Theoretical Computer Science 412 (2011) 4074–4080

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Approximation algorithms for variable voltage processors: Min energy, max throughput and online heuristics☆

Minming Li*

Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong

ARTICLE INFO

Keywords:
Online heuristics
Approximation algorithms
Dynamic voltage scaling
Throughput
Minimum energy

ABSTRACT

Dynamic Voltage Scaling techniques allow the processor to set its speed dynamically in order to reduce energy consumption. It was shown that if the processor can run at arbitrary speeds and uses power s^α when running at speed s, the online heuristic AVR has a competitive ratio $(2\alpha)^\alpha/2$. In this paper we first study the online heuristics for the discrete model where the processor can only run at d given speeds. We propose a method to transform online heuristic AVR to an online heuristic for the discrete model and prove a competitive ratio $\frac{2^{\alpha-1}(\alpha-1)^{\alpha-1}(\delta^\alpha-1)^\alpha}{(\delta-1)(\delta^\alpha-\delta)^{\alpha-1}}+1$, where δ is the maximum ratio between adjacent non-zero speed levels. We also prove that the analysis holds for a class of heuristics that satisfy certain natural properties. We further study the throughput maximization problem when there is an upper bound for the maximum speed. We propose a greedy algorithm with running time $O(n^2 \log n)$ and prove that the output schedule is a 3-approximation of the throughput and a $\frac{(\alpha-1)^{\alpha-1}(3^\alpha-1)^\alpha}{2\alpha^\alpha(3^{\alpha-1}-1)^{\alpha-1}}$-approximation of the energy consumption.

1. Introduction

Energy efficiency currently becomes one of the major concerns in system designs. Portable electronic devices, which are typically powered by batteries, rely on energy efficient schedules to increase the battery lifetime; while non-portable systems need energy efficient schedules to reduce the operating cost. An important strategy to achieve energy saving is via *dynamic voltage scaling (DVS)*, which enables a processor to operate at a range of voltages and frequencies. Because energy consumption is at least a quadratic function of the voltage (hence CPU frequency/speed), it saves energy by executing jobs as slowly as possible while still satisfying the required property like feasibility where all the jobs are required to be finished by their deadlines. The associated scheduling problem is referred to as min-energy feasibility DVS scheduling.

A theoretical study of min-energy feasibility DVS scheduling was initiated by Yao et al. [1]. They formulated the optimization problem and gave an $O(n^3)$ algorithm YDS for computing the optimal schedule, which was later improved to $O(n^2 \log n)$ in [2]. In their formulation, each job j_i has an arrival time a_i, a workload R_i, and a deadline b_i. If job j_i is executed at speed s, then it needs R_i/s time to finish. They also assumed a speed to power function $P(s) = s^\alpha$, where $\alpha \geq 2$ is a system constant (usually $\alpha = 3$ if the cubic-root rule holds). Their model allows the processor speed to be set at any real value and therefore is referred to as the *continuous* model. An online heuristic AVR was proposed in [1] and proved to be $(2\alpha)^\alpha/2$-competitive in energy consumption. Bansal et al. [3] further investigated the online heuristics for the model proposed in [1] and proved that the heuristic OA has a tight competitive ratio of α^α for all job sets. They also gave a new online heuristic which is the best possible heuristic for a wide range of power functions. Quan and Hu [4] considered scheduling jobs with

☆ This work was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 117408].
* Tel.: +852 27889538; fax: +852 27888614.
E-mail address: minmli@cs.cityu.edu.hk.

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.10.011

fixed priorities and characterized the optimal schedule. Yun and Kim [5] later on showed the NP-hardness of computing the optimal schedule and also provided an FPTAS for this problem.

In practice, processors can run at only a finite number of preset speed levels. One can capture the discrete nature of the speed scale with a corresponding *discrete* model. Ishihara and Yasuura [6] initiated the research on discrete DVS problems and solved the case when the processor is only allowed to run at two different speeds. Kwon and Kim [7] extended that to the general discrete DVS model where the processor is allowed to run at speeds chosen from a finite speed set. They gave an algorithm for this problem based on the YDS algorithm in [1]. Recently, It was shown in [8] that the optimal schedule for the discrete model can be computed in $O(dn \log n)$ time where d is the number of speed levels. Compared to numerous analyses for online heuristics proposed for the continuous model, few works exist that design or analyze online heuristics for the discrete model which is more practical. We propose in this paper a systematic way to transform online heuristics for the continuous model to online heuristics for the discrete model with only a small increase in the competitive ratio.

Another practical constraint on the processor is the maximum speed limit, which sometimes makes it impossible for the processor to finish all jobs within their timing constraints. In this bounded speed model, a useful objective is to maximize the throughput of the whole system. In real time scheduling where the processor cannot change the execution speed, the throughput maximization problem where preemptions are not allowed is studied in [9–11]. [12,13] considered preemptive scheduling. It was shown in [12] that the best possible online scheduling algorithm is 4-competitive on throughput without considering energy and [13] proposed an online heuristic which is 14-competitive in throughput and $O(1)$-competitive in energy consumption. We also refer the readers to a recent survey on research in power/temperature management [14].

