Energy Efficient Online Deadline Scheduling

Ho-Leung Chan* Wun-Tat Chan† Tak-Wah Lam* Lap-Kei Lee* Kin-Sum Mak*
Prudence W.H. Wong‡

Abstract. This paper extends the study of online algorithms for energy-efficient deadline scheduling to the overloaded setting. Specifically, we consider a processor that can vary its speed between 0 and a maximum speed T to minimize its energy usage (of which the rate is roughly a cubic function of the speed). As the speed is upper bounded, the system may be overloaded with jobs and no scheduling algorithms can meet the deadlines of all jobs. An optimal schedule is expected to maximize the throughput, and furthermore, its energy usage should be the smallest among all schedules that achieve the maximum throughput. In designing a scheduling algorithm, one has to face the dilemma of selecting more jobs and being conservative in energy usage. Even if we ignore energy usage, the best possible online algorithm is 4-competitive on throughput [12]. On the other hand, existing work on energy-efficient scheduling focuses on minimizing the energy to complete all jobs on a processor with unbounded speed, giving several $O(1)$-competitive algorithms with respect to the energy usage [2,20]. This paper presents the first online algorithm for the more realistic setting where processor speed is bounded and the system may be overloaded; the algorithm is $O(1)$-competitive on both throughput and energy usage. If the maximum speed of the online scheduler is relaxed slightly to $(1+\epsilon)T$ for some $\epsilon > 0$, we can improve the competitive ratio on throughput to arbitrarily close to one, while maintaining $O(1)$-competitive on energy usage.

1 Introduction

Deadline scheduling. Let us first review a classical online problem of deadline scheduling. We are given a processor of speed T, which can do T units of work in one unit of time. Jobs arrive online at unpredictable times; the work and deadline of a job are known when the job arrives. The aim is to design an (online) algorithm that maximizes the throughput, which is the total work of the jobs completed by their deadlines (see, e.g., [4,7,12]). We assume preemption is allowed, and a preempted job can be resumed at the point of preemption. Note that a processor of speed T may not meet the deadlines of all jobs, i.e., the processor is overloaded. An algorithm A is said to be c-competitive, where $c \geq 1$, if for any job sequence, A obtains a throughput at least $1/c$ of the best offline schedule. Koren and Shasha [12] have showed that the online algorithm D^{over} is 4-competitive and no online algorithm can be better than 4-competitive. If the job sequence is restricted to be underloaded, the algorithm EDF (earliest deadline first) guarantees to complete all jobs in time and is thus 1-competitive [7].

Energy efficiency. Recent development on mobile devices makes energy efficiency a major concern as these devices are battery-operated. A popular technology to reduce energy usage is to allow variable processor speed, which is commonly known as *dynamic voltage scaling* (see, e.g., [8,15,19]). As the rate of energy usage P required to run a processor at speed s is believed to be roughly s^α where $\alpha \geq 2$ [5], it is more energy efficient to schedule a job at a low speed whenever possible. In this paper, we assume that the online algorithm can adjust the speed of the processor to any value in $[0, T]$ where T is fixed in advance [15], and we assume a general rate of energy usage in the form s^α. Given a job sequence, an optimal schedule maximizes the throughput, while minimizing the energy usage subject to this throughput. Our primary concern is whether there exists an online algorithm that can be $O(1)$-competitive on throughput and $O(1)$-competitive on energy usage, i.e., the throughput and energy usage are respectively at least $1/c$ and at most c' times of that of an optimal schedule, where c and c' are constants.

