the optimization problem. Additionally, we develop lower bounds, for general convex power functions: No algorithm that constructs s_{crit}-schedules, which execute jobs at speeds of at least s_{crit}, can achieve an approximation factor smaller than 2. Furthermore, no algorithm that minimizes the energy expended for processing jobs can attain an approximation ratio smaller than 2.

We then present an algorithmic framework for designing good approximation algorithms. For general convex power functions, we derive an approximation factor of 4/3. For power functions $P(s) = \beta s^\alpha + \gamma$, we obtain an approximation of $137/117 < 1.171$. We finally show that our framework yields the best approximation guarantees for the class of s_{crit}-schedules. For general convex power functions, we give another 2-approximation algorithm. For functions $P(s) = \beta s^\alpha + \gamma$, we present tight upper and lower bounds on the best possible approximation factor. The ratio is exactly $eW_{-1}(-e^{-1-1/e})/(eW_{-1}(-e^{-1-1/e})+1) < 1.211$, where W_{-1} is the lower branch of the Lambert W function.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Sequencing and Scheduling

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Variable-speed processor, power-down mechanism, NP-hard, approximation algorithm

ACM Reference Format:
Susanne Albers and Antonios Antoniadis. 2014. Race to Idle: New algorithms for speed scaling with a sleep state. ACM Trans. Algor. 10, 2, Article 9 (February 2014), 31 pages.
DOI: http://dx.doi.org/10.1145/2556953

1. INTRODUCTION

Modern microprocessors have various capabilities for energy savings. *Dynamic speed scaling* refers to a processor's ability to dynamically set the speed/frequency depending on the current workload. The lower the speed, the lower the dynamic energy

This work is supported by the German Research Foundation, Project Al 464/7-1.
A preliminary version of this article appeared in the *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2012.
Author's addresses: Department of Computer Science, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany.

DOI: http://dx.doi.org/10.1145/2556953

consumption is. However, even at low speed levels a processor consumes a significant amount of static energy, caused, for example, by leakage current. Additionally, processors are typically equipped with stand-by or *sleep states*. In a deep sleep state, a processor uses negligible or no energy. The combination of speed scaling and low-power states suggests the technique *race-to-idle*: Execute tasks at high speed levels, then transition the processor to a sleep state. This can reduce the overall energy consumption. The time periods in which a processor resides in dormant states may be on the order of milliseconds. The race-to-idle concept has been studied in a variety of settings and usually leads to energy-efficient solutions (see, e.g., Lesswatts.org, Bailis et al. [2011], Gandhi et al. [2009], Garrett [2007], and Kluge et al. [2010]).

We adopt a model introduced by Irani et al. [2007] to combine speed scaling and power-down mechanisms. The problem is called *speed scaling with sleep state*. Consider a variable-speed processor that, at any time, resides in an *active state* or a *sleep state*. In the active state the processor can execute jobs, where the energy consumption is specified by a general convex, nondecreasing power function P. If the processor runs at speed s, with $s \geq 0$, then the required power is $P(s)$. We assume $P(0) > 0$, that is, even at speed 0, when no job is processed, a strictly positive power is required. In the active state, energy consumption is power integrated over time. In the sleep state, the processor consumes no energy but cannot execute jobs. A *wake-up* operation, transitioning the processor from the sleep state to the active state, requires a fixed amount of $C > 0$ energy units. A power-down operation, transitioning from the active to the sleep state, does not incur any energy.

We consider classical deadline-based scheduling. We are given n jobs $J_1, \ldots, J_n$. Each job J_i is specified by a release time r_i, a deadline d_i and a processing volume v_i, $1 \leq i \leq n$. Job J_i can be feasibly scheduled in the interval $[r_i, d_i)$. The processing volume is the amount of work that must be completed on the job. If J_i is processed at constant speed s, then it takes v_i/s time units to finish the job. We may assume that each job is processed at a fixed speed. By convexity of the power function P, it is not beneficial to process a job at varying speed. Preemption of jobs is allowed, that is, at any time the processing of a job may be suspended and resumed later. The goal is to construct a feasible schedule minimizing energy consumption.

Given a schedule $\mathcal{S}$, let $E(\mathcal{S})$ denote the energy incurred. This energy consists of two components, the *processing energy* and the *idle energy*. The processing energy $E_p(\mathcal{S})$ is incurred while the processor executes jobs. It holds that $E_p(\mathcal{S}) = \sum_{i=1}^{n} v_i P(s_i)/s_i$, where s_i is the speed at which J_i is processed. The idle energy $E_i(\mathcal{S})$ is expended while the processor resides in the active state but does not process jobs and whenever a wake-up operation is performed. We assume that initially, prior to the execution of the first job, the processor is in the sleep state. Suppose that $\mathcal{S}$ contains T time units in which the processor is active but not executing jobs. Let k be the number of wake-up operations. Then, $E_i(\mathcal{S}) = TP(0) + kC$.

Irani et al. [2007] observed that in speed scaling with sleep state there exists a *critical speed*s_{crit}, which is the most efficient speed to process jobs. Again, if a job J_i is executed at speed s, then the consumed energy is $v_i P(s)/s$. Speed s_{crit} is the smallest value minimizing $P(s)/s$, and will be important in various algorithms.

Previous work: Speed scaling and power-down mechanisms have been studied extensively over the past years and we review the most important results relevant to our work. We concentrate on deadline-based scheduling on a single processor. There exists a considerable body of literature addressing dynamic speed scaling if the processor is *not* equipped with a sleep state. In a seminal paper, Yao et al. [1995] showed that the offline problem is polynomially solvable. They gave an efficient algorithm, called *YDS* according to the initials of the authors, for constructing minimum energy schedules. Refinements of the algorithm were given in Li et al. [2006] and Li and Yao [2005].

YDS was originally presented for power functions $P(s) = s^\alpha$, where $\alpha > 1$, but can be extended to arbitrary convex functions P, see Irani et al. [2007]. For power functions $P(s) = s^\alpha$, various online algorithms were presented in Bansal et al. [2007, 2011, 2012] and Yao et al. [1995].

Baptiste [2006] studied a problem setting where a *fixed-speed* processor has a sleep state. All jobs must be processed at this fixed speed level, and whenever the processor is in the active state 1 energy unit is consumed per time unit. Using a dynamic programming approach, Baptiste showed that the offline problem is polynomially solvable if all jobs have unit processing time. In a subsequent paper, Baptiste et al. [2012] proved that the approach can be extended to arbitrary job processing times. We will refer to the corresponding polynomial time algorithm as *BCD*.

Irani et al. [2007] initiated the study of speed scaling with sleep state. They consider arbitrary convex power functions. For the offline problem, they devised a polynomial time 2-approximation algorithm. The algorithm first executes *YDS* and identifies job sets that must be scheduled at speeds higher than s_{crit} according to this policy. All remaining jobs are scheduled at speed s_{crit}. The complexity of the offline problem was unresolved. Irani and Pruhs [2005] stated that determining the complexity of speed scaling with sleep state is an intellectually intriguing problem. For the online problem, Irani et al. [2007] presented a strategy that transforms a competitive algorithm for speed scaling without sleep state into a competitive algorithm for speed scaling with sleep state. For functions $P(s) = s^\alpha + \gamma$, where $\alpha > 1$ and $\gamma > 0$, Han et al. [2010] showed an $(\alpha^\alpha + 2)$-competitive algorithm.

We remark that speed scaling has also been investigated in a model where there is an upper bound on the processor speed (see, e.g., Bansal et al. [2008], Chan et al. [2009], and Han et al. [2010]). Furthermore, rather than deadline-based scheduling, a problem scenario with the objective function of minimizing energy plus the total job flow time has been studied. In this setting, again, general power functions have been considered (cf., Andrew et al. [2009], Bansal et al. [2013], and Chan et al. [2013]).

Our contribution: In this article, we investigate the offline setting of speed scaling with sleep state. We consider general convex power functions, which are motivated by current processor architectures and applications (see also Bansal et al. [2013]). Moreover, we consider the family of functions $P(s) = \beta s^\alpha + \gamma$, where $\alpha > 1$ and $\beta, \gamma > 0$. Speed scaling without sleep state, considering the deadline-based scheduling model, has mostly addressed power functions $P(s) = s^\alpha$. The family $P(s) = \beta s^\alpha + \gamma$ is the natural generalization.

First, in Section 2, we develop a complexity result as well as lower bounds. We prove that speed scaling with sleep state is NP-hard and thereby settle the complexity of the offline problem. This hardness result holds even for very simple problem instances consisting of so-called tree-structured jobs. Hence, interestingly, while the setting with a fixed-speed processor, studied by Baptiste et al. [2012], admits polynomial time algorithms, the optimization problem turns NP-hard for a variable-speed processor. As for lower bounds, we refer to a schedule $\mathcal{S}$ as an s_{crit}-schedule if every job is processed at a speed of at least s_{crit}. We prove that for general convex power functions, no algorithm constructing s_{crit}-schedules can achieve an approximation factor smaller than 2. This statement holds true even for piecewise linear power functions. The intuition of this result is that in executing jobs at speeds of at least s_{crit}, a processor might be idle for extended time periods and hence might incur a considerable amount of idle energy. The lower bound implies that the offline algorithm by Irani et al. [2007] attains the best possible approximation ratio among s_{crit}-based algorithms. Furthermore, to obtain smaller approximation factors, one has to use speeds smaller than s_{crit}. Our lower bound construction can be used to show a second result: For general convex power functions, no algorithm minimizing the processing energy of schedules can achieve

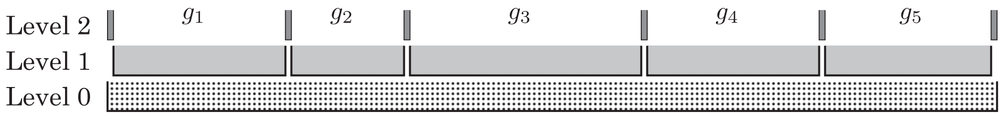

Fig. 1. The execution intervals of the jobs of an $\mathcal{I}_S$ instance, with $n = 5$.

an approximation factor smaller than 2. Both lower bound statements hold for any algorithm, whose running time might even be exponential.

In Section 3, we present a general, generic polynomial time algorithm for speed scaling with sleep state. All the three algorithms we devise in this article are instances of the same algorithmic framework. Our general algorithm combines *YDS* and *BCD*. The main ingredient is a new, specific speed s_0 that determines when to switch from *YDS* to *BCD*. Job sets that must be processed at speeds higher than s_0 are scheduled using *YDS*. All other jobs are processed at speed s_0. While the approach is very natural and simple, it allows us to derive significantly improved approximation factors. For general convex power functions, we present a 4/3-approximation algorithm by choosing s_0 such that $P(s_0)/s_0 = \frac{4}{3}P(s_{crit})/s_{crit}$. The main technical contribution is to properly analyze the algorithmic scheme. The difficult part is to prove that in using speed s_0, but no lower speed levels, we do not generate too much extra idle energy, compared to that of an optimal schedule (cf. Lemmas 3.4 and 3.7 for the 4/3-approximation). In Section 4, we study power functions $P(s) = \beta s^\alpha + \gamma$ and develop an approximation factor of $137/117 < 1.171$ by setting $s_0 = \frac{117}{137}s_{crit}$.

In Section 5, we reconsider s_{crit}-schedules and demonstrate that our algorithmic framework yields the best possible approximation guarantees. For general convex power functions, we give another 2-approximation algorithm, matching our lower bound and the upper bound by Irani et al. [2007]. More importantly, we prove tight upper and lower bounds on the best possible approximation ratio achievable for power functions $P(s) = \beta s^\alpha + \gamma$. The ratio is exactly equal to $eW_{-1}(-e^{-1-1/e})/(eW_{-1}(-e^{-1-1/e}) + 1)$, where W_{-1} is the lower branch of the Lambert W function. The ratio is upper bounded by 1.211.

In summary, in addition to providing a hardness result and small constant-factor approximation guarantees, this article settles the performance of the class of s_{crit}-schedules.

2. COMPLEXITY AND LOWER BOUNDS

In this section, we first prove NP-hardness of speed scaling with sleep state. Then, we present two lower bounds on the performance of s_{crit}-schedules.

A problem instance of speed scaling with sleep state is *tree-structured* if, for any two jobs J_i and J_j and associated execution intervals $I_i = [r_i, d_i)$ and $I_j = [r_j, d_j)$, it holds that $I_i \subseteq I_j$, $I_j \subseteq I_i$ or $I_i \cap I_j = \emptyset$.

THEOREM 2.1. *Speed scaling with sleep state is NP-hard, even on tree-structured problem instances.*

PROOF. Obviously, the decision variant of speed scaling with sleep state is in the class NP. We proceed to describe a reduction from the NP-complete *Partition* problem. In the *Partition* problem, we are given a finite set A of n positive integers $a_1, a_2, \ldots a_n$, and the problem is to decide whether there exists a subset $A' \subset A$ such that $\sum_{a_i \in A'} a_i = \sum_{a_i \in A \setminus A'} a_i$. Let a_{max} be the maximal element of A, that is, $a_{max} = \max_{i \in \{1,\ldots n\}} a_i$. We assume $a_{max} \geq 2$, since otherwise the *Partition* problem is trivial to decide.

Let $\mathcal{I}_P$ be any instance of *Partition* with associated set A. We construct a corresponding instance $\mathcal{I}_S$ of speed scaling with sleep state. Figure 1 depicts a small example of

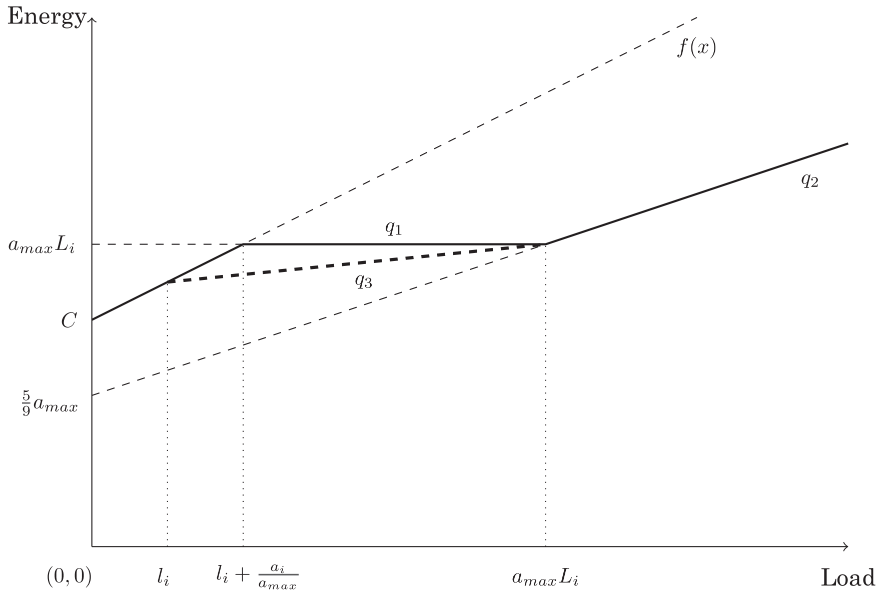

Fig. 2. Energy consumption in gap g_i as a function of the load executed in g_i.

the following construction. For $1 \le i \le n$, set $L_i = 2 - \frac{a_{max}-1}{a_{max}^2}a_i$. The job set $\mathcal{J}$ of $\mathcal{I}_S$ can be partitioned into three levels. We first create $n+1$ jobs of Level 2, comprising set $\mathcal{J}_2 \subset \mathcal{J}$. The i-th job of $\mathcal{J}_2$, with $1 \le i \le n+1$, has a release time of $(i-1)\epsilon + \sum_{j=1}^{i-1} L_j$ and a deadline of $i\epsilon + \sum_{j=1}^{i-1} L_j$, where ϵ is an arbitrary positive constant. The processing volume of each job of $\mathcal{J}_2$ is equal to ϵs_{crit}. For Level 1, we construct n jobs forming the set $\mathcal{J}_1 \subset \mathcal{J}$. The i-th job of Level 1, with $1 \le i \le n$, has a release time of $i\epsilon + \sum_{j=1}^{i-1} L_j$, a deadline of $i\epsilon + \sum_{j=1}^{i} L_j$, and a processing volume of $l_i = L_i a_{max} - a_i$. From now on, we will also use the term *gap* to refer to the intervals where jobs of $\mathcal{J}_1$ can be executed. More specifically, gap g_i is the interval defined by the release time and the deadline of the i-th job of $\mathcal{J}_1$. Finally, there is only one job J_0 of Level 0. It has a release time of 0, a deadline of $(n+1)\epsilon + \sum_{i=1}^{n} L_i$ and a processing volume of $B = \sum_{i=1}^{n} a_i/2$.

Note that $\mathcal{I}_S$ is tree-structured. We set the cost of a wake-up operation equal to $C = a_{max}$. The power function is defined as follows:

$$P(s) = \begin{cases} a_{max}, & 0 \le s \le a_{max}, \\ \frac{4}{9}s + \frac{5}{9}a_{max}, & a_{max} < s \le 10a_{max}, \\ 2s - 15a_{max}, & 10a_{max} < s. \end{cases}$$

It is easy to verify that P is convex and continuous, that $s_{crit} = 10a_{max}$, and that $P(s_{crit})/s_{crit} = 1/2$. For the sake of simplicity, we assume that the processor is in the active state just before the first job gets executed. This is no loss of generality because we can just add the cost of one extra wake-up operation to all the energy consumptions.