In this paper, we assume that preemption is allowed in job execution, i.e., a job can be interrupted when being executed and resumed afterwards. We first investigate online heuristics for the discrete model where the allowed speed set $\{s_1, s_2, \ldots, s_d\}$ satisfies $s_1 > s_2 > \cdots > s_d > 0$. By suitably adjusting the speed used in AVR to adjacent allowed speed levels, we design an online heuristic AVR'_d for the discrete model and prove it to be $(\frac{2^{\alpha-1}(\alpha-1)^{\alpha-1}(\delta^\alpha-1)^\alpha}{(\delta-1)(\delta^\alpha-\delta)^{\alpha-1}} + 1)$-competitive where $\delta = \max_{1\le i\le d-1} \frac{s_i}{s_{i+1}}$. We also identify a class of heuristics which can be modified in a similar way from the continuous model to the discrete model with their competitive ratios increased by a factor of $\frac{(\alpha-1)^{\alpha-1}(\delta^\alpha-1)^\alpha}{\alpha^\alpha(\delta-1)(\delta^\alpha-\delta)^{\alpha-1}}$. A similar idea of adjusting speed to adjacent speed levels is widely known and proved to increase the competitive ratio by δ^α for the throughput maximization algorithm proposed in [13]. Notice that our increased ratio is $\frac{(\delta+1)^2}{4\delta}$ when $\alpha = 2$ which improves upon δ^α almost by a factor of 4δ. Then we study the throughput maximization problem for the continuous model in the overloaded setting where the processor can only vary its speed between 0 and a maximum speed s_{max}. With this practical speed restriction, the processor sometimes cannot finish all the jobs by their deadlines even if it always runs at s_{max}. We aim to find schedules which can minimize energy consumption while maximizing the throughput. The throughput is the total workload of the jobs completed by their deadlines. It is NP-hard to find schedules that maximize throughput since KNAPSACK is a special case when all deadlines are equal and all arrival times are equal. We propose a greedy algorithm to approximately maximize the throughput while also approximately minimizing the energy consumption. By using the properties of s-schedules defined in [8] and properties proved in the study of online heuristics for the discrete model, we prove that the greedy algorithm outputs a schedule which is a 3-approximation of throughput and a $\frac{(\alpha-1)^{\alpha-1}(3^\alpha-1)^\alpha}{2\alpha^\alpha(3^{\alpha-1}-1)^{\alpha-1}}$-approximation of energy consumption.

The remainder of the paper is organized as follows. We give the problem formulation and review some basic properties of the optimal schedules in Section 2. Section 3 describes a $(\frac{2^{\alpha-1}(\alpha-1)^{\alpha-1}(\delta^\alpha-1)^\alpha}{(\delta-1)(\delta^\alpha-\delta)^{\alpha-1}} + 1)$-competitive online heuristic for the discrete model and extends the result to a class of online heuristics. We then study the offline throughput maximization problem in Section 4 and propose a greedy algorithm to compute a schedule that is 3-approximation in throughput and a $\frac{(\alpha-1)^{\alpha-1}(3^\alpha-1)^\alpha}{2\alpha^\alpha(3^{\alpha-1}-1)^{\alpha-1}}$-approximation in energy consumption. Some concluding remarks are given in Section 5.

2. Models and preliminaries

A job set $J = \{j_1, j_2, \ldots, j_n\}$ over $[0, 1]$ is given where each job j_k is characterized by three parameters: arrival time a_k, deadline b_k, and workload R_k. Here workload means the required number of CPU cycles. We also refer to $[a_k, b_k] \subseteq [0, 1]$ as the interval of j_k, and assume without loss of generality that $\cup_k[a_k, b_k] = [0, 1]$ (or J *spans* $[0, 1]$). A *schedule* S for J is a pair of functions $(s(t), job(t))$ which defines the processor speed and the job being executed at time t respectively. Both functions are assumed to be piecewise continuous with finitely many discontinuities. A *feasible* schedule must give each job its required workload between its arrival time and deadline with perhaps intermittent execution. We assume that the power P, or energy consumed per unit time, is $P(s) = s^\alpha$ $(\alpha \ge 2)$ where s is the processor speed. The total energy consumed by a schedule S is $E(S) = \int_0^1 P(s(t))\mathrm{d}t$. The goal of the min-energy feasibility scheduling problem is to find a feasible schedule that minimizes $E(S)$ for any given job set J. We refer to this problem as the continuous *DVS* scheduling problem.

In the discrete version of the problem, we assume that the processor can run at d speed levels $s_1 > s_2 > \cdots > s_d$. The goal is to find a minimum-energy schedule for a job set using only these speeds. We refer to this problem as the *Discrete DVS* scheduling problem if the highest speed s_1 is always fast enough to guarantee a feasible schedule for the given jobs.

For the continuous DVS scheduling problem, the optimal schedule S_{opt} is characterized by using the notion of a *critical interval* for J, which is an interval I in which a group of jobs must be scheduled at maximum constant speed $g(I)$ in any

optimal schedule for J. The algorithm proceeds by identifying such a critical interval I, scheduling those 'critical' jobs at speed $g(I)$ over I, then constructing a subproblem for the remaining jobs and solving it recursively. The details are given below.

Definition 1. For any interval $I \subseteq [0, 1]$, we use J_I to denote the subset of jobs in J whose intervals are completely contained in I. The intensity of an interval I is defined to be $g(I) = (\sum_{j_k \in J_I} R_k)/|I|$.