Previous work. Energy efficient algorithms for deadline scheduling are first studied by Yao et al. [20]. They considered the case where the processor can run at any speed in $[0, \infty)$, and can always complete a job sequence without missing a deadline. In this case, both the online and the offline algorithm aim to complete the entire job sequence, the only concern is the speed and energy usage. Yao et al. [20] gave a simple algorithm called AVR for speed determination. When coupled with EDF, AVR gives an online algorithm that

*Department of Computer Science, University of Hong Kong, Hong Kong. Email: `{hlchan, twlam, lklee, ksmak}@cs.hku.hk`
†Department of Computer Science, King's College London, UK. Email: `jchan@dcs.kcl.ac.uk`
‡Department of Computer Science, University of Liverpool, UK. Email: `pwong@csc.liv.ac.uk`

	unbounded max speed	fixed max speed	fixed discrete speed levels
Throughput	1	14	14
Energy usage	$2^\alpha\alpha^\alpha$ [20], α^α [2, 20] $2(\alpha/(\alpha-1))^\alpha e^\alpha$ [2]	$(\alpha^\alpha + \alpha^2 4^\alpha)$	$\Delta^\alpha(\alpha^\alpha + \alpha^2 4^\alpha) + 1$

Table 1: The performance guarantee (in terms of competitive ratios) for three different settings

is $2^\alpha\alpha^\alpha$-competitive on energy usage. They also proposed another online algorithm OA (Optimal Available), which was later shown by Bansal et al. to be α^α-competitive [2]. Bansal et al. [2] further improved the result with a new algorithm that is $2(\alpha/(\alpha-1))^\alpha e^\alpha$-competitive. This algorithm is also e-competitive with respect to the maximum speed. On the other hand, Li et al. [13] have considered structured jobs and shown that AVR has a better performance.

Our contribution. In this paper, we consider energy-efficient deadline scheduling on a processor with a fixed maximum speed T and the system may be overloaded. We give an algorithm, called FSA(OAT) below, which is 14-competitive on throughput and $(\alpha^\alpha + \alpha^2 4^\alpha)$-competitive on energy usage. If $T = \infty$, then this algorithm is 1-competitive on throughput (all jobs are completed) and α^α-competitive on energy usage. That is, its behavior is identical to OA.

As the maximum speed is bounded, an online scheduling algorithm may not be able to finish all jobs. It needs two kinds of strategies, one for selecting jobs and one determining the speed (as a function over time). In this paper, we consider a simple job selection strategy called FSA (Full Speed Admission). Slightly oversimplifying, FSA attempts to admit a new job J for processing whenever it finds that using the maximum speed, it is feasible to complete J together with the remaining work of all admitted jobs. Such a feasibility test makes FSA very aggressive in admitting new jobs. In Section 2, we will show that if FSA is coupled with a speed function that allows FSA to finish all jobs ever admitted, then FSA is 14-competitive on throughput. In the full paper, we will give a more complicated analysis to improve the competitive ratio to 10.

The key question is whether there is a speed function that is conservative in energy usage and can make FSA finish all jobs ever admitted. To this end, we make use of the previously known algorithm OA, which is for scheduling jobs on a processor with unbounded speed. We denote OAT (Optimal Available, at most T) to be the speed function which, at any time, takes the minimum of T and the speed used by OA. We show that FSA when coupled with OAT can complete all jobs admitted and is thus 14-competitive on throughput. And OAT is conservative in energy usage in the bounded-speed setting, i.e., the energy usage of OAT is at most a constant times of that of any algorithm that maximizes the throughput on a processor with a maximum speed T. The analysis of OAT is non-trivial. It stems from an intrigue classification of underloaded and overloaded periods and the observation that an optimal schedule (which maximizes the throughput) must complete all jobs in underloaded periods and at least a fraction of the largest possible amount of work that can be completed for the remaining jobs.

The result on FSA and OAT can be applied to the following three variations:

• **Discrete speed levels.** Very recently, scheduling on a processor with a fixed number of discrete speed levels has also attracted attention [11,14]; in particular, assuming the underloaded setting, Li and Yao [14] have devised a polynomial-time offline algorithm for finding a schedule with optimal energy usage. But not much has been known for the overloaded setting, let alone competitive online algorithms. Note that FSA(OAT) can be adapted to discrete speed levels; we simply set the maximum speed to be the highest speed level and round up the speed function to the next higher level. A careful analysis would show that this algorithm is still 14-competitive on throughput and $(\Delta^\alpha(\alpha^\alpha+\alpha^2 4^\alpha)+1)$-competitive on energy usage, where Δ is the largest ratio of two consecutive (non-zero) speed levels (e.g., if the speed levels are uniformly distributed between $[0, T]$, then $\Delta = 2$).