Here, in the main body of the article, we present the intuition and main idea of our reduction. Formally, Theorem 2.1 follows from Lemma 2.2, whose proof is given in the appendix. For every gap g_i, $1 \le i \le n$, we consider functions of the energy consumed in g_i depending on the load x executed in the gap (see Figure 2). Function $f(x) = C + (P(s_{crit})/s_{crit})x$ represents the optimal energy consumption in g_i assuming that the processor transitions to the sleep state in the gap. This consumption does not depend on the gap length, and thus the function is the same for all the gaps. Next consider the energy consumption $h_i(x)$ in g_i assuming that the processor remains in the active state throughout the gap. This consumption depends on the required speed s, that is, it is $P(s)L_i$. By the definition of $P(s)$, $h_i(x)$ is given by an arrangement of three lines. More specifically, $h_i(x) = a_{max}L_i$ for $x \in [0, a_{max}L_i]$ (cf. q_1 in Figure 2), $h_i(x) = ((4/9) \cdot (x/L_i) + (5/9)a_{max}) \cdot L_i$ for $x \in (a_{max}L_i, 10a_{max}L_i]$ (shown as q_2), and $h_i(x) = (2(x/L_i) - 15a_{max}) \cdot L_i$ for x in $(10a_{max}L_i, +\infty)$ (not depicted). Function $h_i(x)$ depends on the gap length L_i. Hence, in general, the functions $h_i(x)$, with $1 \leq i \leq n$, are different for the various gaps. For any gap g_i, the optimal energy consumption, with respect to the load executed in it, is given by the lower envelope of f and h_i, represented by the solid line segements in Figure 2. Let $\mathrm{LE}_i(x) = \min\{f(x), h_i(x)\}$ denote this lower envelope function.

Assume now that each job in $\mathcal{J}_2$ is being executed throughout the interval defined by its release time and deadline, and assume that b_i units of J_0's load are executed in gap g_i. In the formal proof, we show that w.l.o.g. J_0 is not executed between gaps. In g_i an energy of at least $\mathrm{LE}_i(l_i + b_i)$ is consumed. We can rewrite this as $\mathrm{LE}_i(l_i) + E_i^b(b_i)$, by simply setting $E_i^b(b_i) = LE_i(l_i + b_i) - LE_i(l_i)$. With this rewriting, we charge an energy of $\mathrm{LE}_i(l_i)$ to the load l_i and an energy of $E_i^b(b_i)$ to the load b_i. Observe that $\mathrm{LE}_i(l_i)$ is the least possible energy expended for gap g_i and that it is attained for $b_i = 0$ when $E_i^b(b_i) = 0$. Since the $\mathrm{LE}_i(l_i)$ energy units charged to the l_i's depend only on the l_i's themselves, the goal of any algorithm is to minimize $\sum_{i=1}^{n} E_i^b(b_i)$ subject to the constraint $\sum_{i=1}^{n} b_i = B$. In other words, the goal is to distribute the B load units to the gaps g_i, $1 \leq i \leq n$, in a way minimizing the energy charged to them.

The average energy consumption per load unit for any b_i corresponds to the slope of the line passing through $(l_i, \mathrm{LE}_i(l_i))$ and $(l_i + b_i, \mathrm{LE}_i(l_i + b_i))$. The key idea of the transformation is that this slope gets minimized when $l_i + b_i = a_{max}L_i$ or, equivalently, when $b_i = a_i$. This minimum possible attainable slope is $1/(2a_{max})$, which is independent of the respective gap g_i. The thick dashed line denoted by q_3 in Figure 2 is exactly this line passing through $(l_i, \mathrm{LE}_i(l_i))$ and $(l_i + b_i, \mathrm{LE}_i(l_i + b_i))$, when $b_i = a_i$.

It follows that the total energy charged to the B load units of J_0 is minimized when each b_i is either 0 or a_i. Calculations show that, in this case, the average energy consumption per load unit is minimized to $1/(2a_{max})$ and hence the total energy charged to the load of J_0, is $B/(2a_{max})$. If there exists at least one gap g_i with $0 < b_i < a_i$ or $b_i > a_i$, then by our construction the slope of the line passing through $(l_i, \mathrm{LE}_i(l_i))$ and $(l_i + b_i, \mathrm{LE}_i(l_i + b_i))$ is greater than $1/(2a_{max})$, which implies that the average energy charged to the load of J_0 is strictly greater than $1/(2a_{max})$, and in turn the total energy consumption is strictly greater than $B/(2a_{max})$.

Formally, our reduction satisfies the following lemma, establishing Theorem 2.1.

LEMMA 2.2. *There exists a feasible schedule for $\mathcal{I}_S$ that consumes energy of at most $5(n+1)\epsilon a_{max} + nC + \frac{1}{2}\sum_{i=1}^{n} l_i + \frac{B}{2a_{max}}$ if and only if A of $\mathcal{I}_P$ admits a partition.*

The proof is given in the appendix.

We next present two lower bounds that hold true for all algorithms, independently of their running times. We exploit properties of schedules but do not take into account their construction time. Again, formally a schedule $\mathcal{S}$ for a job set $\mathcal{J}$ is an s_{crit}*-schedule* if every job is processed at a speed of at least s_{crit}.

THEOREM 2.3. *Let A be an algorithm that computes s_{crit}-schedules for any job instance. Then A does not achieve an approximation factor smaller than 2, for general convex power functions.*

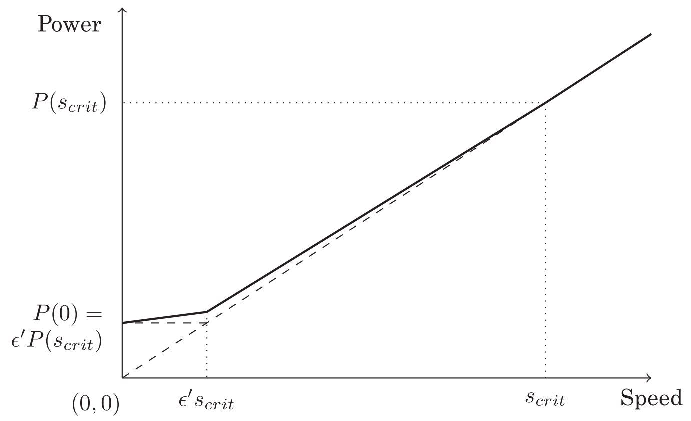

Fig. 3. The power function $P(s)$.

Proof. Let ϵ, where $0 < \epsilon < 1$, be an arbitrary constant. We show that A cannot achieve an approximation factor smaller than $2 - \epsilon$. Set $\epsilon' = \epsilon/7$. Fix an arbitrary critical speed $s_{crit} > 0$ and associated power $P(s_{crit}) > 0$. Let $P(0) = \epsilon' P(s_{crit})$. We define a power function $P(s)$ for which s_{crit} is indeed the critical speed. Function $P(s)$ is piecewise linear, see also Figure 3. In the interval $[0, \epsilon' s_{crit}]$ it is given by the line passing through the points $(0, P(0))$ and $(\epsilon' s_{crit}, (1+\epsilon')P(0))$. In the interval $(\epsilon' s_{crit}, s_{crit})$, it is defined by the line through $(\epsilon' s_{crit}, (1+\epsilon')P(0))$ and $(s_{crit}, P(s_{crit}))$. This line has a slope of $(P(s_{crit}) - (1+\epsilon')P(0))/((1-\epsilon')s_{crit})$. For $s \geq s_{crit}$, $P(s)$ is given by the line $P(s_{crit})s/s_{crit}$. In summary,

$$P(s) = \begin{cases} P(0)(\frac{s}{s_{crit}} + 1), & s \leq \epsilon' s_{crit}, \\ \frac{P(s_{crit}) - (1+\epsilon')P(0)}{1-\epsilon'}(\frac{s}{s_{crit}} - 1) & \\ \quad + P(s_{crit}), & \epsilon' s_{crit} < s < s_{crit}, \\ P(s_{crit})\frac{s}{s_{crit}}, & s_{crit} \leq s. \end{cases}$$

This power function is increasing and convex because the three slopes form a strictly increasing sequence, by our choice of ϵ'. Furthermore, s_{crit} is the smallest value that minimizes $P(s)/s$.

We specify a job sequence. We first define three jobs J_1, J_2 and J_3 with the following characteristics. Let $L > 0$ be an arbitrary constant. Job J_1 has a processing volume of $v_1 = \delta L s_{crit}$, where $\delta = (\epsilon')^2/2$, and can be executed in the interval $I_1 = [0, \delta L)$, that is, $r_1 = 0$ and $d_1 = \delta L$. The second job J_2 has a processing volume of $v_2 = \epsilon' L s_{crit}$ and can be processed in $I_2 = [\delta L, (1+\delta)L)$ so that $r_2 = \delta L$ and $d_2 = (1+\delta)L$. The third job J_3 is similar to the first one with $v_3 = \delta L s_{crit}$. The job can be executed in $I_3 = [(1+\delta)L, (1+2\delta)L)$, that is, $r_3 = (1+\delta)L$ and $d_3 = (1+2\delta)L$. The three intervals I_1, I_2, and I_3 are disjoint, and each of the three jobs can be feasibly scheduled using a speed of s_{crit}. Let $C = LP(0)$ be the energy of a wake-up operation.

We analyze the energy consumption of A and an optimal solution, assuming for the moment that the processor is in the active state at time 0. First consider the energy consumption of A. Suppose that A processes some job at a speed higher than s_{crit}. By the definition of s_{crit}, we can reduce any speed $s \geq s_{crit}$ to s_{crit} without increasing the processing energy needed for the job. The speed reduction only reduces the time while the processor does not execute jobs and thus does not increase the idle energy of the schedule. In other words, any schdule that processes some job at a speed higher than

s_{crit} can be transformed into one that uses a speed of s_{crit} only; the transformation does not increase the schedule's idle energy.

Hence, we may analyze A, assuming that all the three jobs are processed at speed s_{crit}. Jobs J_1 and J_3 each consume an energy of $\delta LP(s_{crit})$. In I_2, job J_2 is processed for $v_2/s_{crit} = \epsilon' L$ time units. During the remaining time, $L - \epsilon' L = (1-\epsilon')L$ time units the processor is idle. Since $C > (1-\epsilon')LP(0)$, it is not worthwhile to power down and the processor should remain in the active state. Hence, A's energy consumption is at least $2\delta LP(s_{crit}) + \epsilon' LP(s_{crit}) + (1-\epsilon')LP(0) > L(\epsilon' P(s_{crit}) + (1-\epsilon')P(0)) = (2-\epsilon')LP(0)$. The last equation holds because $\epsilon' = P(0)/P(s_{crit})$.

In an optimal solution, jobs J_1 and J_3 must also be executed at speed s_{crit}. However, J_2 can be processed using speed $v_2/L = \epsilon' s_{crit}$ in I_2 so that the energy consumption for the job is $LP(\epsilon' s_{crit}) = (1+\epsilon')LP(0)$. Hence, the optimum power consumption is upper bounded by $2\delta LP(s_{crit}) + (1+\epsilon')LP(0) = (\epsilon')^2 LP(s_{crit}) + (1+\epsilon')LP(0) = (1+2\epsilon')LP(0)$. The last equation holds because $\epsilon' = P(0)/P(s_{crit})$.

Now assume that the processor is in the sleep state initially and a wake-up operation must be performed at time 0. In order to deal with this extra cost of C, we repeat the previous job sequence $k = \lceil 1/\epsilon' \rceil$ times. In the i-th repetition, $1 \le i \le k$, there exist three jobs—J_{i1}, J_{i2}, and J_{i3}—with processing volumes $v_{ij} = v_j$, $1 \le j \le 3$. The i-th repetition starts at time $t_i = (i-1)(1+2\delta)L$. For this job sequence, the ratio of the energy consumed by A to that of an optimal solution is greater than $\frac{k(2-\epsilon')LP(0)}{C+k(1+2\epsilon')LP(0)} \ge \frac{2-\epsilon'}{1+3\epsilon'} > 2-\epsilon$. The first inequality holds because $C/k \le \epsilon' LP(0)$, and the second one follows because $\epsilon' = \epsilon/7$. □

For the problem instance defined in the aforementioned proof, s_{crit}-schedules are exactly the ones that minimize the processing energy. This follows by the definition of s_{crit}, and by the fact that for the given power function, s_{crit} is the only speed that minimizes $P(s)/s$. We obtain:

COROLLARY 2.4. *Let A be an algorithm that, for any job instance, computes a schedule minimizing the processing energy. Then, A does not achieve an approximation factor smaller than 2, for general convex power functions.*

3. A 4/3-APPROXIMATION ALGORITHM

We develop a polynomial time 4/3-approximation algorithm for general convex power functions. The algorithm is an instance of a more general algorithmic framework.

3.1. Description of the Algorithm

Our general algorithm combines *YDS* and *BCD* while making crucial use of a new, specific speed level s_0 that determines when to switch from *YDS* to *BCD*. For varying s_0, $0 \le s_0 \le s_{crit}$, we obtain a family of algorithms $ALG(s_0)$. The best choice of s_0 depends on the power function. In order to achieve a 4/3-approximation, we choose s_0 such that $P(s_0)/s_0 = \frac{4}{3}P(s_{crit})/s_{crit}$.

We first argue that our speed level s_0, satisfying $P(s_0)/s_0 = \frac{4}{3}P(s_{crit})/s_{crit}$, is well defined. Speed s_{crit} is the smallest value minimizing $P(s)/s$ (see Irani et al. [2007]). Speed s_{crit} is well defined if $P(s)/s$ does not always decrease, for $s > 0$. If $P(s)/s$ always decreases, then by scheduling each job at infinite speed, or the maximum allowed speed, one obtains trivial optimal schedules. We always assume that there exists a finite speed s_{crit}.

Consider the line $f(s) = P(s_{crit})s/s_{crit}$ with slope $P(s_{crit})/s_{crit}$ passing through $(0, 0)$. This line meets the power function $P(s)$ at point $(s_{crit}, P(s_{crit}))$ (see Figure 4). In fact, $f(s)$ is the tangent to $P(s)$ at s_{crit} (assuming that $P(s)$ is differentiable at s_{crit}), since otherwise $P(s_{crit}+\epsilon)/(s_{crit}+\epsilon) < P(s_{crit})/s_{crit}$, for some $\epsilon > 0$, and s_{crit} would not be

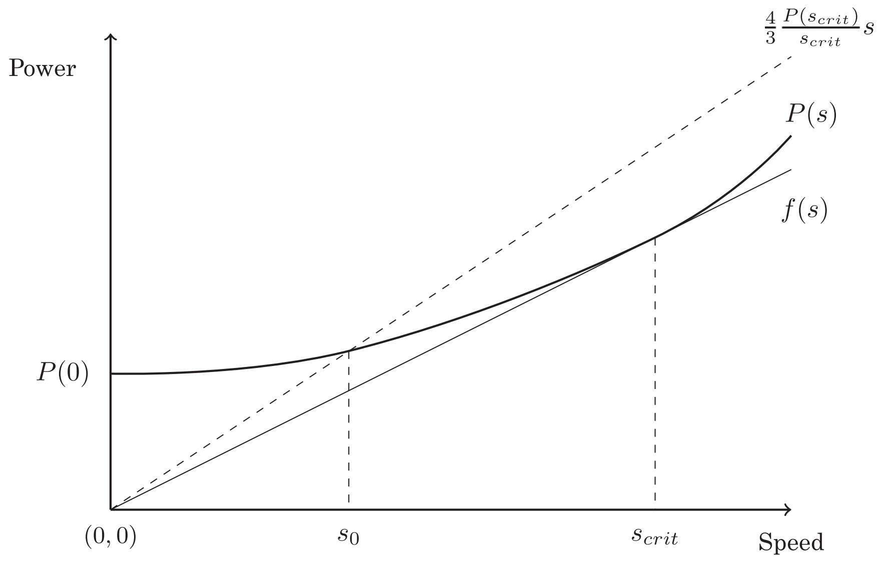

Fig. 4. The functions $P(s)$, $f(s)$, and $g(s)$.

the true critical speed. Moreover, $P(s)$ is strictly above $f(s)$ in the interval $(0, s_{crit})$, i.e. $P(s) > f(s)$ for all $s \in (0, s_{crit})$, because s_{crit} is the smallest value minimizing $P(s)/s$. Next, consider the line $g(s) = \frac{4}{3} f(s) = \frac{4}{3} P(s_{crit}) s / s_{crit}$. We have $g(s) > f(s)$, for all $s > 0$, and hence $g(s)$ intersects $P(s)$, for some speed in the range $(0, s_{crit})$. Our speed s_0 is chosen as this value satisfying $g(s_0) = P(s_0)$, and therefore $P(s_0)/s_0 = \frac{4}{3} P(s_{crit})/s_{crit}$. We remark that $g(s)$ intersects $P(s)$ only once in $(0, s_{crit})$ because $P(s)$ is convex and $g(s_{crit}) > f(s_{crit}) = P(s_{crit})$.

In the following, we present $ALG(s_0)$, for $0 \le s_0 \le s_{crit}$. Let $J_1, \ldots, J_n$ be the jobs to be processed. The scheduling horizon is $[r_{\min}, d_{\max})$, where $r_{\min} = \min_{1 \le i \le n} r_i$ is the earliest release time and $d_{\max} = \max_{1 \le i \le n} d_i$ is the latest deadline of any job. $ALG(s_0)$ operates in two phases.