An interval I^* achieving maximum $g(I)$ over all possible intervals I defines a critical interval for the current job set. It is known that the subset of jobs J_{I^*} can be feasibly scheduled at speed $g(I^*)$ over I^* by the earliest deadline first (EDF) principle. That is, at any time t, a job which is waiting to be executed and having the earliest deadline will be executed during $[t, t+\epsilon]$. The interval I^* is then removed from $[0, 1]$; all the remaining job intervals $[a_k, b_k]$ are updated to reflect the removal, and the algorithm recurses. We denote the optimal schedule which guarantees feasibility and consumes minimum energy in the continuous model as *OPT*.

3. Online heuristics for discrete model

In [1], two on-line heuristics AVR (Average Rate) and OA (Optimal Available) were introduced for the case that jobs arrive one after another. We define these two heuristics as follows.

At any time t, the heuristic AVR runs the earliest deadline job using the speed $\sum_{J(t)} \frac{R_k}{b_k - a_k}$ where $J(t)$ is the set of jobs with $a_k \leq t \leq b_k$. In other words, AVR always uses a speed equal to the summation of the average workload of all the available jobs.

At any time t, the heuristic OA always assumes that the current available jobs are the only jobs to be scheduled. It calculates an optimal offline schedule for all these jobs and schedules them accordingly. Whenever a new job arrives, it will do a recalculation.

The heuristic AVR was shown to have a competitive ratio of at most $(2\alpha)^\alpha/2$ in [1]; recently a tight competitive ratio of α^α was proven for OA in [3]. Both of these results are for the continuous model where the speed can be set at an arbitrary value. In this paper, we design online heuristics for the discrete model where only a finite number of speeds are allowed for the processor but the highest speed is larger than the maximum speed needed by the online heuristic for the given job set in the continuous model.

We first consider a simple case. Suppose $s_1 > s > s_2$ and we have an execution interval of speed s which lasts for a time period of length t. Denote the energy consumption on this execution interval as E_c. Then, we adjust the speed s to s_1 and s_2 with appropriate proportion of time t_1 and t_2 in the execution interval so that the total workload executed remains the same, i.e., $st = s_1 t_1 + s_2 t_2$ and $t = t_1 + t_2$. We denote the energy consumption by the new schedule as E_d. We give an upper bound of E_d in Lemma 1.

Lemma 1. *If $P(s) = s^\alpha$ and $s_1/s_2 \leq \delta$, then $E_d \leq \frac{(\alpha-1)^{\alpha-1}(\delta^\alpha-1)^\alpha}{\alpha^\alpha(\delta-1)(\delta^\alpha-\delta)^{\alpha-1}} E_c$.*

Proof. Let $s = k \cdot s_2$ and $p = s_1/s_2$, then $1 \leq p \leq \delta$. By the definitions of E_c and E_d, we know that $1 \leq k \leq p$. Combining with the fact $st = s_1 t_1 + s_2 t_2$ and $t = t_1 + t_2$, we have $t_1 = \frac{(k-1)s_2}{s_1 - s_2} \cdot t$ and $t_2 = \frac{s_1 - ks_2}{s_1 - s_2} \cdot t$.

On the other hand, the power function $P(s) = s^\alpha$ implies $E_d = s_1^\alpha t_1 + s_2^\alpha t_2$ and $E_c = (ks_2)^\alpha t$. Therefore $E_d/E_c = \frac{(k-1)s_2 s_1^\alpha + (s_1 - ks_2)s_2^\alpha}{k^\alpha s_2^\alpha (s_1 - s_2)} = \frac{p^\alpha(k-1)+(p-k)}{k^\alpha(p-1)}$. Next we aim to find the maximum value achieved by $f(k, p) = \frac{p^\alpha(k-1)+(p-k)}{k^\alpha(p-1)}$ where $1 \leq k \leq p$ and $1 < p \leq \delta$. Notice that $f(k, p)$ increases with p, which corresponds to the fact that the competitive ratio increases when gaps between adjacent discrete speeds become larger. Therefore, $f(k, p) \leq f(k, \delta)$. By calculation, we know that the derivative $\frac{df(k,\delta)}{dk}$ is a decreasing function of k and $\frac{df(k,\delta)}{dk}|_{k=1} = (\delta^\alpha - \alpha\delta + \alpha - 1)/(\delta - 1) > 0$ while $\frac{df(k,\delta)}{dk}|_{k=\delta} = ((1-\alpha)\delta^\alpha + \alpha\delta^{\alpha-1} - 1)/(\delta^\alpha(\delta - 1)) < 0$. Therefore the value $f(k, p)$ is maximized to be $\frac{(\alpha-1)^{\alpha-1}(\delta^\alpha-1)^\alpha}{\alpha^\alpha(\delta-1)(\delta^\alpha-\delta)^{\alpha-1}}$ when $k = \frac{\alpha(\delta^\alpha-\delta)}{(\alpha-1)(\delta^\alpha-1)}$ because $\frac{df(k,\delta)}{dk} = 0$. □

Note that in order to make the lemma hold for all the adjustments, we implicitly need the assumption $s > s_d$ which means that s_d should be small enough. However, for $s < s_d$, if we round s to 0 and s_d, this part of energy will be less than the energy consumption of any schedule that finishes all the jobs using the given speed set observed by [13]. We define AVR_d as the schedule which adjusts each speed in AVR into proportioned execution of two adjacent speed levels. Notice that AVR_d is an offline schedule because online schedulers do not know when a new job will arrive and therefore do not know when the current piece of constant speed execution will end, which makes accurate adjustment impossible. By the definition of AVR_d, we have the following property by Lemma 1.