• **Better throughput via max speed relaxation.** Note that even if the energy concern is ignored, no online algorithm can be better than 4-competitive on throughput [12]. To obtain a better performance, we also consider in this paper the possibility of compensating the online algorithm with a higher maximum speed. We show that if the maximum speed of the online scheduler is relaxed to $(1+\epsilon)T$, for any $\epsilon > 0$, there exists an online algorithm that is $(1 + 1/\epsilon)$-competitive on throughput and $(1 + \epsilon)^\alpha(\alpha^\alpha + \alpha^2 4^\alpha)$-competitive on energy usage.

• **Jobs with arbitrary value.** In some applications, the throughput is measured by the value or profit of jobs, which is not related to the amount of work [12]. In this more general case, a straightforward adaptation of FSA can give an online algorithm that is $14k$-competitive on throughput and $(\alpha^\alpha + \alpha^2 2^\alpha(1 + k)^\alpha)$-competitive on energy usage, where k is the importance ratio (i.e., the ratio of the largest possible value per unit work to the smallest possible one).

Remarks. The literature also contains results on other interesting aspects of energy efficient scheduling [10]. Irani et al. [9] extended the result on AVR [20] to a setting where the processor has a sleep state, and showed that the extension increases the competitive ratio on energy by only a constant factor. We conjecture that using a similar technique, the algorithm in this paper can also be adapted to allow a sleep state. On the other hand, Pruhs et al. [16] have studied the offline problem of minimizing total flow time subject to a fixed amount of energy, while Albers and Fujiwara [1] have studied the online problem of minimizing a cost consisting of the energy usage and the total flow time; both results focus on unit-size jobs. Furthermore, the offline problem of minimizing the makespan subject to a fixed amount of energy has been studied in [6, 17]. Another practical concern is the maximum temperature of the processor as the temperature is related to energy usage. Several interesting results have been reported in [2, 3].

Organization of paper. The rest of the paper is organized as follows. In Section 2, we show that if FSA is coupled with a speed function that allows FSA to finish all jobs admitted, then FSA is 14-competitive. We then define and prove the property of such speed function OAT in Section 3. In Section 4, we prove that the speed function OAT is conservative in energy usage. We then discuss the speed relaxation results in Section 5. Due to space limitation, the results on arbitrary value and discrete speed levels are omitted.

Notations. For any job J, we denote the release time, work and deadline of J as $r(J)$, $w(J)$, and $d(J)$, respectively. The span of J, denoted $\rho(J)$ or $span(J)$, is the time interval $[r(J), d(J)]$. For any set of jobs L, let $w(L)$ denote the total work of all jobs in L. To ease our discussion, we assume that an algorithm will not process a job after missing its deadline, and whenever we say that a job is completed, it is always meant to be completed by the deadline.

2 Throughput analysis of FSA

This section presents the details of FSA, which is a strategy for selecting jobs for possible scheduling. Note that FSA itself does not specify the processor speed. To define a schedule, we need to supplement FSA with a speed function f. The resulting scheduling algorithm will be referred to as FSA(f). The main result in this section is that if the function f is fast enough for FSA(f) to complete every job it has committed, then FSA(f) is 14-competitive on throughput.

FSA maintains an *admitted list* of jobs. Jobs that are not admitted will not get scheduled. At any time, FSA runs the job in the admitted list with the earliest deadline. The admitted list is updated as follows.

Job Arrival. When a job J arrives, let $J_1, J_2, \ldots, J_n$ be the jobs currently in the admitted list, where $d(J_1) \le d(J_2) \le \cdots \le d(J_n)$.