Description of Phase 1: In Phase 1, the algorithm executes *YDS* and identifies job sets to be processed at speeds higher than s_0 according to this strategy. For completeness, we describe *YDS*, which works in rounds. At the beginning of a round R, let $\mathcal{J}$ be the set of unscheduled jobs and H be the available scheduling horizon. Initially, prior to the first round, $\mathcal{J} = \{J_1, \ldots, J_n\}$ and $H = [r_{\min}, d_{\max})$. During the round R, *YDS* identifies an interval $I_{\max}$ of maximum density. The *density* $\Delta(I)$ of an interval $I = [t, t')$ is defined as $\Delta(I) = \sum_{J_i \in S(I)} v_i/(t' - t)$, where $S(I) = \{J_i \in \mathcal{J} \mid [r_i, d_i) \subseteq I\}$ is the set of jobs to be processed in I. Given a maximum density interval $I_{\max} = [t, t')$, *YDS* schedules the jobs of $S(I_{\max})$ at speed $\Delta(I_{\max})$ in that interval according to the *Earliest-Deadline-First (EDF)* discipline. Then, $S(I_{\max})$ is deleted from $\mathcal{J}$, and $I_{\max}$ is removed from H. More specifically, for any unscheduled job $J_i \in \mathcal{J}$ with either $r_i \in I_{\max}$ or $d_i \in I_{\max}$, we set the new release time to $r_i' = t'$ or the new deadline to $d_i' = t$, respectively. Finally, considering the jobs of $\mathcal{J}$, all release times and deadlines of value at least t' are reduced by $t' - t$.

Algorithm $ALG(s_0)$ executes scheduling rounds of *YDS* while $\mathcal{J} \neq \emptyset$, and $\Delta(I_{\max}) > s_0$, that is, jobs are scheduled at speeds higher than s_0. At the end of Phase 1, let $\mathcal{J}_{YDS} \subseteq \{J_1, \ldots, J_n\}$ be the set of jobs scheduled according to *YDS*. Considering the original time horizon $[r_{\min}, d_{\max})$, let $I_1, \ldots, I_l$ be the sequence of disjoint, nonoverlapping intervals in which the jobs of $\mathcal{J}_{YDS}$ are scheduled. These intervals are the portions of $[r_{\min}, d_{\max})$

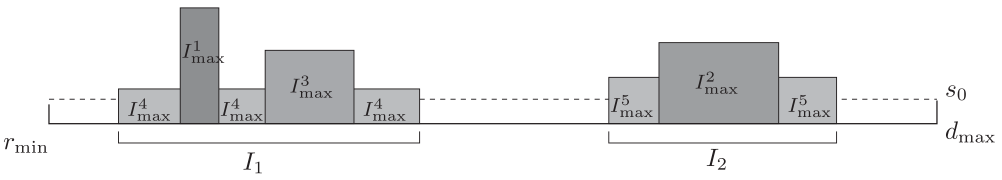

Fig. 5. Five intervals $I^j_{\max}$, $j = 1, \ldots, 5$, that form I_1 and I_2.

used by the *YDS* schedules for $\mathcal{J}_{YDS}$. Figure 5 depicts an example consisting of five maximum density intervals $I^j_{\max}$, $j = 1, \ldots, 5$, forming an interval sequence I_1, I_2. The height of an interval $I^j_{\max}$ corresponds to the density $\Delta(I^j_{\max})$. Given $I_1, \ldots, I_l$, let $I_j = [t_j, t'_j)$, where $1 \le j \le l$. We have $t'_j \le t_{j+1}$, for $j = 1, \ldots, l-1$. We remark that every job of $\mathcal{J}_{YDS}$ is completely scheduled in exactly one interval I_j.

Description of Phase 2: In Phase 2, $ALG(s_0)$ constructs a schedule for the set $\mathcal{J}_0 = \{J_1, \ldots, J_n\} \setminus \mathcal{J}_{YDS}$ of unscheduled jobs, extending the partial schedule of Phase 1. The schedule for $\mathcal{J}_0$ uses a uniform speed of s_0 and is computed by properly invoking *BCD*. Algorithm *BCD* takes as input a set of jobs, each specified by a release time, a deadline and a processing time. The given processor has an active state and a sleep state. In the active state, it consumes 1 energy unit per time unit, even if no job is currently executed. A wake-up operation requires L energy units. *BCD* computes an optimal schedule for the given job set, minimizing energy consumption.

We construct a job set $\mathcal{J}_{BCD}$ to which *BCD* is applied. Initially, we set $\mathcal{J}_{BCD} := \emptyset$. For each $J_i \in \mathcal{J}_0$, we introduce a job J'_i of processing time $v'_i = v_i/s_0$ because in a speed-s_0 schedule, J_i has to be processed for v_i/s_0 time units. The execution interval of J'_i is the same as that of J_i, that is, $r'_i = r_i$ and $d'_i = d_i$. We add J'_i to $\mathcal{J}_{BCD}$. In order to ensure that the jobs J'_i are not processed in the intervals $I_1, \ldots, I_l$, we introduce a job $J(I_j)$ for each such interval $I_j = [t_j, t'_j)$, $1 \le j \le l$. Job $J(I_j)$ has a processing time of $t'_j - t_j$, which is the length of I_j, a release time of t_j and a deadline of t'_j. These jobs $J(I_j)$, $1 \le j \le l$, are also added to $\mathcal{J}_{BCD}$. Using algorithm *BCD*, we compute an optimal schedule for $\mathcal{J}_{BCD}$, assuming that a wake-up operation of the processor incurs $L = C/P(0)$ energy units. Loosely speaking, we normalize energy by $P(0)$ so that, whenever the processor is active and even executes jobs, 1 energy unit per time unit is consumed. Let $\mathcal{S}_{BCD}$ be the schedule obtained. In a final step, we modify $\mathcal{S}_{BCD}$: Whenever a job of $\mathcal{J}_{BCD}$ is processed, the speed is set to s_0. Whenever the processor is active but idle, the speed is $s = 0$. The wake-up operations are as specified in $\mathcal{S}_{BCD}$ but incur a cost of C. In the intervals $I_1, \ldots, I_l$, we replace the jobs $J(I_j)$, $1 \le l \le l$, by *YDS* schedules for the jobs of $\mathcal{J}_{YDS}$. This schedule is output by our algorithm. A pseudocode description is given in Figure 6. Obviously, $ALG(s_0)$ has polynomial running time.

THEOREM 3.1. *Setting s_0 such that $P(s_0)/s_0 = \frac{4}{3}P(s_{crit})/s_{crit}$, $ALG(s_0)$ achieves an approximation factor of* 4/3, *for general convex power functions.*

One remark is in order here: Algorithm *BCD* assumes that time is slotted and all processing times, release times and deadlines are integers. This is no loss of generality if problem instances, in a computer, are encoded using rational numbers. If one insists on working with real numbers, in Phase 2 of $ALG(s_0)$, *BCD* can compute optimal solutions to an arbitrary precision. In this case, our algorithm achieves an approximation factor of $4/3 + \epsilon$, for any $\epsilon > 0$. In the following, we assume that the input is encoded using rational numbers.

Algorithm ALG(s_0):

Phase 1: Let $\mathcal{J} = \{J_1, \ldots, J_n\}$. While $\mathcal{J} \neq \emptyset$ and the maximum density interval $I_{\max}$ satisfies $\Delta(I_{\max}) > s_0$, execute *YDS*. At the end of the phase let $\mathcal{J}_{YDS}$ be the set of jobs scheduled according to *YDS* and $I_1, \ldots, I_l$ be the sequence of intervals used. Let $\mathcal{J}_0 = \{J_1, \ldots, J_n\} \setminus \mathcal{J}_{YDS}$.

Phase 2: Let $\mathcal{J}_{BCD} = \emptyset$. For any $J_i \in \mathcal{J}_0$, add a job J_i' with processing time $v_i' = v_i/s_0$. release time $r_i' = r_i$ and deadline $d_i' = d_i$ to $\mathcal{J}_{BCD}$. For each I_j, $1 \leq j \leq l$, add a job $J(I_j)$ with processing time $t_j' - t_j$, release time t_j and deadline t_j' to $\mathcal{J}_{BCD}$. Compute an optimal schedule $\mathcal{S}_{BCD}$ for $\mathcal{J}_{BCD}$ using *BCD* and assuming that a wake-up operation incurs $C/P(0)$ energy units. In this schedule, set the speed to s_0 whenever a job of $\mathcal{J}_{BCD}$ is processed. In the intervals $I_1, \ldots, I_l$ replace $J(I_j)$, for $1 \leq j \leq l$, by *YDS* schedules for $\mathcal{J}_{YDS}$.

Fig. 6. The algorithm $ALG(s_0)$, where $0 \leq s_0 \leq s_{crit}$.

3.2. Analysis of the Algorithm

We analyze $ALG(s_0)$ and prove Theorem 3.1. Let $\mathcal{J} = \{J_1, \ldots, J_n\}$ denote the set of all jobs to be scheduled. Furthermore, let $\mathcal{S}_A$ be the schedule constructed by $ALG(s_0)$. Let $\mathcal{S}$ be any feasible schedule for $\mathcal{J}$ and $\mathcal{J}' \subseteq \mathcal{J}$ be any subset of the jobs. We say that $\mathcal{S}$ *schedules* $\mathcal{J}'$ *according to YDS* if, considering the time intervals in which the jobs of $\mathcal{J}'$ are processed, the corresponding partial schedule is identical to the schedule constructed by *YDS* for $\mathcal{J}'$, assuming that *YDS* starts from an initially empty schedule. In Phase 1, $ALG(s_0)$ schedules job set $\mathcal{J}_{YDS} \subseteq \mathcal{J}$ according to *YDS*. Let $\mathcal{J}'_{YDS} \subseteq \mathcal{J}_{YDS}$ be the set of jobs that are processed at speeds higher than s_{crit}. Irani et al. [2007] showed that there exists an optimal schedule for the entire job set $\mathcal{J}$ that schedules $\mathcal{J}'_{YDS}$ according to *YDS*. In the following, let $\mathcal{S}_{OPT}$ be such an optimal schedule.

For the further analysis, we transform $\mathcal{S}_{OPT}$ into a schedule $\mathcal{S}_0$ that will allow us to compare $\mathcal{S}_A$ to $\mathcal{S}_{OPT}$. Schedule $\mathcal{S}_0$ schedules $\mathcal{J}_{YDS}$ according to *YDS* and all other jobs at speed s_0. The schedule has the specific feature that its idle energy does not increase too much, compared to that of $\mathcal{S}_{OPT}$. We will prove

$$E(\mathcal{S}_A) \leq E(\mathcal{S}_0) \leq \frac{4}{3} E(\mathcal{S}_{OPT}), \tag{1}$$

which establishes Theorem 3.1. The first part of (1) holds for any s_0 with $0 \leq s_0 \leq s_{crit}$. The second part holds for our choice of s_0, as specified in Theorem 3.1.

Transforming the optimal schedule: We describe the algorithm *Trans* that performs the transformation, for any $0 \leq s_0 \leq s_{crit}$. The transformation consists of four steps. As an overview, in a first step, the processor speeds of jobs of $\mathcal{J}_0$ that are originally executed at speeds lower than s_0 are raised to s_0. Then, in a second step, $\mathcal{J}_{YDS}$ is scheduled in $I_1, \ldots, I_l$ using *YDS*. In the third and fourth steps, all jobs of $\mathcal{J}_0$ are scheduled at speed exactly s_0. In the original schedule, some jobs of $\mathcal{J}_0$ might be processed at speeds higher than s_0. In order to obtain a feasible schedule, in which all jobs of $\mathcal{J}_0$ are processed using speed s_0, we may have to resort to times where the processor is idle or in the sleep state in $\mathcal{S}_{OPT}$.

Step 1: Given $\mathcal{S}_{OPT}$, *Trans* first raises processor speeds to s_0. For any job $J_i \in \mathcal{J}_0 = \mathcal{J} \setminus \mathcal{J}_{YDS}$ that is processed at a speed smaller than s_0, the speed is raised to s_0. The processing of J_i can be done at any time in the intervals reserved for J_i in $\mathcal{S}_{OPT}$. This speed increase generates processor idle times. At those times, the processor remains in the active state. The state transitions of the schedule are not affected. Let $\mathcal{S}_{0,1}$ be the schedule obtained after this step. We will use this schedule later when analyzing idle energy.

Step 2: Next, *Trans* schedules $\mathcal{J}_{YDS}$ according to *YDS* in the intervals $I_1, \ldots, I_l$. At the beginning of an interval $I_j = [t_j, t'_j)$, a wake-up operation has to be performed if the processor is originally in the sleep state at time t_j. Similarly, a power-down operation is performed at the end of the interval if the processor is in the sleep state at time t'_j.

The major part of the transformation consists in scheduling the jobs of $\mathcal{J}_0$ at the remaining times. In the scheduling horizon $[r_{\min}, d_{\max})$, let $I'_1, \ldots, I'_{l'}$ be the time intervals not covered by $I_1, \ldots, I_l$, that is, the sequences $I_1, \ldots, I_l$ and $I'_1, \ldots, I'_{l'}$ form a partition of $[r_{\min}, d_{\max})$. Within $I'_1, \ldots, I'_{l'}$, let $T_1, \ldots, T_m$ be the sequence of intervals in which the processor resides in the active state in $\mathcal{S}_{OPT}$ and hence in $\mathcal{S}_{0,1}$. The processor may or may not execute jobs at those times. Each T_k is a subinterval of some I'_j, for $1 \leq k \leq m$ and $1 \leq j \leq l'$. An interval I'_j may contain several T_k if the processor transitions to the sleep state in between. An interval I'_j does not contain any T_k if the processor resides in the sleep state throughout I'_j. In order to schedule $\mathcal{J}_0$ in $I'_1, \ldots, I'_{l'}$, *Trans* performs two passes over the schedule. The speed used for any job of $\mathcal{J}_0$ is equal to s_0. In the first pass (Step 3), *Trans* assigns as much work as possible to the intervals $T_1, \ldots, T_m$. In a second pass (Step 4), *Trans* constructs a feasible schedule for $\mathcal{J}_0$, resorting to times at which the processor is in the sleep state. Jobs are always scheduled according to the *Earliest Deadline First (EDF)* discipline.

Step 3: In the first pass, *Trans* sweeps over the intervals $T_1, \ldots, T_m$ and constructs an *EDF schedule* for $\mathcal{J}_0$, that is, at any time it schedules a job having the earliest deadline among the available unfinished jobs in $\mathcal{J}_0$. A job J_i is *available* at any time t if $t \in [r_i, d_i)$.

The schedule obtained after the first pass might not be feasible in that some jobs are not processed completely. The intervals $T_1, \ldots, T_m$ might be too short to execute all jobs of $\mathcal{J}_0$ at speed s_0, considering in particular the times when the jobs are available for processing. In the second pass *Trans* schedules the remaining work. After the first pass, let p_i be the total time for which $J_i \in \mathcal{J}_0$ is processed in the current schedule. Let $\delta_i = v_i/s_0 - p_i$ be the remaining time for which J_i has to be executed.

Step 4: In the second pass, *Trans* considers the jobs of $\mathcal{J}_0$ in nondecreasing order of deadlines; ties may be broken arbitrarily. For each $J_i \in \mathcal{J}_0$ with $\delta_i > 0$, the following steps are executed. In the current schedule, let τ_i, where $\tau_i < d_i$, be the last point of time such that $[\tau_i, d_i)$ contains exactly δ_i time units at which the processor does not execute jobs. In a correctness proof given in Lemma 3.2, we will show that τ_i is well defined. We distinguish two cases.

Case (a): If the processor resides in the sleep state throughout $[\tau_i, d_i)$, then we can easily modify the schedule. Let τ'_i, where $\tau'_i < \tau_i$, be the most recent time when the processor transitions to the sleep state. We will prove $\tau'_i \geq r_i$. *Trans* modifies the schedule by processing J_i in the interval $[\tau'_i, \tau'_i + \delta_i)$; then the processor is transitioned to the sleep state.

Case (b): If the processor resides in the active state at any time in $[\tau_i, d_i)$, then *Trans* constructs an *EDF* schedule for the remaining work of J_i and the work of $\mathcal{J}_0$ currently processed in $[\tau_i, d_i)$. Formally, *Trans* constructs a small problem instance $\mathcal{W}$, representing the work of $\mathcal{J}_0$ to be done in $[\tau_i, d_i)$. For each job $J_k \in \mathcal{J}_0$ that is currently processed for p'_k time units in $[\tau_i, d_i)$, *Trans* adds a job J'_k of processing volume $v'_k = p'_k s_0$ to $\mathcal{W}$. The job's release time and deadline are the same as those of J_k, that is, $r'_k = r_k$ and $d'_k = d_k$. If $\mathcal{W}$ already contains a job J'_i associated with J_i, we increase v'_i by $\delta_i s_0$. Otherwise, a new job J'_i of processing volume $v'_i = \delta_i s_0$, release time $r'_i = r_i$, and deadline $d'_i = d_i$ is added to $\mathcal{W}$. *Trans* then generates an *EDF* schedule for $\mathcal{W}$ in $[\tau_i, d_i)$, ignoring the intervals $I_1, \ldots, I_l$ that may be contained in $[\tau_i, d_i)$. Again, in Lemma 3.2, we will show that this is always possible. In the modified schedule, the processor executes jobs throughout $[\tau_i, d_i)$ and any wake-up operations performed within the interval are

Algorithm Trans:

1. Given $\mathcal{S}_{OPT}$, for any job of $\mathcal{J}_0$ processed at a speed smaller than s_0, raise the speed to s_0. Let $\mathcal{S}_{0,1}$ be the resulting schedule.
2. In $I_1, \ldots, I_l$ schedule $\mathcal{J}_{YDS}$ according to *YDS*.
3. In $T_1, \ldots, T_m$ construct an *EDF* schedule for $\mathcal{J}_0$ using speed s_0. For any $J_i \in \mathcal{J}_0$, let p_i be the time for which J_i is processed and $\delta_i = v_i/s_0 - p_i$.
4. Consider jobs of $\mathcal{J}_0$ in non-decreasing order of deadlines. For each J_i with $\delta_i > 0$, execute the following steps. In the current schedule $\mathcal{S}$, let $[\tau_i, d_i)$ be the shortest interval containing exactly δ_i time units at which the processor does not execute jobs.
   If the processor is in the sleep state throughout $[\tau_i, d_i)$, schedule J_i in $[\tau_i', \tau_i' + \delta_i)$, where $\tau_i' < \tau_i$ is the most recent time at which the processor powers down.
   If the processor is in the active state at some time in $[\tau_i, d_i)$, then for any $J_k \in \mathcal{J}_0$ that is processed for $p_k' > 0$ time units in $[\tau_i, d_i)$, add an entry (J_k', r_k, d_k, v_k') with $v_k' = p_k' s_0$ to $\mathcal{W}$. If $\mathcal{W}$ contains an entry for J_i, increase v_i' by $\delta_i s_0$; otherwise add an entry (J_i', r_i, d_i, v_i') with $v_i' = \delta_i s_0$. Construct an *EDF* schedule for $\mathcal{W}$ in $[\tau_i, d_i)$, ignoring the intervals $I_1, \ldots, I_l$ where $\mathcal{J}_{YDS}$ is scheduled.