Corollary 1. *$E(AVR_d) \leq \frac{(\alpha-1)^{\alpha-1}(\delta^\alpha-1)^\alpha}{\alpha^\alpha(\delta-1)(\delta^\alpha-\delta)^{\alpha-1}} E(AVR) + E(OPT)$, where $\delta = \max_{1 \leq i \leq d-1} \frac{s_i}{s_{i+1}}$.*

We know that in the AVR online heuristic, whenever an arrival time or a deadline is met, the scheduler will possibly adopt a different execution speed, i.e., it will recalculate the execution speed. The online heuristic we design for the discrete model tries to capture the same idea. The only difference is that some speeds are not allowed in the discrete model. Therefore, speed adjustment is needed to satisfy the speed requirement. We describe the online scheduling algorithm AVR'_d for the

discrete model as follows. Whenever an arrival time or a deadline is met, the speed curve adopted by AVR is calculated. Then every speed is adjusted to adjacent speed levels according to the right proportion. For the contiguous time period which is originally assigned the same speed by the AVR, the heuristic AVR'_d executes the higher speed portion first and then the lower. If some job's arrival at time a interrupts this proportioned schedule, the scheduler just recalculates the new speed using AVR for the remaining jobs in the job set J under AVR'_d. This remaining job set is denoted as $J_{a,remain}$. The arrival time of the job whose arrival time is earlier than a will be adjusted to a if it is not finished by a and the workload of the jobs will also be updated reflecting the previous execution. Notice that AVR'_d is different from AVR_d because it is not a simple offline adjustment of AVR. We then aim to prove the following lemma:

Lemma 2. $E(AVR'_d) \leq E(AVR_d)$.

In order to prove the lemma, we sort the arrival times and deadlines into an increasing sequence $0 = v_0 \leq v_1 \leq v_2 \cdots \leq v_{2n-1}$ and call each interval $[v_i, v_{i+1}]$ $(0 \leq i \leq 2n-2)$ a *stable interval*. Notice that a stable interval may be a trivial point, and within one stable interval the online scheduler need not calculate a new execution speed because the execution speed allocation is decided at the beginning of that interval. We will prove Lemma 2 by induction on the number of non-trivial stable intervals contained in the whole interval.

We denote the energy consumed by schedule S within interval $[0, a]$ as $E(S[0, a])$, furthermore, if the schedule is for job set J, we write the energy consumption as $E(S(J)[0, a])$. Suppose that the first non-trivial stable interval is $[0, a]$. We can prove the following lemma which is then used to prove Lemma 2.

Lemma 3. $E(AVR'_d(J)[0, a]) + E(AVR_d(J_{a,remain})[a, 1]) \leq E(AVR_d(J)[0, 1])$.

Proof. If a corresponds to some job's deadline, then the decision of AVR_d and AVR'_d are the same in interval $[0, a]$. Therefore the lemma holds in this case.

If a corresponds to some job's arrival time, and suppose $[0, b]$ is the largest interval which uses the same speed s computed by AVR at time 0 ($s_{i+1} \leq s < s_i$). Because AVR_d is an offline algorithm, the workload executed in $[0, a]$ by AVR_d will be equivalent to $s \cdot a$. While the workload executed in $[0, a]$ by AVR'_d will be at least $s \cdot a$ because AVR'_d will execute the higher speed portion first. Therefore, for the jobs whose arrival times are 0, the remaining average workload in $[a, b]$ by adopting AVR'_d will be no more than (but still no less than s_{i+1}) the remaining average workload in $[a, b]$ by adopting AVR_d. Denote the non-negative difference of average workload as ρ and let $\Delta W = \rho(b - a)$ be the workload difference. According to the rules adopted by online heuristic AVR'_d, we have the following relation on energy consumption:

$$E(\mathrm{AVR}'_d(J)[0, a]) - E(\mathrm{AVR}_d(J)[0, a]) = \Delta W/(s_i - s_{i+1}) \cdot (s_i^2 - s_{i+1}^2) = \Delta W \cdot (s_i + s_{i+1}). \tag{1}$$

Then, we divide the interval $[a, b]$ into smaller intervals I_k $(1 \leq k \leq m)$ so that within the same smaller interval, the speed by AVR(J) and the speed by AVR($J_{a,remain}$) both remain constant. Notice that the speed used by AVR(J) will be no smaller than that used by AVR($J_{a,remain}$). Denote the non-negative difference of workload finished by these two schedules in interval I_k as ΔW_k. We know that these speeds are both larger than or equal to s_{i+1}. Therefore, the non-negative difference of the energy consumption by these two schedules will be no less than $\Delta W_k \cdot (s_i + s_{i+1})$. Notice that AVR($J$) and AVR($J_{a,remain}$) are the same in $[b, 1]$ and therefore $\Delta W = \sum_{i=1}^{m} \Delta W_k$. Summing up these energy consumption differences gives us the following relation:

$$E(\mathrm{AVR}_d(J)[a, 1]) - E(\mathrm{AVR}_d(J_{a,remain})[a, 1]) \geq \Delta W \cdot (s_i + s_{i+1}). \tag{2}$$

Combining inequalities (1) and (2), the lemma is proved. □

By using induction on the number of non-trivial stable intervals, we proved Lemma 2. We further combine Corollary 1 and Lemma 2 to derive the following theorem on the performance of AVR'_d when the power function is $P(s) = s^\alpha$.