- J is admitted if J together with $J_1, J_2, \ldots, J_n$ are full-speed admissible. [A set S of jobs is said to be *full-speed admissible* at a certain time if, when using the maximum speed T onwards, the remaining work of every job in S can be completed by its deadline.]
- Otherwise, J can still be admitted if $w(J) > 2(w(J_1) + w(J_2) + \cdots + w(J_k))$ and $\{J, J_{k+1}, J_{k+2}, \ldots, J_n\}$ are full-speed admissible, where k is the smallest possible integer in $[1, n]$. In this case, $J_1, \ldots, J_k$ will be *expelled* from the admitted list.

Job Completion. When a job J finishes, J is removed from the admitted list.
Job Overdue. At the deadline of a job J, if job J is still in the admitted list but has not yet finished, J is removed from the admitted list.

Note that a job admitted at release time may be expelled due to other bigger jobs released later. A job is said to be admitted *perennially* if this job, after being admitted, never gets expelled due to bigger jobs. Note that if the online algorithm always runs at speed T, then no job will become overdue and all jobs admitted perennially will be completed by their deadlines. In general, for an arbitrary speed function f, FSA(f) may not be able to complete all jobs admitted perennially.

Definition 2.1. FSA(f) is said to be *honest* if for any job sequence I, FSA(f) completes all jobs admitted perennially on or before their deadlines.

Theorem 2.1. *If* FSA(f) *is honest, then* FSA(f) *is* 14-*competitive on throughput.*

The rest of this section is devoted to proving Theorem 2.1. In the next section, we will describe a speed function OAT and show that FSA(OAT) is honest. Assume that FSA(f) is honest and consider any job sequence I. Let $A \subseteq I$ be the set of jobs that have been admitted by FSA(f) at their release time, and let $N = I - A$. We further divide A into C and E such that C is the set of jobs that are admitted perennially, and $E = A - C$. Since FSA(f) is honest, FSA(f) completes exactly the jobs in C and expels all jobs in E. The competitiveness of FSA(f) (i.e., Theorem 2.1) stems from the following upper bounds on E and N.

Lemma 2.1. *$w(E) \le w(C)$.*

Proof. For every job X in A that forces another job J in E to be expelled from the admitted list, we link up X and J so that X is the parent of J. This parent relationship forms a forest, and the root of each tree is a job in C and all other nodes are jobs in E. By the definition of expelling jobs, the work of a parent is at least two times the total work of its children. Thus, for each root r, we have $w(r)$ at least the total work of all other nodes in the tree. Sum over all trees, we obtain the relationship $w(E) \leq w(C)$. □

Consider the union of the spans of all jobs in N, which may cover one or more intervals. Let $\ell = |\bigcup_{X \in N} \rho(X)|$ be the total length of these intervals. The total work of N might be huge, yet the amount of work of N that can be completed (by the optimal algorithm) is bounded by ℓT. Below we give an upper bound of ℓT.

LEMMA 2.2. $\ell T \leq 6\, w(A)$.

With Lemmas 2.1 and 2.2, proving Theorem 2.1 is straightforward. Consider any optimal algorithm. It can at most complete all jobs in A. With respect to N, the jobs that can be completed have a total work at most ℓT, which, by Lemma 2.2, is at most $6\, w(A)$. Thus, the total amount of work completed is at most $7\, w(A)$, or equivalently, $7(w(C) + w(E))$. By Lemma 2.1, $w(E) \leq w(C)$ and $7(w(C) + w(E)) \leq 14w(C)$. Recall that FSA($f$) completes all jobs in C, attaining a throughput of $w(C)$. Thus, Theorem 2.1 follows.