Fig. 7. The algorithm *Trans*.

canceled. However, a wake-up operation must be performed at time τ_i if the processor is in the sleep state immediately before time τ_i. Similarly, the processor powers down at the end of $[\tau_i, d_i)$ if the processor is in the sleep state at time d_i.

A summary of *Trans* is given in Figure 7. The following lemma shows correctness of *Trans*.

LEMMA 3.2. *Trans constructs a feasible schedule $\mathcal{S}_0$ in which all jobs of $\mathcal{J}$ are completely scheduled.*

PROOF. All jobs of $\mathcal{J}_{YDS}$ are feasibly and completely scheduled in $I_1, \ldots, I_l$. Therefore, it suffices to show that *Trans* constructs a feasible and complete schedule for $\mathcal{J}_0$. For ease of exposition, we now black out $I_1, \ldots, I_l$ in the scheduling horizon because these intervals are not used in Steps 3 and 4 of *Trans*. Similarly, we ignore $\mathcal{J}_{YDS}$. In the condensed time horizon, consisting of $I_1', \ldots, I_{l'}'$, let r_i and d_i denote the release time and deadline of any $J_i \in \mathcal{J}_0$. By the definition of Phase 1 of $ALG(s_0)$, in the condensed time horizon, any interval $[t, t')$ has a density of at most s_0.

We number the jobs of $\mathcal{J}_0$ in nondecreasing order of deadlines; ties are broken arbitrarily. In this sequence, let J_i be the i-th job, $1 \le i \le |\mathcal{J}_0|$. In the following, a schedule $\mathcal{S}$ is referred to as an *EDF schedule for* $\mathcal{J}_0$, if at any time when the processor executes a job, it processes one having the earliest deadline among the available unfinished jobs of $\mathcal{J}_0$. Schedule $\mathcal{S}$ might not process each job $J_i \in \mathcal{J}_0$ completely. We will prove that the following statement holds, for any $i = 1, \ldots, |\mathcal{J}_0|$.

(S) Let $\mathcal{S}$ be an *EDF* schedule for $\mathcal{J}_0$ in which the first $i-1$ jobs of $\mathcal{J}_0$ are processed completely. Then the execution of Step 4 of *Trans* for J_i yields an *EDF* schedule for $\mathcal{J}_0$ in which the first i jobs of $\mathcal{J}_0$ are processed completely.

Lemma 3.2 then follows because the schedule constructed in Step 3 of *Trans* is an *EDF* schedule for $\mathcal{J}_0$.

Let $\mathcal{S}$ be an *EDF* schedule for $\mathcal{J}_0$ in which the first $i-1$ jobs are processed completely. Consider job J_i with its deadline d_i. If J_i is processed for $p_i = v_i/s_0$ time units, we are done. So suppose $p_i < v_i/s_0$, and let $\delta_i = v_i/s_0 - p_i$ be the additional time for which J_i

has to be executed. We first prove the following fact: Let $I = [\tau, d_i)$ be any time interval in $\mathcal{S}$ containing strictly less than δ_i time units at which the processor does not execute jobs. Then, in this interval I, the processor does not execute any jobs having a deadline after d_i.

We prove the fact by contradiction. Suppose that in I the processor executes a job whose deadline is after d_i. The processing of such a job cannot end at time d_i because $\mathcal{S}$ is an *EDF* schedule and J_i is not completely processed at time d_i. Let τ', with $\tau < \tau' < d_i$, be the earliest time such that in $[\tau', d_i)$ the processor does not execute jobs having a deadline after d_i. All the jobs J_k, with $k \neq i$, executed in $[\tau', d_i)$ have a release time of at least τ' because immediately before time τ' a job with a deadline after d_i is processed. By the same argument, J_i has a release time of at least τ'. The total processing volume of J_i and the jobs J_k, with $k \neq i$, executed in $[\tau', d_i)$ is at least $(p + \delta_i)s_0$, where p is the total time for which jobs are executed in $[\tau', d_i)$. This is the case because (i) since the jobs J_i and J_k with $k \neq i$ executed in $[\tau', d_i)$ are executed for p time units at speed s_0, they must have a volume of $p \cdot s_0$, and (ii) J_i has an additional volume of at least $\delta_i \cdot s_0$ that is not processed. However, the latter interval has a length strictly smaller than $p + \delta_i$ because I contains less than δ_i time units at which the processor does not execute any jobs. Hence, the density of $[\tau', d_i)$ is strictly higher than s_0, which contradicts the fact that in the scheduling horizon all intervals have a density of at most s_0.

Using the this fact, we can easily show that the value τ_i in Step 4 of *Trans* is well defined. Let r_0 be the earliest release time among the jobs of $\mathcal{J}_0$. If in the schedule $\mathcal{S}$ interval $[r_0, d_i)$ contained less than δ_i time units at which the processor does not execute any jobs, then by the previous fact, only jobs with a deadline of at most d_i can be processed in $[r_0, d_i)$. The total processing volume of J_i and the jobs J_k, with $k \neq i$, executed in $[r_0, d_i)$ is at least $(p + \delta_i)s_0$, where p is the total time for which jobs are executed in $[r_0, d_i)$. We obtain again a contradiction to the fact that the density of $[r_0, d_i)$ is at most s_0. Hence, $[r_0, d_i)$ contains at least δ_i time units at which the processor does not execute jobs and a value τ_i, with $\tau_i \geq r_0$, as specified in Step 4 of *Trans* can be feasibly chosen.

Next, we analyze the schedule modifications of Step 4. First, suppose that the processor is in the sleep state throughout $[\tau_i, d_i)$, considering the full time horizon given by $I_1, \ldots, I_l$ and $I'_1, \ldots, I'_{l'}$. Then, let τ'_i, with $\tau'_i < \tau_i$ be the most recent time when the processor transitions to the sleep state. The processor is in the active state immediately before τ'_i. Time τ'_i satisfies $\tau'_i \geq r_i$ because in the original schedule $\mathcal{S}_{OPT}$ job J_i was completely scheduled and the processor must be in the active state at some time while J_i is available for processing. We note that τ'_i might coincide with the end of an interval I_j, $1 \leq j \leq l$. Job J_i is now scheduled for the missing δ_i time units in $[\tau'_i, \tau'_i + \delta_i)$. The modified schedule is an *EDF* schedule for $\mathcal{J}_0$ because no further job is processed in $[\tau'_i + \delta_i, d_i)$.

Next, suppose that the processor is in the active state at some time in $[\tau_i, d_i)$, taking again into account the full time horizon. We show that when Step 4 of *Trans* is executed for J_i, all the work of $\mathcal{W}$ is completely scheduled in $[\tau_i, d_i)$. Recall that $\mathcal{S}$ is the schedule prior to the modification. By the choice of τ_i, the processor does not execute any jobs at time τ_i in $\mathcal{S}$. Choose an $\epsilon > 0$ such that the processor does not execute jobs in $[\tau_i, \tau_i + \epsilon)$. Interval $[\tau_i + \epsilon, d_i)$ contains less than δ_i time units at which the processor does not execute jobs. By the aforementioned fact, in $[\tau_i + \epsilon, d_i)$ and hence in $[\tau_i, d_i)$, only jobs with a deadline of at most d_i are executed.

Suppose that the execution of Step 4 yielded a schedule $\mathcal{S}'$ in which $[\tau_i, d_i)$ does not allocate all the work of $\mathcal{W}$. Then there must exist a time in that interval at which no job is executed. Let τ, with $\tau > \tau_i$, be the earliest time such that the processor executes jobs throughout $[\tau, d_i)$ in $\mathcal{S}'$. We argue that $\tau < d_i$, that is, some job is scheduled until time d_i: *Trans* schedules the work $\mathcal{W}$ in $[\tau_i, d_i)$ according to *EDF*. Hence, at any time when jobs with a deadline smaller than d_i can be processed, they have preference over the jobs with a deadline equal to d_i. Thus, if some work of $\mathcal{W}$ is not allocated to $[\tau_i, d_i)$, the work corresponds to jobs having a deadline of d_i. These jobs can be feasibly scheduled immediately before time d_i. Hence, $\tau < d_i$. Consider the jobs J'_k of $\mathcal{W}$ that are executed in $[\tau, d_i)$ in $\mathcal{S}'$ or are not completely allocated to $[\tau_i, d_i)$. All these jobs have a release time of at least τ because the processor does not execute any job immediately before time τ. Moreover, these jobs have a deadline of at most d_i. It follows that the total processing volume of these jobs J'_k and hence of the original jobs J_k is greater than $(d_i - \tau)s_0$, contradicting the fact that $[\tau, d_i)$ has a density of at most s_0.

It remains to show that $\mathcal{S}'$ is an *EDF* schedule for $\mathcal{J}_0$. In the interval $[r_0, \tau_i)$, schedule $\mathcal{S}$ and hence $\mathcal{S}'$ are *EDF* schedules for $\mathcal{J}_0$. In $[\tau_i, d_i)$, both schedules execute jobs having a deadline of at most d_i. In $[d_i, d_0)$, where d_0 denotes the maximum deadline of any job of $\mathcal{J}_0$, jobs with a deadline greater than d_i are scheduled. Schedule $\mathcal{S}'$ represents an *EDF* schedule for $\mathcal{J}_0$ because (a) *Trans* schedules $\mathcal{W}$ according to *EDF* and (b) in $[d_i, d_0)$ schedule $\mathcal{S}$ and hence $\mathcal{S}'$ process the respective workload according to *EDF*. □

Analyzing energy: We first analyze idle energy. In Step 1 of *Trans*, when speeds are raised to s_0, the idle energy increases. We will analyze this increase later. Lemmas 3.3 and 3.4 imply that Steps 2–4 of *Trans* do not cause a further increase. Recall that $\mathcal{S}_{0,1}$ denotes the schedule obtained after Step 1 of *Trans*. Lemmas 3.3, 3.4, and 3.5 hold for any speed s_0, with $0 \leq s_0 \leq s_{crit}$.

Consider the schedule obtained after Step 3 of *Trans*, which we denote by $\mathcal{S}_{0,3}$.

LEMMA 3.3. *It holds that* $E_i(\mathcal{S}_{0,3}) \leq E_i(\mathcal{S}_{0,1})$.

PROOF. In order to prove $E_i(\mathcal{S}_{0,3}) \leq E_i(\mathcal{S}_{0,1})$, we transform $\mathcal{S}_{0,1}$ into $\mathcal{S}_{0,3}$ without increasing the idle energy. Considering the scheduling horizon $[r_{\min}, d_{\max})$, we sweep over the schedule $\mathcal{S}_{0,1}$ from left to right. At any time t, $r_{\min} \leq t \leq d_{\max}$, we maintain a schedule $\mathcal{S}_t$. To the left of t, that is, in $[r_{\min}, t)$, $\mathcal{S}_t$ is identical to $\mathcal{S}_{0,3}$. To the right of t, in $[t, d_{\max})$, the schedule still has to be transformed. During the transformation, we maintain a buffer B that contains, for each job $J_i \in \mathcal{J}_0$, an entry (J_i, w_i) representing the amount of work that is not finished for J_i in $\mathcal{S}_t$. We maintain the invariant that, for any $J_i \in \mathcal{J}_0$, the processing volume finished for J_i in $\mathcal{S}_t$ plus w_i is equal to v_i. Jobs of $\mathcal{J}_{YDS}$ are always completely processed in the current schedule and hence need not be represented in the buffer. Initially, at time $t = r_{\min}$, B contains an entry $(J_i, 0)$, for all $J_i \in \mathcal{J}_0$. While sweeping over $\mathcal{S}_{0,1}$, we consider the full intervals $I_1, \ldots, I_l$. In the sequence $I'_1, \ldots, I'_{l'}$ we only consider the intervals $T_1, \ldots, T_m$ in which the processor is active; times at which the processor is in the sleep state can be ignored. Time t is always equal to the beginning of some interval I_j, $1 \leq j \leq l$, or equal to some time in an interval T_j, $1 \leq j \leq m$. Initially, t is set to the beginning of I_1 or T_1, depending on which of the two intervals occurs earlier in the scheduling horizon. The initial $\mathcal{S}_t$ is $\mathcal{S}_{0,1}$.

Suppose that we have already constructed a schedule $\mathcal{S}_t$ up to time t, $r_{\min} \leq t \leq d_{\max}$. If t is equal to the starting point of an interval $I_j = [t_j, t'_j)$, then we modify the schedule as follows: For any job $J_i \in \mathcal{J}_0$ that is processed for δ time units using speed s in I_j, we increase w_i by δs in the buffer B and remove the respective parts of J_i from the schedule. Then we schedule the jobs of $\mathcal{J}_{YDS}$ to be processed in I_j according to *YDS*. Recall that every job in $\mathcal{J}_{YDS}$ is completely scheduled in exactly one interval I_j and cannot be scheduled outside that interval. Next, we determine the smallest time t', where $t' \geq t'_j$, such that t' is the beginning of an interval I_k, where $1 \leq k \leq l$, or of an interval T_k, where $1 \leq k \leq m$. Time t is set to t' and the modified schedule is the new $\mathcal{S}_t$. If no such time t' exists, the transformation is complete.

Next, assume that t is a time in some interval T_j, $1 \leq j \leq m$. Determine the largest δ, $\delta > 0$, such that $[t, t + \delta) \subseteq T_j$ and the following two properties hold: (1) In $\mathcal{S}_{0,3}$

throughout $[t, t + \delta)$ the processor executes the same job J_i or does not execute any job at all. (2) In $\mathcal{S}_t$ throughout $[t, t + \delta)$, the processor executes the same job J_k or does not execute any job at all. Hence, in $[t, t + \delta)$, each of the two schedules executes either no job or exactly one job. As we consider an interval T_j, the jobs J_i and J_k possibly executed in $[t, t + \delta)$ belong to $\mathcal{J}_0$. We have to consider various cases.

Suppose that in $\mathcal{S}_{0,3}$ a job J_i is executed while in $\mathcal{S}_t$ no job is executed, considering the interval $[t, t + \delta)$. Determine the largest ϵ, where $0 \leq \epsilon \leq \delta$, such that $\epsilon s_0 \leq w_i$. In the buffer B, we reduce J_i's residual processing volume by ϵs_0. Moreover, in $\mathcal{S}_t$, determine the next $\epsilon' \geq 0$ time units in which a processing volume of $(\delta - \epsilon)s_0$ is finished for J_i. We have $\epsilon' \leq \delta$ because in $\mathcal{S}_t$ at any time $t' > t$, jobs of $\mathcal{J}_0$ are processed at a speed of at least s_0. In these ϵ' time units, we cancel the processing of J_i and set the processor speed to 0. Finally, in $[t, t + \delta)$, we process δs_0 units of J_i using speed s_0. These schedule modifications do not increase the idle energy because $\epsilon' \leq \delta$. We obtain a feasible schedule that is equal to $\mathcal{S}_{0,3}$ in $[r_{\min}, t + \delta)$.

Next, suppose that in $\mathcal{S}_t$ a job J_k is executed throughout $[t, t+\delta)$. If the used speed s is higher than s_0, then we reduce the speed to s_0 and increase w_k by $(s - s_0)\delta$ in the buffer entry (J_k, w_k). If $J_k = J_i$, we are done. If $J_k \neq J_i$, further modifications are required. Again, we determine the largest ϵ, where $0 \leq \epsilon \leq \delta$, such that $\epsilon s_0 \leq w_i$ and the next ϵ' time units in $\mathcal{S}_t$ in which a work volume of $(\delta - \epsilon)s_0$ is finished for J_i. Again, $0 \leq \epsilon' \leq \delta$. We modify $\mathcal{S}_t$ by processing J_k using speed s_0 at these ϵ' time units. This can be feasibly done because $\mathcal{S}_{0,3}$ is an *EDF* schedule and hence $d_i \leq d_k$. The remaining work of J_k is assigned to B by increasing w_k by $(\delta - \epsilon')s_0$. In $[t, t + \delta)$, we schedule J_i at speed s_0. To this end, we take ϵs_0 processing units from the buffer B and hence reduce w_i's value by the corresponding amount. Again, the idle energy of the modified schedule has not increased.