Theorem 1. $E(AVR'_d) \leq (\frac{2^{\alpha-1}(\alpha-1)^{\alpha-1}(\delta^\alpha-1)^\alpha}{(\delta-1)(\delta^\alpha-\delta)^{\alpha-1}} + 1)E(OPT)$, *where* $\delta = \max_{1 \leq i \leq d-1} \frac{s_i}{s_{i+1}}$.

Let $[0, a]$ be the first non-trivial stable interval and s be the speed calculated by the heuristic H at time 0 for $[0, a]$ where $s_i > s \geq s_{i+1}$. Define $J_{a,remain}$ similarly as above and let the speed functions of the heuristic H for J and $J_{a,remain}$ in $[a, 1]$ be s_H and s_{H_r} respectively. Lemma 2 can be extended to a class of online heuristics besides AVR which satisfy the following two properties.

(1) Monotone Property: $s_H(t) \geq s_{H_r}(t)$ for $a \leq t \leq 1$.

(2) Separation Property: there exists $b \leq 1$ where $s_{H_r}(t) \geq s_{i+1}$ for $a \leq t \leq b$ and $s_H(t) = s_{H_r}(t)$ for $b < t \leq 1$.

The proof techniques used in Lemma 3 still work for online heuristics that satisfy the above two properties. Therefore, we have the following theorem.

Theorem 2. *If H is a c-competitive online heuristic for the continuous model which satisfies the Monotone Property and the Separation Property, then the heuristic H'_d for the discrete model which adjusts the calculated speed to adjacent speed levels and executes the higher speed part first is* $(\frac{c(\alpha-1)^{\alpha-1}(\delta^\alpha-1)^\alpha}{\alpha^\alpha(\delta-1)(\delta^\alpha-\delta)^{\alpha-1}} + 1)$*-competitive.*

4. Minimizing energy while maximizing throughput

In most systems, the processor can only vary its speed between 0 and a maximum speed s_{max}. Because the speed is bounded from above, the processor may be overloaded with jobs and no scheduling algorithms can finish all the jobs before their deadlines. In this section, we investigate the continuous model with a speed limit and investigate the offline throughput maximization problem in this overloaded setting where preemptions are allowed. Throughput is defined to be the total workload of the jobs completed by their deadlines. We propose a greedy algorithm which produces a schedule that is a 3-approximation to the maximum throughput, while at the same time achieves $\frac{(\alpha-1)^{\alpha-1}(3^\alpha-1)^\alpha}{2\alpha^\alpha(3^{\alpha-1}-1)^{\alpha-1}}$-approximation with respect to energy consumption.

A tool called *s*-schedule was introduced in [8] which can provide useful information regarding the optimal speed function for J without explicitly computing it. The properties of the *s*-schedule are also crucial in our proof of approximation ratios for the greedy algorithm. For easy reference, we give the relevant definitions and properties below.

Definition 2. For any constant s, the *s*-schedule for J is an EDF schedule which uses constant speed s in executing any jobs of J. It will give up a job when the deadline of the job has passed. In general, *s*-schedules may have idle periods or unfinished jobs.

Definition 3. In a schedule S, a maximal subinterval of $[0, 1]$ devoted to executing the same job j_k is called an execution interval for j_k (with respect to S). Denote by $I_k(S)$ the union of all execution intervals for j_k with respect to S. Execution intervals with respect to the *s*-schedule will be called *s*-execution intervals.

It is easy to see that the *s*-schedule for n jobs contains at most $2n$ *s*-execution intervals, since the end of each execution interval (including an idle interval) corresponds to the moment when either a job is finished or a new job arrives. Also, the *s*-schedule can be computed in $O(n \log n)$ time by using a priority queue to keep all jobs currently available, prioritized by their deadlines. Denote the *s*-schedule of J by S_J. Next we prove a basic property of *s*-schedules.

Lemma 4. *Given a job set J, for any $J' \subseteq J$, we have $|\bigcup_{j_k \in J'} I_k(S_J)| \leq |\bigcup_{j_k \in J'} I_k(S_{J'})|$.*

Proof. For a job j_k in J', if $|I_k(S_J)| = R_k$, which means that in S_J the job j_k can be finished with all the competing jobs in J whose deadlines are earlier than j_k, then j_k can also be finished in $S_{J'}$ where less jobs are competing with it. If $|I_k(S_J)| < R_k$, which means that in S_J, the job j_k is not finished, then in $S_{J'}$ the job j_k will be executed for no less time because: (1) Job j_k has the highest priority among all unfinished jobs in $I_k(S_J)$; (2) Job j_k may have its priority increased in $[0, 1]\backslash I_k(S_J)$ due to the removal of jobs in $J\backslash J'$. This completes the proof. □

In the following, we will use a slightly different result for the *s*-partition introduced in [8] and summarize it into the following lemma.