It remains to prove the upper bound of ℓ. Recall that $\ell = |\bigcup_{X \in N} \rho(X)|$. We will define, for each job $X \in N$, an interval $\rho'(X)$ that encloses $\rho(X)$. Then $\ell \leq |\bigcup_{X \in N} \rho'(X)|$. The way ρ' is defined allows us to upper bound each $|\rho'(X)|$ in terms of the jobs in A whose deadlines are in $\rho'(X)$. (see Lemma 2.3). The following simple observation further shows that ℓ can be upper bounded by considering a subset M of N with all elements X of M having disjoint $\rho'(X)$. Then we can easily show that $\ell T \leq 6w(A)$.

FACT 2.1. *N contains a subset M such that all elements X of M have mutually disjoint $\rho'(X)$, and* $|\bigcup_{X \in N} \rho'(X)| \leq 2 \sum_{X \in M} |\rho'(X)|$. [1]

We have yet to define $\rho'(X)$ and relate it to jobs in A whose deadlines are in $\rho'(X)$.

[1]The argument is as follows: Let M_o be a minimal subset of N such that $\bigcup_{X \in M_o} \rho'(X) = \bigcup_{X \in N} \rho'(X)$. I.e., M_o and N define the union of time intervals. Note that no three jobs in M_o have their intervals overlapping at a common time. We can further partition M_o into two disjoint subsets such that in each subset, no two intervals overlap. Let M be the subset whose union of intervals has a bigger total length. Then we have $\ell \leq |\bigcup_{X \in M_o} \rho'(X)| \leq 2|\bigcup_{X \in M} \rho'(X)| \leq 2 \sum_{X \in M} |\rho'(X)|$.

Definition. At time $r(X)$ (i.e., when X is released), let $S(X) = \{J_1, J_2, \ldots, J_n\}$ be the jobs in the admitted list, where $d(J_1) \leq d(J_2) \leq \cdots \leq d(J_n)$. Note that X is in N, and X and $S(X)$ together are not full-speed admissible. Let $k \leq n$ be the smallest integer such that $X, J_k, \ldots, J_n$ are not full-speed admissible, but $X, J_{k+1}, \ldots, J_n$ are. Furthermore, let $m \leq n$ be the smallest integer such that $X, J_k, J_{k+1}, \ldots, J_m$ are not full-speed admissible. Let $\rho'(X) = [r(X), s]$, where $s = \max\{d(X), d(J_m)\}$. Furthermore, denote $S_1(X) = \{J_1, \ldots, J_k\}$, and $S_2(X) = \{J_k, \ldots, J_m\}$. Note that each job in $S_1(X)$ or $S_2(X)$ has deadline in $\rho'(X)$.

LEMMA 2.3. *For any job $X \in N$, $|\rho'(X)|T < 2w(S_1(X)) + w(S_2(X)) \leq 3w(A|_{\rho'(X)})$, where $A|_{\rho'(X)}$ is the subset of jobs in A whose deadlines are in $\rho'(X)$.*

Proof. Any job $X \in N$ is not admitted at time $r(X)$. By the definition of $S_1(X)$, $X \cup (S(X) - S_1(X))$ is full-speed admissible at $r(X)$. Since X is not admitted, we have $w(X) \leq 2w(S_1(X))$.

By definition, at time $r(X)$, X plus $S_2(X) = \{J_k, \ldots, J_m\}$ are not full-speed admissible, but become full-speed admissible if J_m is removed. In other words, $w(X)$ plus the remaining work of $S_2(X)$ at time $r(X)$ is strictly greater than $(\max\{d(X), d(J_m)\} - r(X))T = |\rho'(X)|T$. Thus, we have $|\rho'(X)|T < w(X) + w(S_2(X)) \leq 2w(S_1(X)) + w(S_2(X))$. Since jobs in $S_1(X)$ or $S_2(X)$ are all in A and have deadlines in $\rho'(X)$, we have $2w(S_1(X)) + w(S_2(X)) \leq 3w(A|_{\rho'(X)})$. The lemma follows. □