Finally, suppose that in $\mathcal{S}_{0,3}$ no job is executed in the interval $[t, t + \delta)$. This implies that among the available jobs, which can be feasibly scheduled in this interval, all jobs are finished. As $\mathcal{S}_t$ is identical to $\mathcal{S}_{0,3}$ in $[r_{\min}, t)$, $\mathcal{S}_t$ cannot process any job in $[t, t + \delta)$ either.

In all the previous cases, we obtain a (modified) schedule $\mathcal{S}_t$ that is identical to $\mathcal{S}_{0,3}$ in $[r_{\min}, t+\delta)$. The idle energy has not increased. If $t+\delta$ is still within the current interval T_j, we set $t' = t + \delta$. Otherwise we determine the smallest t' with $t' > t + \delta$ such that t' is the beginning of some I_j, where $1 \leq j \leq l$, or of some T_j, where $1 \leq j \leq m$. We set $t = t'$ and the current, modified schedule is the new $\mathcal{S}_t$. Again, if no such t' exits, the transformation is complete.

The aforementioned schedule modifications are repeated until the transformation is finally finished. □

LEMMA 3.4. *It holds that $E_i(\mathcal{S}_0) \leq E_i(\mathcal{S}_{0,1})$.*

PROOF. In Lemma 3.3, we have already shown that $E_i(\mathcal{S}_{0,3}) \leq E_i(\mathcal{S}_{0,1})$. It, therefore, remains to prove that the scheduling operations of Step 4 do not increase the idle energy: Consider a job J_i for which Step 4 is executed. If the processor is in the sleep state throughout $[\tau_i, d_i)$, then *Trans* determines the most recent time τ_i' at which the processor transitions to the sleep state. Job J_i is scheduled in the interval $[\tau_i', \tau_i' + \delta_i)$, then the processor powers down. Neither the number of wake-up operations nor the total time for which the processor is in the active state but does not process jobs increase. If the processor is in the active state at some time in $[\tau_i, d_i)$, then the *EDF* schedule generated for this interval does not increase the idle energy, either. The processor might have to transition to the active state at time τ_i. However, this cancels the subsequent wake-up operation needed to transition the processor to the active state in $[\tau_i, d_i)$. Throughout $[\tau_i, d_i)$, the processor executes jobs and no idle energy is incurred. □

The next lemma establishes the first part of inequality (1).

LEMMA 3.5. *It holds that $E(\mathcal{S}_A) \leq E(\mathcal{S}_0)$.*

PROOF. Given our job instance $\mathcal{J} = \{J_1, \dots J_n\}$, let $\mathcal{C}$ be the class of schedules that process $\mathcal{J}_{YDS}$ according to *YDS* and all jobs of $\mathcal{J}_0 = \mathcal{J} \setminus \mathcal{J}_{YDS}$ using speed s_0. Both $\mathcal{S}_A$ and $\mathcal{S}_0$ belong to $\mathcal{C}$. We prove that among the schedules of $\mathcal{C}$, $\mathcal{S}_A$ minimizes energy consumption. The lemma then follows.

Consider any $\mathcal{S} \in \mathcal{C}$. Let $E(\mathcal{J}_{YDS})$ be the energy incurred in processing $\mathcal{J}_{YDS}$ in the intervals $I_1, \dots, I_l$. In these intervals, no idle energy is incurred. Any job $J_i \in \mathcal{J}_0$ is processed for v_i/s_0 time units using speed s_0. Let T be the total time for which the processor is in the active state but does not process any jobs in $\mathcal{S}$. Furthermore, let k be the number of wake-up operations performed in $\mathcal{S}$. We have

$$E(\mathcal{S}) = E(\mathcal{J}_{YDS}) + \sum_{J_i \in \mathcal{J}_0} \frac{v_i}{s_0} P(s_0) + TP(0) + kC \tag{2}$$

$$= E(\mathcal{J}_{YDS}) - \sum_{j=1}^{l} |I_j| P(0) + \sum_{J_i \in \mathcal{J}_0} \frac{v_i}{s_0} (P(s_0) - P(0)) \tag{3}$$

$$+ P(0) \left(\sum_{j=1}^{l} |I_j| + \sum_{J_i \in \mathcal{J}_0} \frac{v_i}{s_0} + T + \frac{kC}{P(0)} \right). \tag{4}$$

Let E_1 be the sum given in (3). Furthermore, let E_2 be the expression given in the brackets of (4). Then E_1 is a fixed overhead incurred by any schedule of $\mathcal{C}$, and E_2 is the cost of a schedule $\mathcal{S}_{BCD}$ that can be obtained from $\mathcal{S}$ for a job instance $\mathcal{J}_{BCD}$ defined as follows: For any $J_i \in \mathcal{J}_0$, we add a job J_i' of processing time $v_i' = v_i/s_0$, release time $r_i' = r_i$ and deadline $d_i' = d_i$ to $\mathcal{J}_{BCD}$. For each interval $I_j = [t_j, t_j')$, where $1 \leq j \leq l$, we add a job $J(I_j)$ of processing time $t_j' - t_j$, release time t_j, and deadline t_j' to $\mathcal{J}_{BCD}$. In order to process $\mathcal{J}_{BCD}$, we are given a uniform speed processor. Whenever the processor is in the active state, one cost unit is consumed per time unit. A transition from the sleep state to the active state costs $C/P(0)$.

Given $\mathcal{S}$, we can easily derive a feasible schedule $\mathcal{S}_{BCD}$ for $\mathcal{J}_{BCD}$ that incurs a cost of E_2. In the intervals $I_1, \dots, I_l$, the *YDS* schedules are replaced by $J(I_1), \dots, J(I_l)$. The processor speed is set to a uniform speed of 1. A wake-up operation incurs a cost of $C/P(0)$. Conversely, a feasible schedule $\mathcal{S}_{BCD}$ for $\mathcal{J}_{BCD}$ incurring cost E_2 can be converted into a schedule $\mathcal{S}$ for $\mathcal{J}$ consuming an energy of $E(\mathcal{S})$ as specified in (2). In $\mathcal{S}_{BCD}$, we replace the jobs $J(I_1), \dots, J(I_l)$ by *YDS* schedules for $\mathcal{J}_{YDS}$. At all other times where the processor executes jobs, the speed is set to s_0. A wake-up operation incurs C energy units.

Therefore, the problem of finding a minimum energy schedule in $\mathcal{C}$ is equivalent to computing a minimum cost schedule for $\mathcal{J}_{BCD}$. In Phase 2, algorithm $ALG(s_0)$ solves exactly this problem. We conclude that $\mathcal{S}_A$ is a minimum energy schedule in $\mathcal{C}$. □

We proceed to prove the second part of inequality (1). Given a schedule $\mathcal{S}$ and a job $J_i \in \mathcal{J}$, let $E_p(\mathcal{S}, J_i)$ denote the processing energy incurred in executing J_i in $\mathcal{S}$. Let $\mathcal{J}_{0,1}$ be the subset of the jobs of $\mathcal{J}_0$ processed at a speed smaller than s_0 in $\mathcal{S}_{OPT}$. Moreover, let $\mathcal{J}_{0,2} = \mathcal{J}_0 \setminus \mathcal{J}_{0,1}$ be the set of remaining jobs of $\mathcal{J}_0$. We present two Lemmas 3.6 and 3.7 that hold for speeds s_0 satisfying $P(s_0)/s_0 = cP(s_{crit})/s_{crit}$, where c is a constant specified in the lemmas. As we shall see in Lemma 3.7, $c = 4/3$ gives the best approximation ratio.

LEMMA 3.6. *Let s_0 be a speed such that $P(s_0)/s_0 = cP(s_{crit})/s_{crit}$, where $c \geq 1$. Then for any $J_i \in \mathcal{J}_{YDS} \cup \mathcal{J}_{0,2}$, it holds that $E_p(\mathcal{S}_0, J_i) \leq c \cdot E_p(\mathcal{S}_{OPT}, J_i)$.*

PROOF. Recall that $\mathcal{J}'_{YDS}$, with $\mathcal{J}'_{YDS} \subseteq \mathcal{J}_{YDS}$, is the set of jobs that, using algorithm *YDS*, are processed at a speed higher than s_{crit}. We chose $\mathcal{S}_{OPT}$ such that $\mathcal{J}'_{YDS}$ is scheduled according to *YDS*. Moreover, $\mathcal{S}_0$ schedules $\mathcal{J}'_{YDS}$ according to *YDS*. Hence, any job $J_i \in \mathcal{J}'_{YDS}$ is processed at the same speed in $\mathcal{S}_0$ and $\mathcal{S}_{OPT}$. We obtain $E_p(\mathcal{S}_0, J_i) = E_p(\mathcal{S}_{OPT}, J_i)$, for any $J_i \in \mathcal{J}'_{YDS}$.

Next, consider any $J_i \in \mathcal{J}_{YDS} \setminus \mathcal{J}'_{YDS} \cup \mathcal{J}_{0,2}$. In general, if J_i is processed at speed s, then the incurred processing energy is $\frac{v_i}{s}P(s)$. Speed s_{crit} minimizes $P(s)/s$. Hence $E_p(\mathcal{S}_{OPT}, J_i) \geq \frac{v_i}{s_{crit}}P(s_{crit})$. In $\mathcal{S}_0$, job J_i is processed at a speed s_i that is at least s_0. Moreover, since $J_i \notin \mathcal{J}'_{YDS}$, we have $s_i \leq s_{crit}$. Since s_{crit} is the smallest speed minimizing $P(s)/s$, we have $P(s_0)/s_0 \geq P(s_i)/s_i \geq P(s_{crit})/s_{crit}$. By the choice of s_0, it holds that $P(s_0)/s_0 = c \cdot P(s_{crit})/s_{crit}$. Hence, $P(s_i)/s_i \leq c \cdot P(s_{crit})/s_{crit}$ and $E_p(\mathcal{S}_0, J_i) \leq c \cdot E_p(\mathcal{S}_{OPT}, J_i)$. □

We next turn to the set $\mathcal{J}_{0,1}$. For any $J_i \in \mathcal{J}_{0,1}$, algorithm *Trans* raises the speed to s_0. This speed increase causes idle energy in $\mathcal{S}_{0,1}$ and hence in $\mathcal{S}_0$ because the processor remains in the active state at all the times at which J_i was originally scheduled. For any $J_i \in \mathcal{J}_{0,1}$, let $E_i(J_i)$ be the idle energy incurred. The next lemma implies that, loosely speaking, we can charge $E_i(J_i)$ to $E_p(\mathcal{S}_0, J_i)$.

LEMMA 3.7. *Let s_0 be a speed such that $P(s_0)/s_0 = c \cdot P(s_{crit})/s_{crit}$, where $c \geq 4/3$. Then for any $J_i \in \mathcal{J}_{0,1}$, it holds that $E_p(\mathcal{S}_0, J_i) + E_i(J_i) \leq c \cdot E_p(\mathcal{S}_{OPT}, J_i)$.*

PROOF. Consider an arbitrary job $J_i \in \mathcal{J}_{0,1}$. Let T be the total time for which J_i is processed at a speed s_i, where $s_i < s_0$, in $\mathcal{S}_{OPT}$. Moreover, let T' be the total time for which J_i is executed at speed s_0 in $\mathcal{S}_{0,1}$ and $\mathcal{S}_0$. We have $T' < T$. Choose λ, where $0 \leq \lambda \leq 1$, such that $T' = \lambda T$. By raising the processor speed to s_0, an idle energy of $(T - T')P(0) = (1-\lambda)TP(0)$ is incurred. In $\mathcal{S}_0$, the processing energy for J_i is $T'P(s_0) = \lambda TP(s_0)$. Hence, $E_p(\mathcal{S}_0, J_i) + E_i(J_i) = \lambda TP(s_0) + (1-\lambda)TP(0) = T(P(0) + \lambda(P(s_0) - P(0)))$. The processing volume of J_i is $T's_0 = \lambda T s_0$, and hence J_i is executed at speed λs_0 in $\mathcal{S}_{OPT}$. We obtain $E_p(\mathcal{S}_{OPT}, J_i) = TP(\lambda s_0)$. We substitute λs_0 by s, which implies $\lambda = s/s_0$. Let

$$R(s) = \frac{E_p(\mathcal{S}_0, J_i) + E_i(J_i)}{E_p(\mathcal{S}_{OPT}, J_i)} = \frac{P(0) + \frac{s}{s_0}(P(s_0) - P(0))}{P(s)}. \tag{5}$$

In order to establish the lemma, we have to show that the ratio $R(s)$ is upper bounded by c, for all s with $0 \leq s \leq s_0$. Consider the ratio in (5). The numerator is a line, which we denote by $f(s)$, passing through $(0, P(0))$ and $(s_0, P(s_0))$. We have to compare $f(s)$ to $P(s)$ in the interval $[0, s_0]$ (see Figure 8).

Let $g(s)$ be the line passing through $(s_0, P(s_0))$ and $(s_{crit}, P(s_{crit}))$. We will use this line to lower bound $P(s)$. Line $g(s)$ has a slope of $(P(s_{crit}) - P(s_0))/(s_{crit} - s_0)$, which is smaller than $P(s_{crit})/s_{crit}$. This holds true because $(P(s_{crit}) - P(s_0))/(s_{crit} - s_0) < P(s_{crit})/s_{crit}$ is equivalent to $P(s_{crit})/s_{crit} < P(s_0)/s_0$. The latter inequality is satisfied as s_{crit} is the smallest value minimizing $P(s)/s$. The line connecting $(0, 0)$ and $(s_{crit}, P(s_{crit}))$ has a slope of $P(s_{crit})/s_{crit}$. Since g has a smaller slope, we have $g(0) \geq 0$.

Moreover, we have $f(s_0) = P(s_0) = g(s_0)$. The power function P is convex, and hence the slope of f cannot be greater than the slope of g. Since $f(0) = P(0)$, it follows $g(0) \leq P(0)$. We conclude $0 \leq g(0) \leq P(0)$. Choose a constant a, where $0 \leq a \leq 1$, such that $g(0) = aP(0)$. Then $g(s)$, which passes through $(s_{crit}, P(s_{crit}))$, is defined as follows.

$$g(s) = aP(0) + \frac{P(s_{crit}) - aP(0)}{s_{crit}}s.$$

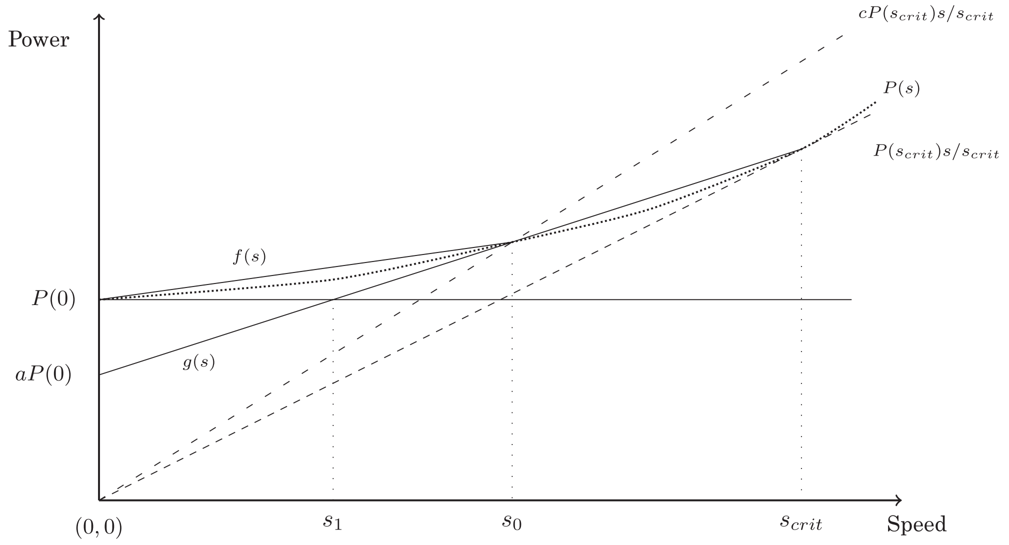

Fig. 8. Lines $f(s)$ and $g(s)$.