Lemma 5. *The job set J can be partitioned into two job sets $J^{\leq s}$ and $J^{>s}$ in $O(n \log n)$ time, where jobs in $J^{\leq s}$ require speeds no larger than s in OPT while jobs in $J^{>s}$ require speeds larger than s in OPT.*

Recall that OPT is the min-energy schedule without speed limit. We use opt_T to denote the optimal schedule with speed upper bounded by s_{max}, and opt_T maximizes the throughput first and then minimizes the energy subject to this throughput. Given a job set, we first remove all the jobs whose average workload is more than s_{max} because these jobs will not contribute to the throughput even in opt_T. We denote the remaining job set as J. Then, we generate an s_{max}-schedule for J. By Lemma 5, we can identify critical intervals whose speeds are larger than s_{max}. We denote the union of those intervals as I. According to the way critical intervals are chosen, there is a schedule where each job j_k with $[a_k, b_k] \cap I \neq [a_k, b_k]$ can be finished by its deadline using execution speed at most s_{max} and is only executed outside I. We denote the set of jobs satisfying the above requirement as J_2. We know that all the jobs in J_2 can contribute to the throughput without affecting the feasibility of jobs in $J_1 = J\backslash J_2$. Therefore, we only consider jobs in J_1 in the following discussion.

Let $t = |I|$. We propose a greedy algorithm which achieves at least $\frac{1}{3}t \cdot s_{max}$ throughput within I as follows.

The interval T_k^* is obtained from T_k in the following way. If $|T_k| = \frac{R_k}{s_{max}}$, then let T_k^* be T_k; otherwise, give all the time in T_k to T_k^* and then add more time to T_k^* in a backward manner starting from b_k until $|T_k^*| = \frac{R_k}{s_{max}}$. For example, if $T_k = [0, 1], [4, 5]$, $b_k = 6$ and $\frac{R_k}{s_{max}} = 4$, then $T_k^* = [0, 1], [3, 6]$.

The set $J^* \cup J_2$ is the subset of jobs we choose to execute in the schedule. The feasibility of S^* using speeds at most s_{max} is guaranteed by the choice of jobs into J^*.

We can prove that jobs in J^* have a total workload at least $\frac{1}{3}t \cdot s_{max}$ by proving the following lemma.

Lemma 6. *Given a set J_h of n jobs, where the average workload of every job is no more than s_{max} and OPT always uses speeds higher than s_{max}, the total workload R of the jobs added to J^* in the first iteration of Algorithm 1 is at least $\frac{1}{3}(t - t')s_{max}$ where t represents the length of the union of jobs in J_h at the beginning of the iteration and t' represents the length of the union of jobs in J_h at the end of the iteration.*

Algorithm 1 Greedy Algorithm

1. $J^* = \emptyset$ and $J_h = J_1$.
2. $I = \cup_{j_i \in J_1}[a_i, b_i]$.

repeat

3. $J^* = J^* \cup j_k$, where $j_k \in J_h$ and $R_k \geq R_i$ for all $j_i \in J_h$.
4. Compute the s_{max}-schedule for J_h and denote the time intervals allocated to j_k as T_k.
5. Assign time intervals T_k^* to j_k (The way to choose T_k^* is explained below).
6. $I = I \backslash T_k^*$.
7. Update the job arrival times and deadlines after removing T_k^* from I (by treating time in T_k^* as non-existent).
8. Remove from J_h the job j_k and the jobs whose average workload are larger than s_{max} after the update.
9. Compute the s_{max}-partition for J_h on I using Lemma 5 and move the jobs in $J_h^{\leq s_{max}}$ from J_h to J^*.

until $J_h \neq \emptyset$.

10. Compute the optimal schedule S^* for $J^* \cup J_2$.

Proof. First, it is easy to see that when Step 6 is finished, the s_{max}-schedule for $J_h \backslash j_k$ on I (we denote this schedule as S' in the following proof) runs at a non-zero speed for a total period of $t - \frac{R_k}{s_{max}}$. We next prove that the s_{max}-schedule for J_h computed in Step 9 runs at a non-zero speed for a total period of at least $t - 3\frac{R_k}{s_{max}}$. Notice that in Step 8, if a job j_i is removed from J_h, then its interval $[a_i, b_i]$ should intersect with T_k^*, otherwise its average workload will not change after we remove T_k^* from I. Furthermore, $[a_i, b_i]$ should intersect with the last interval in T_k^*. Because if $b_i > b_k$, then $[a_i, b_i]$ has to intersect the last interval in T_k^*, otherwise it will not intersect T_k^*; if $b_i \leq b_k$, then $[a_i, b_i]$ only intersecting non-last intervals in T_k^* means that in the s_{max}-schedule for J_h (with j_k in), the job j_i can finish all its workload before the earliest intersection point; otherwise, the intersecting part with the non-last intervals in T_k^*, which is also an interval in T_k will not be used to execute j_k because j_i has a higher priority than j_k, therefore removing T_k^* will not make the average workload of j_i be larger than s_{max}. Furthermore, it is easy to see that $|[a_i, b_i] \backslash T_k^*| \leq \frac{R_i}{s_{max}} \leq \frac{R_k}{s_{max}}$. Based on the above analysis, we know that the jobs removed from J_h in Step 8 occupy a time period with length at most $2\frac{R_k}{s_{max}}$ (it is the time in $I \backslash T_k^*$ starting from both sides of the last interval of T_k^* with length at most $\frac{R_k}{s_{max}}$ on each side), which implies that the remaining jobs in J_h after Step 8 will be executed for a period of at least $t - 3\frac{R_k}{s_{max}}$ in schedule S'. Therefore, the s_{max}-schedule for J_h computed in Step 9 runs at non-zero speed for a total period of at least $t - 3\frac{R_k}{s_{max}}$ by Lemma 4. In Step 9, by moving more jobs from J_h to J^*, we increase the workload in J^* by at least $t_r \cdot s_{max}$ where t_r is the execution time of those jobs in s_{max}-schedule for J_h computed in Step 9. Notice that the removal of jobs in Step 9 will not affect the s_{max}-schedule for the remaining jobs in J_h by the definition of critical intervals. From the above analysis, we have the following two relations: $t' + t_r \geq t - 3\frac{R_k}{s_{max}}$ and $\frac{R}{s_{max}} \geq \frac{R_k}{s_{max}} + t_r$. The lemma then follows. □