Proof of Lemma 2.2. By Lemma 2.3, for each job $X \in N$, $|\rho'(X)|T < 3w(A|_{\rho'(X)})$. By Fact 2.1, N contains a subset M with all elements X having disjoint $\rho'(X)$, and $\ell T \leq 2 \sum_{X \in M} |\rho'(X)|T \leq \sum_{X \in M} 6\, w(A|_{\rho'(X)})$. For any two jobs X, X' in M, $\rho'(X)$ and $\rho'(X')$ do not overlap, and the sets $A|_{\rho'(X)}$ and $A|_{\rho'(X')}$ are also disjoint. Thus, $\sum_{X \in M} 6\, w(A|_{\rho'(X)}) \leq 6\, w(A)$. □

3 The speed function OAT makes FSA honest

We supplement FSA with a speed function OAT, which is derived from the algorithm OA (Optimal Available) [20]. As mentioned before, OA is designed for scheduling on a processor with unbounded speed and targets to complete all jobs. Given a job sequence I, we maintain an imaginary schedule of I using OA. Let OA(t) be the speed of OA at time t. Note that OA(t) may be higher than the given maximum speed T. The speed function OAT(t) is defined to be $\min\{OA(t), T\}$. Let us reiterate that OAT does not depend on how FSA works.

In this section, we first review the definition and some properties of OA. Then we present the main result that FSA(OAT) is honest. At first glance, one might worry that OAT is sometimes too slow to allow FSA to complete the jobs admitted. Nevertheless, we have a non-trivial observation that when OA is slow (using a speed at most T) in processing a job J that has been admitted by FSA, J indeed has better progress in the schedule of FSA(OAT) and doesn't require a higher speed to get completed.

Let I be a job sequence. Let $w_{\text{OA}}(J,t)$ denote the remaining work of a job J at time t in the OA schedule (we assume that $w_{\text{OA}}(J,t) = 0$ if J has not yet arrived at t). At time t, the OA algorithm schedules the earliest-deadline job J with positive remaining work at speed

$$\text{OA}(t) = \max_{t'>t} \frac{\sum_{d(J)\le t'} w_{\text{OA}}(J,t)}{t'-t}.$$

Intuitively, $\frac{\sum_{d(J)\le t'} w_{\text{OA}}(J,t)}{t'-t}$ measures the "intensity" of the interval $[t,t']$, which is a lower bound of the speed required to complete the remaining jobs with deadlines falling into the interval $[t,t']$.

We order the jobs in I in ascending order of their release times (ties are broken by job ID). Below the variable I' (or I_o) refers to a prefix of I. We often compare the OA schedule of I and the OA schedule of I'. At any time t, let $\text{OA}_I(t)$ and $\text{OA}_{I'}(t)$ denote the current speed of the OA schedule of I and I', respectively. Similarly, we define $\text{OAT}_I(t)$ or $\text{OAT}_{I'}(t)$ for OAT. The following properties can be proven based on the existing knowledge of OA [2, 20].

Fact 3.1. *At any time t, let $I' \subseteq I$ be all the jobs released on or before t. Then, for all $t' > t$, we have $\sum_{d(J)\le t'} w_{\text{OA}}(J,t) \le \int_t^{t'} OA_{I'}(x)dx$.*

Fact 3.2. *Let J be a job in I. Let $I' \subseteq I$ denote all jobs preceding J, and $I'' = I' \cup \{J\}$. Consider the two* OA *schedules for I' and I'' respectively. At any time before $r(J)$, these two schedules run at the same speed. At any time $t \ge r(J)$, $\text{OA}_{I'}(t) \le \text{OA}_{I''}(t)$. Furthermore, $\text{OA}_{I''}(t) \le \text{OA}(t)$.*