It holds that $g(s_0) = P(s_0)$. By the choice of s_0, we have that $P(s_0) = cP(s_{crit})s_0/s_{crit}$. We solve the equation $g(s_0) = cP(s_{crit})s_0/s_{crit}$ for s_0. Moreover, let s_1 be the value satisfying $g(s_1) = P(0)$. Then

$$s_0 = \frac{aP(0)s_{crit}}{(c-1)P(s_{crit}) + aP(0)}, \quad \text{and} \quad s_1 = \frac{(1-a)P(0)s_{crit}}{P(s_{crit}) - aP(0)}.$$

Line g passes through $(s_0, P(s_0))$ and $(s_{crit}, P(s_{crit}))$. Hence, by convexity of P, we have $g(s) \le P(s)$, for all s with $0 \le s \le s_0$. Moreover, $P(s) \ge P(0)$, for all $s \ge 0$. Let $P'(s)$ be the following function:

$$P'(s) = \begin{cases} P(0), & 0 \le s < s_1, \\ g(s), & s_1 \le s \le s_0. \end{cases}$$

Then $P'(s) \le P(s)$ for all $0 \le s \le s_0$, and instead of upper bounding $R(s)$ by c we will show that $R'(s) = (P(0) + \frac{s}{s_0}(P(s_0) - P(0)))/P'(s)$ is upper bounded by c, for all $0 \le s \le s_0$. Line $f(s)$ is again the numerator of $R'(s)$. The function f is increasing and $f(s) \ge P(0)$, for all $s \ge 0$. In the interval $[0, s_1)$, we have $P'(s) = P(0)$. Hence, for any $s \in [0, s_1)$, it holds that $R'(s) \le f(s_1)/P(0) = f(s_1)/g(s_1)$. In the interval $[s_1, s_0]$, we have $R'(s) = f(s)/g(s)$. Both functions f and g are increasing. It holds that $f(s_1) \ge P(0) = g(s_1)$ and $f(s_0) = P(s_0) = g(s_0)$. It follows $R'(s) \ge R'(s')$, for all $s_1 \le s < s' \le s_0$. Hence, $R'(s)$ is maximized for $s = s_1$, and it suffices to show that $R'(s_1) = f(s_1)/g(s_1) = f(s_1)/P(0)$ is upper bounded by c. We have

$$\begin{aligned} f(s_1) &= P(0) + \frac{s_1}{s_0}(P(s_0) - P(0)) \\ &= P(0) + \left(c \cdot \frac{P(s_{crit})}{s_{crit}} - \frac{P(0)}{s_0}\right) s_1 \\ &= P(0) + \left(c \cdot \frac{P(s_{crit})}{s_{crit}} - \frac{(c-1)P(s_{crit}) + aP(0)}{as_{crit}}\right) \cdot \frac{(1-a)P(0)s_{crit}}{P(s_{crit}) - aP(0)} \\ &= P(0) + \left(\frac{(ca - c + 1)P(s_{crit}) - aP(0)}{P(s_{crit}) - aP(0)}\right) \cdot \frac{1-a}{a} \cdot P(0). \end{aligned}$$

The second equation holds because $P(s_0)/s_0 = cP(s_{crit})/s_{crit}$. The third equation is obtained by plugging in the values of s_0 and s_1. Hence,

$$R'(s_1) = 1 + \left(\frac{(ca - c + 1)P(s_{crit}) - aP(0)}{P(s_{crit}) - aP(0)}\right)\frac{1-a}{a},$$

and it remains to show that the aforementioned term of $R'(s_1)$ is upper bounded by c. This is equivalent to proving $-(c(a-1/2)^2 + 3c/4 - 1)P(s_{crit}) + (ca^2 - a)P(0) \le 0$. We have $-(c(a-1/2)^2 + 3c/4 - 1) \le 0$, for any $0 \le a \le 1$ and $c \ge 4/3$. Furthermore, $P(s_{crit}) \ge P(0) > 0$. We conclude, as desired, $-(c(a-1/2)^2 + 3c/4 - 1)P(s_{crit}) + (ca^2 - a)P(0) \le (ca - c + 1 - a)P(0) = (c-1)(a-1)P(0) \le 0$. □

Finally, we need the next lemma in order to show the second part of inequality (1).

LEMMA 3.8. *Assume that there exists some c so that (i) for any $J_i \in \mathcal{J}_{0,1}$, $E_p(\mathcal{S}_0, J_i) + E_i(J_i) \le c \cdot E_p(\mathcal{S}_{OPT}, J_i)$, and (ii) for any $J_i \in \mathcal{J}_{YDS} \cup \mathcal{J}_{0,2}$, $E_p(\mathcal{S}_0, J_i) \le c \cdot E_p(\mathcal{S}_0, J_i)$. Then, it holds that $E(\mathcal{S}_0) \le c \cdot E(\mathcal{S}_{OPT})$.*

PROOF. Consider again the schedule $\mathcal{S}_{0,1}$ obtained after Step 1 of *Trans*. Compared to $\mathcal{S}_{OPT}$, the idle energy increases by $\sum_{J_i \in \mathcal{J}_{0,1}} E_i(J_i)$. Hence, $E_i(\mathcal{S}_{0,1}) = E_i(\mathcal{S}_{OPT}) + \sum_{J_i \in \mathcal{J}_{0,1}} E_i(J_i)$ and by Lemma 3.4 $E_i(\mathcal{S}_0) \le E_i(\mathcal{S}_{OPT}) + \sum_{J_i \in \mathcal{J}_{0,1}} E_i(J_i)$.

We have

$$\begin{aligned}
E(\mathcal{S}_0) &= E_p(\mathcal{S}_0) + E_i(\mathcal{S}_0)\\
&= \sum_{J_i \in \mathcal{J}_{YDS} \cup \mathcal{J}_{0,2}} E_p(\mathcal{S}_0, J_i) + \sum_{J_i \in \mathcal{J}_{0,1}} E_p(\mathcal{S}_0, J_i) + E_i(\mathcal{S}_0)\\
&\le \sum_{J_i \in \mathcal{J}_{YDS} \cup \mathcal{J}_{0,2}} c \cdot E_p(\mathcal{S}_{OPT}, J_i) + \sum_{J_i \in \mathcal{J}_{0,1}} (E_p(\mathcal{S}_0, J_i) + E_i(J_i)) + E_i(\mathcal{S}_{OPT})\\
&\le \sum_{J_i \in \mathcal{J}_{YDS} \cup \mathcal{J}_{0,2}} c \cdot E_p(\mathcal{S}_{OPT}, J_i) + \sum_{J_i \in \mathcal{J}_{0,1}} c \cdot E_p(\mathcal{S}_{OPT}, J_i) + E_i(\mathcal{S}_{OPT})\\
&= c \cdot E_p(\mathcal{S}_{OPT}) + E_i(\mathcal{S}_{OPT}) \le c \cdot E(\mathcal{S}_{OPT}). \quad \square
\end{aligned}$$

Recall that we chose s_0 such that $P(s_0)/s_0 = \frac{4}{3}P(s_{crit})/s_{crit}$. Combining Lemmas 3.6, 3.7, and 3.8 for $c = \frac{4}{3}$ proves the second part of inequality (1).

4. POWER FUNCTIONS $P(s) = \beta s^{\alpha} + \gamma$

We develop an improved approximation guarantee for the family of power functions $P(s) = \beta s^{\alpha} + \gamma$, where $\alpha > 1$ and $\beta, \gamma > 0$ are constants. The critical speed is $s_{crit} = \sqrt[\alpha]{\gamma/(\beta(\alpha-1))}$. Let $s_{\alpha} = \frac{c}{c+1}s_{crit}$, where $c = 117/20 = 5.85$, and $ALG(s_{\alpha})$ be the algorithm obtained from $ALG(s_0)$ be setting s_0 to s_{α}.

THEOREM 4.1. *$ALG(s_{\alpha})$ achieves an approximation factor of $(c+1)/c = 137/117 < 1.171$.*

The proof of Theorem 4.1 proceeds along the same lines as the proof of Theorem 3.1, and we have to replace Lemmas 3.6 and 3.7.

LEMMA 4.2. *Let $s_{\alpha} = \frac{c}{c+1}s_{crit}$. Then for any $J_i \in \mathcal{J}_{YDS} \cup \mathcal{J}_{0,2}$, it holds that $E_p(\mathcal{S}_0, J_i) \le \frac{c+1}{c}E_p(\mathcal{S}_{OPT}, J_i)$.*

PROOF. For any $J_i \in \mathcal{J}'_{YDS}$, it holds that $E_p(\mathcal{S}_0, J_i) = E_p(\mathcal{S}_{OPT}, J_i)$. Recall that $s_{\alpha} = c/(c+1)s_{crit}$. This implies $P(s_{\alpha})/s_{\alpha} = \frac{c+1}{c}P(\frac{c}{c+1}s_{crit})/s_{crit} \le \frac{c+1}{c}P(s_{crit})/s_{crit}$. Hence, for any $J_i \in \mathcal{J}_{YDS} \setminus \mathcal{J}'_{YDS} \cup \mathcal{J}_{0,2}$, we have $E_p(\mathcal{S}_0, J_i) \leq v_i P(s_\alpha)/s_\alpha \leq \frac{c+1}{c} v_i P(s_{crit})/s_{crit} \leq \frac{c+1}{c} E_p(\mathcal{S}_{OPT}, J_i)$. $\square$

We need the following technical lemma. The proof requires the Lambert W function whose defining equation, for any number x, is $x = W(x)e^{W(x)}$. The function is double-valued in the interval $(-1/e, 0)$, and $W_{-1}(x)$ and $W_0(x)$ denote the lower and upper branches, respectively.

LEMMA 4.3. *The inequality*

$$\frac{\lambda\left(\frac{c}{c+1}\right)^\alpha + \alpha - 1}{\lambda^\alpha\left(\frac{c}{c+1}\right)^\alpha + \alpha - 1} \leq \frac{c+1}{c},$$

holds for $c = 117/20$*, all* $\alpha > 1$ *and all* $0 \leq \lambda \leq 1$.

PROOF. For simplicity, set $x = \frac{c+1}{c}$. We then have to show $\frac{\lambda\left(\frac{1}{x}\right)^\alpha + \alpha - 1}{\lambda^\alpha\left(\frac{1}{x}\right)^\alpha + \alpha - 1} \leq x$, which is equivalent to

$$\begin{aligned} & \lambda(1/x)^\alpha + (\alpha - 1) \leq x\lambda^\alpha(1/x)^\alpha + x(\alpha - 1) \\ \Leftrightarrow \quad & (x\lambda^\alpha - \lambda)(1/x)^\alpha + (x-1)(\alpha - 1) \geq 0. \end{aligned}$$

Set $f(\lambda) = (x\lambda^\alpha - \lambda)(1/x)^\alpha + (x-1)(\alpha-1)$. It follows that $f'(\lambda) = \alpha\lambda^{\alpha-1}(1/x)^{\alpha-1} - (1/x)^\alpha$ and $f''(\lambda) = \alpha(\alpha-1)\lambda^{\alpha-2}(1/x)^{\alpha-1}$, which is nonnegative. This implies that f is convex and gets minimized for λ satisfying $\alpha\lambda^{\alpha-1} = \frac{1}{x}$. Hence,

$$\lambda = \left(\frac{1}{x\alpha}\right)^{\frac{1}{\alpha-1}}.$$

Thus, it is sufficient to show that

$$\left[x\left(\frac{1}{x\alpha}\right)^{\alpha/(\alpha-1)} - \left(\frac{1}{x\alpha}\right)^{1/(\alpha-1)}\right]\left(\frac{1}{x}\right)^\alpha + (x-1)(\alpha-1) \geq 0,$$

which is equivalent to

$$\begin{aligned} & \left(\frac{1}{x\alpha}\right)^{\frac{1}{\alpha-1}}\left(\frac{1}{\alpha} - 1\right)\left(\frac{1}{x}\right)^\alpha + (x-1)(\alpha-1) \geq 0 \\ \Leftrightarrow \quad & \left(\frac{1}{x\alpha}\right)^{\frac{1}{\alpha-1}}\frac{1}{\alpha}\left(\frac{1}{x}\right)^\alpha \leq x - 1. \end{aligned}$$

By setting x back to $\frac{c+1}{c}$, we obtain

$$\begin{aligned} \left(\frac{c}{(c+1)\alpha}\right)^{\frac{1}{\alpha-1}}\left(\frac{c}{c+1}\right)^\alpha &\leq \frac{1}{c} \cdot \alpha \\ \Leftrightarrow \quad c^{\alpha+1+\frac{1}{\alpha-1}} &\leq \alpha^{1+\frac{1}{\alpha-1}} \cdot (c+1)^{\alpha+\frac{1}{\alpha-1}} \\ \Leftrightarrow \quad c^{\alpha^2} &\leq \alpha^\alpha \cdot (c+1)^{\alpha^2-\alpha+1} \\ \Leftrightarrow \ln(c+1) - \ln(c) + \frac{1-\alpha}{\alpha^2}\ln(c+1) + \frac{1}{\alpha}\ln(\alpha) &\geq 0. \end{aligned}$$

Let $g(\alpha) = \ln(c+1) - \ln(c) + \frac{1-\alpha}{\alpha^2}\ln(c+1) + \frac{1}{\alpha}\ln(\alpha)$. We wish to show that $g(\alpha) \geq 0$. It suffices to show that $g(\alpha) \geq 0$ for the extrema of g in $(1, +\infty)$. These are attained when

$\alpha \to 1^+$, $\alpha \to +\infty$, and at the points where $g'(\alpha) = 0$. For the first two, we have

$$\lim_{\alpha\to 1^+} g(\alpha) = \lim_{\alpha\to+\infty} g(\alpha) = \ln(c+1) - \ln(c),$$

which is strictly positive for any $c > 0$. We next determine the roots of $g'(\alpha) = 0$. It holds that

$$g'(\alpha) = \frac{\alpha-2}{\alpha^3}\ln(c+1) - \frac{\ln(\alpha)}{\alpha^2} + \frac{1}{\alpha^2}.$$

Thus,

$$\begin{aligned} g'(\alpha) &= 0 \\ \Leftrightarrow\quad (\alpha-2)\ln(c+1) - \alpha\ln(\alpha) + \alpha &= 0 \\ \Leftrightarrow\quad \ln\left(\frac{(c+1)^{\alpha-2}}{\alpha^\alpha}\right) &= -\alpha \\ \Leftrightarrow\quad (c+1)^\alpha e^\alpha &= \alpha^\alpha (c+1)^2 \\ \Leftrightarrow\quad \frac{1}{\alpha}(c+1)e &= (c+1)^{\frac{2}{\alpha}}. \end{aligned}$$

By setting $t = -\frac{2}{\alpha}$, we obtain

$$-\frac{c+1}{2}et = (c+1)^{-t} \Leftrightarrow t(c+1)^t = -\frac{2}{(c+1)e}$$

$$\text{and}\quad t = \frac{W\left(-\frac{2\ln(c+1)}{(c+1)e}\right)}{\ln(c+1)},$$

giving the following roots for $g'(\alpha) = 0$:

$$\alpha_1 = -\frac{2\ln(c+1)}{W_0\left(-\frac{2\ln(c+1)}{(c+1)e}\right)}, \quad \text{and} \quad \alpha_2 = -\frac{2\ln(c+1)}{W_{-1}\left(-\frac{2\ln(c+1)}{(c+1)e}\right)}.$$

By evaluating g at α_1 and α_2, with $c = 117/20$, we get 0.2186875389 and 0.0000352487, respectively. Since both values are nonnegative, the proof is complete. □

LEMMA 4.4. *Let $s_\alpha = \frac{c}{c+1}s_{crit}$. Then for any $J_i \in \mathcal{J}_{0,1}$, it holds that $E_p(\mathcal{S}_0, J_i) + E_i(J_i) \leq \frac{c+1}{c}E_p(\mathcal{S}_{OPT}, J_i)$.*

PROOF. Let J_i be any job in $\mathcal{J}_{0,1}$. Let T be the total time for which J_i is executed using a speed of $s_i < s_\alpha$ in $\mathcal{S}_{OPT}$. Let $T' < T$ be the total time for which the job is processed at speed s_α in $\mathcal{S}_{0,1}$ and $\mathcal{S}_0$. Choose λ, with $0 \leq \lambda \leq 1$, such that $T' = \lambda T$. We have $s_\alpha = \frac{c}{c+1}s_{crit}$ and $s_{crit} = \sqrt[\alpha]{\gamma/(\beta(\alpha-1))}$. Therefore, $P(s_\alpha) = (\frac{c}{c+1})^\alpha\gamma/(\alpha-1) + \gamma$ and $E_p(\mathcal{S}_0, J_i) + E_i(J_i) \leq \lambda T P(s_\alpha) + (1-\lambda)TP(0) = T(\lambda(\frac{c}{c+1})^\alpha\gamma/(\alpha-1) + \gamma)$. In $\mathcal{S}_{OPT}$ job J_i is processed at speed λs_α, and thus $E_p(\mathcal{S}_{OPT}, J_i) = TP(\lambda s_\alpha) = T(\lambda^\alpha(\frac{c}{c+1})^\alpha\gamma/(\alpha-1) + \gamma)$. By Lemma 4.3, the ratio

$$\frac{E_p(\mathcal{S}_0, J_i) + E_i(J_i)}{E_p(\mathcal{S}_{OPT}, J_i)} \leq \frac{\lambda(\frac{c}{c+1})^\alpha\gamma/(\alpha-1) + \gamma}{\lambda^\alpha(\frac{c}{c+1})^\alpha\gamma/(\alpha-1) + \gamma} = \frac{\lambda(\frac{c}{c+1})^\alpha + \alpha - 1}{\lambda^\alpha(\frac{c}{c+1})^\alpha + \alpha - 1}$$

is upper bounded by $(c+1)/c$. □

Using Lemmas 4.2 and 4.4 as well as Lemma 3.8 of Section 3.2, we can show $E(\mathcal{S}_0) \leq \frac{c+1}{c}E(\mathcal{S}_{OPT})$.

5. REVISITING S_{CRIT}-SCHEDULES

Let $ALG(s_{crit})$ be the algorithm $ALG(s_0)$, where s_0 is set to s_{crit}.