We can interpret Lemma 6 in the following way. Picking the job j_k with the maximum R_k value will reduce the *useful* s-execution time by at most $3\frac{R_k}{s_{max}}$. We remark that T_k^* cannot be chosen arbitrarily within $[a_k, b_k]$. For example, if we choose T_k^* to be $[a_k, a_k + \frac{R_k}{s_{max}}]$, then picking the job j_k may reduce the useful s-execution time by $4\frac{R_k}{s_{max}}$ because the time in $I_k(S_{J_h})$ (s_{max}-execution intervals of j_k) may not overlap with any other job in J_h and therefore cannot be used to execute other jobs in the s_{max}-schedule.

Lemma 7. *The jobs in J^* have a total workload of at least $\frac{1}{3}t \cdot s_{max}$ and can be feasibly scheduled using speeds at most s_{max}.*

Proof. We prove the lemma by induction on n, the number of jobs contained in J_h. When $n = 1, 2, 3$, the lemma easily holds. Assume the lemma holds for $n \leq k$. When $n = k + 1$, let t represent the length of the union of jobs in J_h at the beginning of the iteration and let t' represent the length of the union of jobs in J_h at the end of the iteration. After doing the first iteration of the greedy algorithm, the workload of the jobs newly added into J^* is at least $\frac{1}{3}(t - t')s_{max}$ by Lemma 6. By the induction hypothesis, in the remaining set J_h (which satisfies the conditions in Lemma 6 due to the removal of jobs in Step 9), jobs with at least $\frac{1}{3}t' \cdot s_{max}$ workload will be chosen into J^*. The feasibility of J^* follows naturally from the algorithm. This completes the proof. □

The above lemma directly shows that Algorithm 1 is a 3-approximation in throughput compared to opt_T because opt_T can at most select $t \cdot s_{max}$ workload from J_1 to execute. Next, we show that the energy consumption of S^* is at most $\frac{(\alpha-1)^{\alpha-1}(3^\alpha-1)^\alpha}{2\alpha^\alpha(3^{\alpha-1}-1)^{\alpha-1}}$ times that of opt_T.

Lemma 8. *The schedule S^* is a $\frac{(\alpha-1)^{\alpha-1}(3^\alpha-1)^\alpha}{2\alpha^\alpha(3^{\alpha-1}-1)^{\alpha-1}}$-approximation of energy consumption compared to opt_T.*

Proof. Denote the union of job intervals in J_1 as I_1 and let $t_1 = |I_1|$. Denote the union of job intervals in J_2 as I_2. Notice that opt_T must schedule some of the jobs in J_1 with at least $\frac{t_1}{3}s_{max}$ workload and all the jobs in J_2. This also implies that the average speed of opt_T within I_1 is at least $\frac{s_{max}}{3}$. Let S' be the schedule which executes J^* within I_1 optimally and executes J_2 outside I_1

optimally. Since S^* is optimal for $J^* \cup J_2$, we have $E(S^*) \leq E(S')$. Denote the set of jobs in J_1 selected by opt_T as J_1'. We then relax the interval of every job in J_1' to be a maximal interval in I_1 and get a job set J_1''.

We first discuss the case when I_1 is a single interval. We use opt_T^r to denote the optimal schedule for $J_1'' \cup J_2$. It is easy to see that $E(opt_T^r) \leq E(opt_T)$. Let S be a schedule which executes jobs in J_2 as S' does and executes jobs in J_1'' using a constant speed $s \geq \frac{s_{max}}{3}$ throughout I_1. It is easy to show that $E(S') \leq 3^{\alpha-1}E(S)$. Next we compare S and opt_T^r. The reason opt_T^r executes some jobs in J_2 within I_1 is that S executes those jobs with a speed larger than s. The schedule opt_T^r reduces energy consumption by merging two speeds of S (one speed for executing jobs in J_1 and one for executing jobs in J_2) into the same speed while taking suitable portion of execution time. Although the real adjustment from S to opt_T^r can be quite complicated, we can equivalently assume that they are done in parallel for different critical intervals containing jobs in J_2. We can also assume that each job in J_2 which is involved in some merge operations is only involved in one merge operation with jobs in J_1''. From the observation above, each involved job in J_2 is executed with a speed larger than s in S. By the way critical intervals are chosen, we can also claim that no workloads are shifted from one critical interval containing jobs in J_2 to another that contains jobs in J_2. By Lemma 1, the energy reduction by merging is at most $1 - \frac{2\alpha^\alpha(3^\alpha-3)^{\alpha-1}}{(\alpha-1)^{\alpha-1}(3^\alpha-1)^\alpha}$ (because $\delta = \frac{s_{max}}{s} \leq 3$), which implies that $E(opt_T^r) \geq \frac{2\alpha^\alpha(3^\alpha-3)^{\alpha-1}}{(\alpha-1)^{\alpha-1}(3^\alpha-1)^\alpha}E(S)$. Hence, the lemma is true in this case. If I_1 consists of multiple intervals, we can use similar arguments to prove the correctness of the lemma because in each maximal interval I_{1i} of I_1 with length t, the jobs in I_1' whose intervals are contained in I_{1i} have a total workload of at least $\frac{t}{3}s_{max}$. This ends the proof. □

We summarize the above results into the following theorem.