THEOREM 5.1. *$ALG(s_{crit})$ achieves an approximation factor of 2, for general convex power functions.*

THEOREM 5.2. *$ALG(s_{crit})$ achieves an approximation factor of $eW_{-1}(-e^{-1-\frac{1}{e}})/(eW_{-1}(-e^{-1-\frac{1}{e}})+1)$, for power functions $P(s) = \beta s^\alpha + \gamma$, where $\alpha > 1$ and $\beta, \gamma > 0$.*

The ratio given in Theorem 5.2 is smaller than 1.211. In order to prove Theorems 5.1 and 5.2, we have to replace Lemmas 3.6 and 3.7. For $s_0 = s_{crit}$, the set $\mathcal{J}_{YDS} \setminus \mathcal{J}'_{YDS}$ of jobs scheduled according to *YDS* but at speeds of at most s_{crit} is empty. Hence, $\mathcal{J}_{YDS} = \mathcal{J}'_{YDS}$. Let $\mathcal{S}_{OPT}$ denote again an optimal schedule in which the jobs of $\mathcal{J}_{YDS}$ are scheduled according to *YDS*. Schedule $\mathcal{S}_0$ is obtained from $\mathcal{S}_{OPT}$ by applying *Trans* and satisfies the additional property that all jobs of $\mathcal{J}_0$ are executed at speed s_{crit}. Any job $J_i \in \mathcal{J}_{YDS}$ is processed at the same speed in $\mathcal{S}_0$ and $\mathcal{S}_{OPT}$. Hence, $E_p(\mathcal{S}_0, J_i) = E_p(\mathcal{S}_{OPT}, J_i)$, for any $J_i \in \mathcal{J}_{YDS}$, and this fact replaces Lemma 3.6. For the proof of Theorem 5.1, we show the following lemma.

LEMMA 5.3. *For any $J_i \in \mathcal{J}_0$, it holds that $E_p(\mathcal{S}_0, J_i) + E_i(J_i) \le 2E_p(\mathcal{S}_{OPT}, J_i)$.*

PROOF. Let $J_i \in \mathcal{J}_0$ be an arbitrary job. As for the processing energy, $E_p(\mathcal{S}_0, J_i) = v_i P(s_{crit})/s_{crit} \le E_p(\mathcal{S}_{OPT}, J_i)$ because s_{crit} is the speed minimizing $P(s)/s$. Suppose that J_i is processed for T time units in $\mathcal{S}_{OPT}$. If J_i's speed is raised to s_{crit} in Step 1 of *Trans*, the extra idle energy incurred cannot be higher than $TP(0)$, which in turn is a lower bound on the processing energy incurred for J_i in $\mathcal{S}_{OPT}$. Hence, $E_i(J_i) \le E_p(\mathcal{S}_{OPT}, J_i)$, and the lemma follows. □

Using Lemmas 5.3 and 3.8, we can prove the desired inequality $E(\mathcal{S}_0) \le 2E(\mathcal{S}_{OPT})$. This concludes the proof of Theorem 5.1. For the proof of Theorem 5.2, we need another technical lemma.

LEMMA 5.4. *The inequality*

$$\frac{\lambda + \alpha - 1}{\lambda^\alpha + \alpha - 1} \le \frac{eW_{-1}(-e^{-1-1/e})}{eW_{-1}(-e^{-1-1/e}) + 1}$$

holds for all $\alpha > 1$ and $0 \le \lambda \le 1$. Furthermore, there exist $\alpha = \alpha' > 1$ and $\lambda = \lambda' \in [0, 1]$ such that equality holds.

PROOF. Assume that $(\lambda + \alpha - 1)/(\lambda^\alpha + \alpha - 1) \le x$. We will show that the smallest possible x satisfying this inequality is $eW(-e^{-1-1/e})/(eW(-e^{-1-1/e})+1)$. Inequality $(\lambda + \alpha - 1)/(\lambda^\alpha + \alpha - 1) \le x$ is equivalent to

$$x\lambda^\alpha + x(\alpha - 1) - \lambda - (\alpha - 1) \ge 0.$$

Setting $f(\lambda) = x\lambda^\alpha - \lambda + (x-1)(\alpha - 1)$, we have $f'(\lambda) = x\alpha\lambda^{\alpha-1} - 1$ and $f''(\lambda) \ge 0$. It follows that $f(\lambda)$ is minimized for $\lambda = (\frac{1}{x\alpha})^{\frac{1}{\alpha-1}}$, and it suffices to find the minimal x so that

$$x\left(\frac{1}{x\alpha}\right)^{\frac{\alpha}{\alpha-1}} - \left(\frac{1}{x\alpha}\right)^{\frac{1}{\alpha-1}} + (x-1)(\alpha-1) \ge 0$$

holds for every $\alpha > 1$. The latter inequality is equivalent to

$$\begin{aligned}
& \left(\frac{1}{\alpha} - 1\right)\left(\frac{1}{x\alpha}\right)^{\frac{1}{\alpha-1}} + (x-1)(\alpha-1) \geq 0 \\
\Leftrightarrow \quad & (\alpha - 1)\left(x - 1 - \frac{\left(\frac{1}{x\alpha}\right)^{\frac{1}{\alpha-1}}}{\alpha}\right) \geq 0 \\
\Leftrightarrow \quad & x - 1 - \frac{\left(\frac{1}{x\alpha}\right)^{\frac{1}{\alpha-1}}}{\alpha} \geq 0 \\
\Leftrightarrow \quad & x\alpha^\alpha (x-1)^{\alpha-1} \geq 1.
\end{aligned}$$

Substituting x by $\frac{c+1}{c}$, the last inequality becomes $(c+1)\alpha^\alpha \geq c^\alpha$, and we seek the largest possible c so that it is satisfied. We have

$$(c+1)\alpha^\alpha \geq c^\alpha \quad \Leftrightarrow \quad \ln(c+1) + \alpha(\ln\alpha - \ln c) \geq 0.$$

Let $g(\alpha) = \ln(c+1) + \alpha(\ln\alpha - \ln c)$. By derivating, we obtain $g'(\alpha) = \ln\alpha - \ln c + 1$ and $g''(\alpha) = 1/\alpha \geq 0$, which implies that g is minimized for $\alpha = c/e$. We, therefore, seek the largest possible c such that $\ln(c+1) \geq c/e$ holds.

For the largest possible c, equality holds, i.e., $\ln(c+1) = c/e$. By setting $c = -et - 1$ the equality becomes $te^t = -e^{-1-1/e}$. It follows that $t = W(-e^{-1-1/e})$, and $c = -et - 1 = -eW_{-1}(-e^{-1-1/e}) - 1$, which leads to

$$x = \frac{c+1}{c} = \frac{eW_{-1}(-e^{-1-1/e})}{eW_{-1}(-e^{-1-1/e}) + 1},$$

concluding the proof for the first statement of the lemma. Note that we are only interested in the lower branch of the W function, since $W_0(-e^{-1-\frac{1}{e}}) = W_0(-\frac{1}{e}e^{-\frac{1}{e}}) = -\frac{1}{e}$ resulting in $c = 0$.

For the second statement, it is sufficient to show that the extrema α' and λ', by which we substituted α and λ in the earlier analysis, are greater than 1 and in the range $[0, 1]$, respectively. For α, we have

$$\alpha' = \frac{c}{e} = \frac{-eW_{-1}(-e^{-1-1/e}) - 1}{e} > 1.$$

The last step holds because W_{-1} is a monotonically decreasing function, which implies that $W_{-1}(-e^{-1-\frac{1}{e}}) < W_{-1}(-e^{-1-\frac{1}{e}} - e^{-2-\frac{1}{e}}) = W_{-1}((-1-\frac{1}{e})e^{-1-\frac{1}{e}}) = -1 - \frac{1}{e}$. As for λ, we first prove that $\lambda' \leq 1$. It suffices to show that $x\alpha \geq 1$ and hence that $x \geq 1$. We thus have to show

$$\frac{eW_{-1}(-e^{-1-1/e})}{eW_{-1}(-e^{-1-1/e}) + 1} \geq 1. \tag{6}$$

The last inequality is satisfied: We already observed that $W_{-1}(-e^{-1-\frac{1}{e}}) < -1 - \frac{1}{e}$, which is less than $-1/e$, and (6) becomes $eW_{-1}(-e^{-1-1/e}) \leq eW_{-1}(-e^{-1-1/e}) + 1$. It remains to prove that $\lambda' \geq 0$, which is again equivalent to showing that $W_{-1}(-e^{-1-\frac{1}{e}}) \leq -\frac{1}{e}$. $\square$

LEMMA 5.5. *For any $J_i \in \mathcal{J}_0$, it holds that $E_p(\mathcal{S}_0, J_i) + E_i(J_i) \leq cE_p(\mathcal{S}_{OPT}, J_i)$, where $c = eW_{-1}(-e^{-1-1/e})/(eW_{-1}(-e^{-1-1/e}) + 1)$.*

PROOF. Let $J_i \in \mathcal{J}_0$ be an arbitrary job. If J_i is processed at speed s_{crit} in $\mathcal{S}_{OPT}$, there is nothing to show, so assume that J_i is processed at a speed smaller than s_{crit}. Let T and T' be the total times for which J_i is executed in $\mathcal{S}_{OPT}$ and $\mathcal{S}_0$, respectively. Choose λ, with

$0 \le \lambda \le 1$, such that $T' = \lambda T$. We have $s_{crit} = \sqrt[\alpha]{\gamma/(\beta(\alpha-1))}$ and $P(s_{crit}) = \gamma/(\alpha-1)+\gamma$. Hence, $E_p(\mathcal{S}_0, J_i) + E_i(J_i) \le \lambda T P(s_{crit}) + (1-\lambda)TP(0) = T(\lambda\gamma/(\alpha-1)+\gamma)$. In $\mathcal{S}_{OPT}$, job J_i is processed at speed λs_{crit}, and thus $E_p(\mathcal{S}_{OPT}, J_i) = TP(\lambda s_{crit}) = T(\lambda^\alpha\gamma/(\alpha-1)+\gamma)$. By Lemma 5.4, as desired,

$$\frac{E_p(\mathcal{S}_0, J_i) + E_i(J_i)}{E_p(\mathcal{S}_{OPT}, J_i)} \le \frac{\lambda\gamma/(\alpha-1)+\gamma}{\lambda^\alpha\gamma/(\alpha-1)+\gamma} = \frac{\lambda+\alpha-1}{\lambda^\alpha+\alpha-1} \le \frac{eW_{-1}(-e^{-1-1/e})}{eW_{-1}(-e^{-1-1/e})+1}. \quad \square$$

Using Lemmas 5.5 and 3.8, we can show $E(\mathcal{S}_0) \le cE(\mathcal{S}_{OPT})$, where $c = eW_{-1}(-e^{-1-1/e})/(eW_{-1}(-e^{-1-1/e})+1)$. Note that, here, $\mathcal{J}_{0,2} = \emptyset$ and that $E_p(\mathcal{S}_0, J_i) = E_p(\mathcal{S}_{OPT}, J_i) \le c \cdot E_p(\mathcal{S}_{OPT}, J_i)$, for $J_i \in \mathcal{J}_{YDS}$. This establishes Theorem 5.2. For power functions $P(s) = \beta s^\alpha + \gamma$, we prove a matching lower bound on the performance of s_{crit}-schedules.

THEOREM 5.6. *Let A be an algorithm that computes s_{crit}-schedules for any job instance. Then A does not achieve an approximation factor smaller than* $eW_{-1}(-e^{-1-\frac{1}{e}})/(eW_{-1}(-e^{-1-\frac{1}{e}})+1)$, *for power functions $P(s) = \beta s^\alpha + \gamma$, where $\alpha > 1$ and $\beta, \gamma > 0$.*

PROOF. Let ϵ, $0 < \epsilon < 1$, be a constant. We show that A cannot achieve an approximation factor smaller than

$$\frac{eW_{-1}(-e^{-1-\frac{1}{e}})}{eW_{-1}(-e^{-1-\frac{1}{e}})+1} - \epsilon.$$

Fix a power function $P(s) = \beta s^{\alpha'} + \gamma$, where α' is defined as in Lemma 5.4. Then $s_{crit} = \sqrt[\alpha']{\gamma/(\beta(\alpha'-1))}$. We specify a job sequence that is similar to the one in the proof of Theorem 2.3. Let again $L > 0$ be an arbitrary constant. We define three jobs J_1, J_2, and J_3: Jobs J_1 and J_3 both have a processing volume of $v_1 = v_3 = \delta L s_{crit}$ and can be executed in intervals $I_1 = [0, \delta L)$ and $I_3 = [(1+\delta)L, (1+2\delta)L)$, respectively. Job J_2 has a processing volume of $v_2 = \lambda' L s_{crit}$, where λ' is as defined in Lemma 5.4, and can be executed in $I_2 = [\delta L, (1+\delta)L)$. The energy consumed by a wake-up operation is $C = LP(0) = \gamma L$.

As in the proof of Theorem 2.3, we first assume that the processor is in the active state at time 0. We analyze the energy of algorithm A and an optimal solution. We assume that A processes all the three jobs at speed s_{crit}. If a job is processed at a higher speed, we can reduce its speed to s_{crit}. This speed reduction reduces the processing energy of the job and does not increase the idle energy of the schedule. Hence, jobs J_1 and J_3 consume an energy of $\delta L P(s_{crit})$ each. Job J_2 is processed for $v_2/s_{crit} = \lambda' L$ time units in I_2, resulting in an energy consumption of $\lambda' L P(s_{crit}) = \lambda' L(\gamma/(\alpha'-1)+\gamma)$. During the $(1-\lambda')L$ time units remaining in I_2 the processor is idle. Since $C > (1-\lambda')LP(0)$, the processor should stay in the active state for this amount of time. It follows that the energy consumption of A is at least

$$2\delta L P(s_{crit}) + \lambda' L P(s_{crit}) + (1-\lambda')LP(0) > L\left(\frac{\lambda'\gamma}{\alpha'-1}+\gamma\right).$$

An optimal solution will also process J_1 and J_3 at speed s_{crit}. However, J_2 can be executed during the whole interval I_2 at speed $\lambda' s_{crit}$, yielding an energy consumption of $LP(\lambda' s_{crit})$. It follows that the energy consumption of an optimal solution is upper bounded by

$$2\delta L P(s_{crit}) + LP(\lambda' s_{crit}) = L\left(2\delta P(s_{crit}) + (\lambda')^{\alpha'}\frac{\gamma}{\alpha'-1}+\gamma\right).$$

Now assume that the processor is in the sleep state at time 0. We repeat the previous job sequence k times, where the value of k will be determined later. For each repetition i, $1 \leq i \leq k$, three jobs are introduced. The job J_{i1}, J_{i2}, and J_{i3} have processing volumes of $v_{ij} = v_j$, for $1 \leq j \leq 3$, and respective execution intervals $[t_i, t_i + \delta L)$,$[t_i + \delta L, t_i + (1+\delta)L)$ and $[t_i + (1+\delta)L, t_i + (1+2\delta)L)$, where $t_i = (i-1)(1+2\delta)L$. For this job sequence, the ratio of the energy consumed by A to that of an optimal solution is at least

$$\frac{kL\left(\lambda' \frac{\gamma}{\alpha'-1} + \gamma\right)}{C + kL\left(2\delta P(s_{crit}) + (\lambda')^{\alpha'} \frac{\gamma}{\alpha'-1} + \gamma\right)}.$$

Set $X = \frac{\lambda'}{\alpha'-1} + 1$ and $Y = \frac{(\lambda')^{\alpha'}}{\alpha'-1} + 1$. Moreover, set $\epsilon' = \epsilon Y^2/X$. With these settings, let $\delta = \epsilon'\gamma/(4P(s_{crit}))$ and $k = \lceil 2/\epsilon' \rceil$. Then $2\delta P(s_{crit}) \leq \epsilon'\gamma/2$ and $C = kC/k \leq kC\epsilon'/2 = kL\gamma\epsilon'/2$. Hence, the aforementioned ratio is at least

$$\frac{\frac{\lambda'}{\alpha'-1} + 1}{\frac{(\lambda')^{\alpha'}}{\alpha'-1} + 1 + \epsilon'} \geq \frac{\frac{\lambda'}{\alpha'-1} + 1}{\frac{(\lambda')^{\alpha'}}{\alpha'-1} + 1} - \epsilon = \frac{\lambda' + \alpha' - 1}{(\lambda')^{\alpha'} + \alpha' - 1} - \epsilon.$$

The first inequality holds because $X/(Y + \epsilon') \geq X/Y - \epsilon$ by our choices of X, Y, and ϵ'. The theorem now follows from Lemma 5.4. □

6. CONCLUSIONS

Speed scaling with sleep state is a timely energy conservation problem as the static energy consumed by modern microprocessors in the active state is comparable to the dynamic energy needed for processing. In this article, we have developed offline algorithms achieving small approximation guarantees. All the algorithms use only one speed level, in addition to those of *YDS*. This is a positive feature because speed adjustments incur overhead in practice. On the other hand, the use of several, that is, a constant number of, speeds below s_{crit} will most likely lead to a PTAS. The development of such an approximation scheme is the major open problem for speed scaling with sleep state. At this point, it is not clear how to construct good schedules using several low speeds.

APPENDIX

PROOF OF LEMMA 2.2 ($\Leftarrow$) Assume that A of $\mathcal{I}_P$ admits a partition. We show how to construct a feasible schedule for $\mathcal{I}_S$ with an energy consumption of exactly $5(n+1)\epsilon a_{max} + nC + \frac{1}{2}\sum_{i=1}^{n} l_i + \frac{B}{2a_{max}}$. Let A' be the respective subset in the solution of $\mathcal{I}_P$, and assume that $|A'| = m$. Schedule each job of $\mathcal{J}_2$ at a speed of s_{crit}. This fills the respective execution interval of the job. The total energy consumed by all the jobs of $\mathcal{J}_2$ is $(n+1)\epsilon P(s_{crit}) = 5(n+1)\epsilon a_{max}$. Next, in the gaps g_i such that $a_i \in A'$, execute J_0 and the respective jobs of $\mathcal{J}_1$. We will show that this can be done in a balanced way so that all the total processing volume gets executed at a constant speed of a_{max}. We first observe that any job of $\mathcal{J}_1$ alone has a density less than a_{max} in its execution interval. It therefore remains to show that the total density of the jobs $J_i \in \mathcal{J}_1$, with $a_i \in A'$, and J_0, restricted to the gaps g_i with $a_i \in A'$, is a_{max}. This density is

$$\frac{\sum_{i:a_i \in A'} l_i + B}{\sum_{i:a_i \in A'} L_i} = \frac{a_{max} \sum_{i:a_i \in A'} L_i - B + B}{\sum_{i:a_i \in A'} L_i} = a_{max},$$

as claimed. Finally, we run the jobs $J_i \in \mathcal{J}_1$, with $a_i \notin A'$, at a speed of $s_{crit} = 10a_{max}$ starting directly at their release time. We then transition the processor to the sleep state for the rest of the respective gap. This is feasible because $(L_i a_{max} - a_i)/(10a_{max}) < L_i$. Therefore, the energy expended for J_0, the jobs in $\mathcal{J}_1$ and the wake-up operations is equal to

$$\begin{aligned} P(a_{max}) \cdot \sum_{i:a_i \in A'} L_i + P(s_{crit}) \cdot \sum_{i:a_i \notin A'} \frac{l_i}{s_{crit}} + (n-m)C \\ = a_{max} \sum_{i:a_i \in A'} L_i + \frac{1}{2} \sum_{i:a_i \notin A'} l_i + (n-m)C. \end{aligned}$$

It, therefore, suffices to show that

$$mC + \frac{1}{2} \sum_{i:a_i \in A'} l_i + \frac{B}{2a_{max}} = a_{max} \sum_{i:a_i \in A'} L_i,$$

which is equivalent to

$$m a_{max} + \frac{B}{2a_{max}} - \frac{B}{2} = \frac{1}{2} a_{max} \sum_{i:a_i \in A'} L_i.$$

The latter equation holds true because $a_{max} \sum_{i:a_i \in A'} L_i = 2a_{max}m - B + B/a_{max}$.

($\Rightarrow$) Assume now that no solution to $\mathcal{I}_P$ exists. That is, for all subsets $A' \subseteq A$, it holds that $\sum_{a_i \in A'} a_i \neq \sum_{a_i \in A \setminus A'} a_i$. We will show that an optimal schedule for $\mathcal{I}_S$ consumes energy strictly greater than $5(n+1)\epsilon a_{max} + nC + \frac{1}{2}\sum_{i=1}^n l_i + \frac{B}{2a_{max}}$. We first argue that there exists an optimal schedule that executes the jobs of $\mathcal{J}_2$ during their whole execution intervals at a speed of s_{crit}. So let $\mathcal{S}$ be any optimal schedule. If no portion of J_0 is processed during the execution intervals of jobs of $\mathcal{J}_2$, there is nothing to show. If a portion of J_0 is executed in such an interval I, then we can modify $\mathcal{S}$ without increasing the total energy consumption: In I, an average speed higher than s_{crit} must be used. In the schedule there must exist a gap g_i in which (a) the processor transitions to the sleep state or (b) an average speed less than s_{crit} is used. The latter property (b) holds because the average speed required to execute the jobs of $\mathcal{J}_2$ and J_0 in the gaps g_i, $1 \leq i \leq n$, is smaller than $a_{\max} < s_{crit}$.

In case (a), we execute a portion of J_0 originally scheduled in I at speed s_{crit} in g_i. This can be done immediately before the processor transitions to the sleep state. By the convexity of $P(s)$, the total energy does not increase. In case (b), we process a portion of J_0 in g_i by slightly raising the processor speed of $s < s_{crit}$ up to a value of at most s_{crit}. Again, the energy consumption of the schedule does not increase, due to the convexity of the power function. These schedule modifications can be repeated until the jobs of $\mathcal{J}_2$ are processed exclusively in their execution intervals.

In the following, let $\mathcal{S}$ be an optimal schedule in which the jobs of $\mathcal{J}_2$ are executed at speed s_{crit} in their execution intervals, incurring an energy of $5(n+1)\epsilon a_{max}$. It remains to show that the energy consumed by the wake-up operations, the processing of J_0 and of the jobs in $\mathcal{J}_1$ is strictly greater than $nC + \frac{1}{2}\sum_{i=1}^n l_i + \frac{B}{2a_{max}}$.

Assume that $\mathcal{S}$ executes b_i units of J_0's processing volume in gap g_i, $1 \leq i \leq n$. It holds that $\sum_{i=1}^n b_i = B$. For each gap, there is a lower bound threshold on the processing volume required so that it is worthwhile not to transition the processor to the sleep state in between. We next argue that, for gap g_i, this threshold is $l_i + a_i/a_{max}$. First, consider a load of exactly $l_i + a_i/a_{max}$. The energy consumed in g_i if jobs are processed at speed s_{crit} and a transition to the sleep state is made equals

$$C + \frac{1}{2}\left(l_i + \frac{a_i}{a_{max}}\right) = \frac{1}{2}2a_{max} + \frac{1}{2}a_{max}L_i - \frac{1}{2}a_i + \frac{1}{2}\frac{a_i}{a_{max}} = a_{max}L_i.$$

If no transition to the sleep state is made, by the convexity of $P(s)$, it is optimal to execute the respective processing volume at uniform speed. Hence, the energy consumption is

$$L_i \cdot P\left(\frac{l_i + a_i/a_{max}}{L_i}\right) = L_i \cdot P\left(a_{max} - \frac{a_i - a_i/a_{max}}{L_i}\right) = a_{max}L_i,$$

which is the same value. Next, consider a processing volume of $l_i + a_i/a_{max} - \delta$, for any $\delta > 0$. If a transition to the sleep state is made, the consumed energy is $a_{max}L_i - \delta/2$. If the processor does not transition to the sleep state, the incurred energy is $a_{max}L_i$, which is greater than the former expression. In other words, for processing volume strictly less than $l_i + a_i/a_{max}$, it is preferable to transition to the sleep state. Finally, consider a processing volume of $l_i + a_i/a_{max} + \delta$, for any $\delta > 0$. If the processor transitions to the sleep state, the expended energy is $a_{max}L_i + \delta/2$. If a transition to the sleep state is made, it remains to distinguish two cases. If $a_i - a_i/a_{max} - \delta \geq 0$, then the incurred energy is $a_{max}L_i$. Otherwise, the energy is $a_{max}L_i + (4/9)(\delta - a_i + a_i/a_{max})$. In both cases, the incurred energy is smaller than $a_{max}L_i + \delta/2$. This implies that when the processing volume is strictly greater than $l_i + a_i/a_{max}$, then it is advantageous to remain in the active state.

Let $A' \subseteq A$ contain the a_i's such that $b_i \geq a_i/a_{max}$, and let again $|A'| = m$. We assume that the processing volume handled in the g_i's, with $a_i \in A'$, is executed at a uniform speed equal to

$$\frac{\sum_{i:a_i \in A'}(l_i + b_i)}{\sum_{i:a_i \in A'} L_i}.$$

This might not be feasible, but again, due to the convexity of the power function, the resulting energy consumption is in no case higher than the energy consumption of the original schedule $\mathcal{S}$. Hence, in the gaps g_i with $a_i \in A'$, the energy consumption is at least

$$\sum_{i:a_i \in A'} L_i \cdot P\left(\frac{\sum_{i:a_i \in A'}(l_i + b_i)}{\sum_{i:a_i \in A'} L_i}\right) = \sum_{i:a_i \in A'} L_i \cdot P\left(a_{max} + \frac{\sum_{i:a_i \in A'} b_i - \sum_{a_i \in A'} a_i}{\sum_{i:a_i \in A'} L_i}\right). \quad (7)$$

In the gaps g_i with $a_i \notin A'$, the processor executes jobs at speed s_{crit} and transitions to the sleep state. In these gaps, the total energy consumption is

$$(n - m)C + \frac{1}{2}\sum_{i:a_i \notin A'}(l_i + b_i). \quad (8)$$

We have to prove that the total energy consumption of (7) and (8) is strictly greater than $nC + \frac{1}{2}\sum_{i=1}^{n} l_i + \frac{B}{2a_{max}}$. Thus, we have to show that

$$\frac{1}{2}\sum_{i=1}^{n} l_i + \frac{B}{2a_{max}}$$
$$< -mC + \frac{1}{2}\sum_{i:a_i \notin A'} l_i + \frac{1}{2}\sum_{i:a_i \notin A'} b_i + \sum_{i:a_i \in A'} L_i \cdot P\left(a_{max} + \frac{\sum_{i:a_i \in A'} b_i - \sum_{a_i \in A'} a_i}{\sum_{i:a_i \in A'} L_i}\right),$$

which is equivalent to

$$mC + \frac{1}{2}\sum_{i:a_i\in A'} l_i + \frac{B}{2a_{max}} < \frac{1}{2}\sum_{i:a_i\notin A'} b_i + \sum_{i:a_i\in A'} L_i \cdot P\left(a_{max} + \frac{\sum_{i:a_i\in A'} b_i - \sum_{a_i\in A'} a_i}{\sum_{i:a_i\in A'} L_i}\right).$$

We consider two distinct cases.

Case (1): Suppose that $\sum_{i:a_i\in A'} b_i \le \sum_{a_i\in A'} a_i$.

Since in this case the argument of P in the aforementioned inequality is at most $a_{\max}$, we have to show

$$ma_{max} + \frac{1}{2}\sum_{i:a_i\in A'} l_i + \frac{B}{2a_{max}} < \frac{1}{2}\sum_{i:a_i\notin A'} b_i + a_{max}\sum_{i:a_i\in A'} L_i.$$

Substituting l_i, we get

$$ma_{max} - \frac{1}{2}\sum_{a_i\in A'} a_i + \frac{B}{2a_{max}} < \frac{1}{2}\sum_{i:a_i\notin A'} b_i + \frac{1}{2}a_{max}\sum_{i:a_i\in A'} L_i.$$

We then substitute L_i and have

$$-\frac{1}{2}\sum_{a_i\in A'} a_i + \frac{B}{2a_{max}} < \frac{1}{2}\sum_{i:a_i\notin A'} b_i - \frac{1}{2}\frac{a_{max}-1}{a_{max}}\sum_{a_i\in A'} a_i,$$

which is equivalent to

$$\frac{B}{2a_{max}} < \frac{1}{2}\sum_{i:a_i\notin A'} b_i + \frac{1}{2a_{max}}\sum_{a_i\in A'} a_i.$$

If $\sum_{i:a_i\notin A'} b_i = 0$, then $\sum_{i:a_i\in A'} b_i = B$. Since by our assumption $\sum_{a_i\in A'} a_i \ne B$, it must be the case that $B = \sum_{i:a_i\in A'} b_i < \sum_{a_i\in A'} a_i$ and the inequality follows.

If, on the other hand, $\sum_{i:a_i\notin A'} b_i = X > 0$, we have $\sum_{i:a_i\in A'} b_i = B - X \le \sum_{a_i\in A'} a_i$, and we wish to show

$$\frac{B}{2a_{max}} < \frac{1}{2}X + \frac{B}{2a_{max}} - \frac{X}{2a_{max}}.$$

This holds for any $X > 0$ and $a_{max} \ge 2$.

Case (2): Suppose that $\sum_{i:a_i\in A'} b_i > \sum_{a_i\in A'} a_i$.

Let $\sum_{i:a_i\notin A'} b_i = X \ge 0$. It follows that $\sum_{i:a_i\in A'} b_i = B - X > \sum_{a_i\in A'} a_i$. We wish to show that

$$\begin{aligned} mC &+ \frac{1}{2}a_{max}\sum_{i:a_i\in A'} L_i - \frac{1}{2}\sum_{a_i\in A'} a_i + \frac{B}{2a_{max}} \\ &< \frac{1}{2}X + \sum_{i:a_i\in A'} L_i \cdot P\left(a_{max} + \frac{B - \sum_{a_i\in A'} a_i}{\sum_{i:a_i\in A'} L_i} - \frac{X}{\sum_{i:a_i\in A'} L_i}\right). \end{aligned}$$

Since $(4/9)s + (5/9)a_{max} \le 2s - 15a_{max}$ for any $s \ge 10a_{max}$, and the argument of P in the aforementioned inequality is strictly greater than a_{max}, we may use the middle branch of the power function. The inequality then becomes

$$mC + \frac{1}{2}a_{max}\sum_{i:a_i\in A'} L_i - \frac{1}{2}\sum_{a_i\in A'} a_i + \frac{B}{2a_{max}} < \frac{1}{2}X + a_{max}\sum_{i:a_i\in A'} L_i + \frac{4}{9}\left(B - \sum_{a_i\in A'} a_i\right) - \frac{4}{9}X,$$

which is equivalent to

$$mC - \frac{1}{2}\sum_{a_i \in A'} a_i + \frac{B}{2a_{max}} < \frac{1}{18}X + \frac{1}{2}a_{max}\sum_{i:a_i \in A'} L_i + \frac{4}{9}\left(B - \sum_{a_i \in A'} a_i\right).$$

By substituting L_i, we get

$$-\frac{1}{2}\sum_{a_i \in A'} a_i + \frac{B}{2a_{max}} < \frac{1}{18}X - \frac{1}{2}\sum_{a_i \in A'} a_i + \frac{\sum_{a_i \in A'} a_i}{2a_{max}} + \frac{4}{9}\left(B - \sum_{a_i \in A'} a_i\right),$$

or equivalently,

$$\frac{B}{2a_{max}} < \frac{1}{18}X + \frac{\sum_{a_i \in A'} a_i}{2a_{max}} + \frac{4}{9}\left(B - \sum_{a_i \in A'} a_i\right).$$

It suffices to show that $B/(2a_{max}) < (\sum_{a_i \in A'} a_i)/(2a_{max}) + (4/9)(B - \sum_{a_i \in A'} a_i)$. This is equivalent to $(B - \sum_{a_i \in A'} a_i)/(2a_{max}) < (4/9)(B - \sum_{a_i \in A'} a_i)$, and the latter inequality holds for $a_{max} \geq 2$. The proof is complete. $\square$

REFERENCES

Lachlan L. H. Andrew, Adam Wierman, and Ao Tang. 2009. Optimal speed scaling under arbitrary power functions. *SIGMETRICS Performance Evaluation Review* 37, 2 (2009), 39–41.

Peter Bailis, Vijay Janapa Reddi, Sanjay Gandhi, David Brooks, and Margo I. Seltzer. 2011. Dimetrodon: Processor-Level Preventive Thermal Management via Idle Cycle Injection. In *DAC*, Leon Stok, Nikil D. Dutt, and Soha Hassoun (Eds.). ACM, 89–94.

Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. 2011. Average Rate Speed Scaling. *Algorithmica* 60, 4 (2011), 877–889.

Nikhil Bansal, Ho-Leung Chan, Dmitriy Katz, and Kirk Pruhs. 2012. Improved bounds for speed scaling in devices obeying the cube-root rule. *Theory of Computing* 8, 1 (2012), 209–229.

Nikhil Bansal, Ho-Leung Chan, Tak Wah Lam, and Lap-Kei Lee. 2008. Scheduling for speed bounded processors. In *ICALP*. Springer, 409–420.

Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. 2013. Speed Scaling with an Arbitrary Power Function. *ACM Transactions on Algorithms* 9, 2 (2013), 18.

Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. 2007. Speed scaling to manage energy and temperature. *J. ACM* 54, 1 (2007).

Philippe Baptiste. 2006. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *SODA*. ACM, 364–367.

Philippe Baptiste, Marek Chrobak, and Christoph Dürr. 2012. Polynomial-time algorithms for minimum energy scheduling. *ACM Transactions on Algorithms* 8, 3 (2012), 26.

Ho-Leung Chan, Joseph Wun-Tat Chan, Tak Wah Lam, Lap-Kei Lee, Kin-Sum Mak, and Prudence W. H. Wong. 2009. Optimizing throughput and energy in online deadline scheduling. *ACM Transactions on Algorithms* 6, 1 (2009).

Sze-Hang Chan, Tak Wah Lam, and Lap-Kei Lee. 2013. Scheduling for weighted flow time and energy with rejection penalty. *Theor. Comput. Sci.* 470 (2013), 93–104.

Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, and Charles Lefurgy. 2009. Optimal power allocation in server farms. In *SIGMETRICS/Performance*, John R. Douceur, Albert G. Greenberg, Thomas Bonald, and Jason Nieh (Eds.). ACM, 157–168.

Matthew Garrett. 2007. Powering down. *ACM Queue* 5, 7 (2007), 16–21.

Xin Han, Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. 2010. Deadline scheduling and power management for speed bounded processors. *Theor. Comput. Sci.* 411, 40–42 (2010), 3587–3600.

Sandy Irani and Kirk Pruhs. 2005. Algorithmic problems in power management. *SIGACT News* 36, 2 (2005), 63–76.

Sandy Irani, Sandeep K. Shukla, and Rajesh Gupta. 2007. Algorithms for power savings. *ACM Transactions on Algorithms* 3, 4 (2007).

Florian Kluge, Sascha Uhrig, Jörg Mische, Benjamin Satzger, and Theo Ungerer. 2010. Optimisation of energy consumption of soft real-time applications by workload prediction. In *Proc. 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*. 63–72.

Minming Li, Becky Jie Liu, and Frances F. Yao. 2006. Min-energy voltage allocation for tree-structured tasks. *J. Comb. Optim.* 11, 3 (2006), 305–319.

Minming Li and F. Frances Yao. 2005. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM J. Comput.* 35, 3 (2005), 658–671.

F. Frances Yao, Alan J. Demers, and Scott Shenker. 1995. A scheduling model for reduced CPU energy. In *FOCS*. IEEE Computer Society, 374–382.

Received February 2007; revised March 2009; accepted June 2009