Theorem 3. *Algorithm 1 generates a schedule S^* in $O(n^2 \log n)$ time which is 3-approximation of throughput and a $\frac{(\alpha-1)^{\alpha-1}(3^\alpha-1)^\alpha}{2\alpha^\alpha(3^{\alpha-1}-1)^{\alpha-1}}$-approximation of energy consumption compared to opt_T.*

The running time of Algorithm 1 is $O(n^2 \log n)$ because every step inside the loop takes at most $O(n \log n)$ time and the computation of S^* takes $O(n^2 \log n)$ time [2].

5. Conclusion

In this paper, we first investigate online heuristics for the discrete model. By suitably adjusting the speeds used in AVR to adjacent speed levels, we design an online heuristic AVR_d' for the discrete model and prove it to be $(\frac{2^{\alpha-1}(\alpha-1)^{\alpha-1}(\delta^\alpha-1)^\alpha}{(\delta-1)(\delta^\alpha-\delta)^{\alpha-1}} + 1)$-competitive where $P(s) = s^\alpha$ and δ is the maximum ratio between adjacent non-zero speed levels. We also identify a class of heuristics which can be modified in a similar way from the continuous model to the discrete model with their competitive ratios increased by a constant factor $\frac{(\alpha-1)^{\alpha-1}(\delta^\alpha-1)^\alpha}{\alpha^\alpha(\delta-1)(\delta^\alpha-\delta)^{\alpha-1}}$ with an extra addition of 1. Then we study the throughput maximization problem for the preemptive continuous model in the overloaded setting where the maximum speed of the processor is upper bounded by s_{max}. We propose a greedy algorithm to approximately maximize throughput while also approximately minimizing the energy consumption. By using the properties of the s-schedule defined in [8] and properties proved in the study of online heuristics for the discrete model, we prove that the greedy algorithm is a 3-approximation of throughput and a $\frac{(\alpha-1)^{\alpha-1}(3^\alpha-1)^\alpha}{2\alpha^\alpha(3^{\alpha-1}-1)^{\alpha-1}}$-approximation of energy consumption.

References

[1] F. Yao, A. Demers, S. Shenker, A scheduling model for reduced CPU energy, in: Proceedings of the 36th Annual Symposium on Foundations of Computer Science, FOCS, 1995, pp. 374–382.
[2] M. Li, A.C. Yao, F.F. Yao, Discrete and continuous min-energy schedules for variable voltage processors, in: Proceedings of the National Academy of Sciences of USA, 103, 2006, pp. 3983–3987.
[3] N. Bansal, T. Kimbrel, K. Pruhs, Speed scaling to manage energy and temperature, Journal of the ACM 54 (1) (2007) (Article No. 3).
[4] G. Quan, X.S. Hu, Energy efficient fixed-priority scheduling for hard read-time systems, in: Proceedings of the 36th Conference on Design Automation, 1999, pp. 134–139.
[5] H.S. Yun, J. Kim, On energy-optimal voltage scheduling for fixed-priority hard real-time systems, ACM Transactions on Embedded Computing Systems 2 (3) (2003) 393–430.
[6] T. Ishihara, H. Yasuura, Voltage scheduling problem for dynamically variable voltage processors, in: Proceedings of International Symposium on Low Power Electronics and Design, 1998, pp. 197–202.
[7] W. Kwon, T. Kim, Optimal voltage allocation techniques for dynamically variable voltage processors, in: Proceedings of the 40th Conference on Design Automation, 2003, pp. 125–130.
[8] M. Li, F. Yao, An efficient algorithm for computing optimal discrete voltage schedules, SIAM Journal on Computing 35 (3) (2005) 658–671.
[9] M.R. Garey, D.S. Johnson, Two processor scheduling with start times and deadlines, SIAM Journal on Computing 6 (3) (1977) 416–426.
[10] F.C.R. Spieksma, On the approximability of an interval scheduling problem, Journal of Scheduling 2 (1999) 215–227.
[11] A. Bar-Noy, S. Guha, Approximating the throughput of multiple machines in real-time scheduling, SIAM Journal on Computing 31 (2) (2001) 331–352.
[12] G. Koren, D. Shasha, D^{over}: an optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems, SIAM Journal on Computing 24 (2) (1995) 318–339.
[13] H.L. Chan, W.T. Chan, T.W. Lam, L.K. Lee, K.S. Mak, P. Wong, Energy efficient online deadline scheduling, in: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, 2007, pp. 795–804.
[14] S. Irani, K. Pruhs, Online algorithms: algorithmic problems in power management, ACM SIGACT News 36 (2) (2005) 63–76.