At any time t, let $w_{\text{FSA}}(J,t)$ denote the remaining work of job J at time t in the schedule given by FSA(OAT). Note that a set S of jobs is full-speed admissible at time t if and only if, for any $t' > t$,

$$\sum_{J\in S \text{ and } d(J)\le t'} w_{\text{FSA}}(J,t) \le T(t'-t).$$

First of all, we prove a weaker form of honesty for FSA(OAT), namely, whenever FSA(OAT) finds a full-speed admissible set of jobs, FSA(OAT) can complete them if no more jobs will arrive. Consider any time t_o when a job in I is released. Let I_o be the set of jobs arrived on or before t_o, and let $S_o \subseteq I_o$ be the set of jobs in the admitted list of FSA(OAT). Since FSA(OAT) has performed an admission test at time t_o, the jobs in S_o must be full-speed admissible. In the next two lemmas, we will show that if no more job arrives after I_o, the remaining work of all jobs in S_o can be completed using the speed function OAT_{I_o}, or equivalently, for any $t' > t_o$,

$$\sum_{J\in S_o \text{ and } d(J)\le t'} w_{\text{FSA}}(J,t_o) \le \int_{t_o}^{t'} \text{OAT}_{I_o}(x)dx.$$

Consider the OA schedule of I_o. Let $t_1 \ge t_o$ be the latest time this OA schedule runs at a speed higher than T (if t_1 does not exists, i.e., $\text{OA}_{I_o}(t_o) \le T$, then we set t_1 to be the time immediately before t_o). In other words, for any $t_0 \le t \le t_1$, $\text{OA}_{I_o}(t) > T$ and $\text{OAT}_{I_o}(t) = T$. And for any $t > t_1$, $\text{OA}_{I_o}(t) = \text{OAT}_{I_o}(t)$. The following two lemmas consider the remaining work of the jobs in S_o at time t_o according to whether the deadlines of these jobs are before t_1 or not.

Lemma 3.1. *For any $t_o \le t' \le t_1$,*

$$\sum_{J\in S_o \text{ and } t_o\le d(J)\le t'} w_{\text{FSA}}(J,t_0) \le T(t'-t_o) = \int_{t_0}^{t'} \text{OAT}_{I_o}(x)dx.$$

Proof. The lemma is true since S_o is full-speed admissible at t_o and $\text{OAT}_{I_o}(x) = T$ for $t_o \le x \le t'$. □

Lemma 3.2. *For any $t' > t_1$,*

$$\sum_{J\in S_o \text{ and } t_1<d(J)\le t'} w_{\text{FSA}}(J,t_o) \le \sum_{J\in I_o \text{ and } t_1<d(J)\le t'} w_{\text{OA}}(J,t_o)$$

$$\le \int_{t_1}^{t'} \text{OA}_{I_o}(x)dx = \int_{t_1}^{t'} \text{OAT}_{I_o}(x)dx.$$

The first inequality in Lemma 3.2 is based on an interesting observation (Lemma 3.3) that at time t_o, FSA(OAT) has made more progress than OA for those jobs with deadlines later than t_1. It can be easily proved by induction on I_o, using Facts 3.1 and 3.2.

Lemma 3.3. *$w_{\text{FSA}}(J,t_o) \le w_{\text{OA}}(J,t_o)$ for any $J \in S_o$ with $d(J) > t_1$.*

Proof of Lemma 3.2. The first inequality follows from Lemma 3.3, and the second inequality follows from the definition of OA and Fact 3.1. The last equality is due to the definition of t_1, which asserts that $OA_{I_o}(t) \le T$ for any $t > t_1$. □

Theorem 3.1. FSA(OAT) *is honest.*

Proof. Let J be a job that FSA(OAT) admitted perennially but the deadline of J is missed. Let $t < d(J)$ be the latest time that there is a job J' arriving at t. After running an admission test on J' by FSA(OAT), the admitted list, which must still include J, remains full-speed admissible. By Lemmas 3.1 and 3.2, no job in the admitted list should miss deadline before any new job arrives, which contradicts the assumption that J misses its deadline. □

4 Energy usage of FSA(OAT)

This section shows that the energy usage of OAT is at most $(\alpha^\alpha + \alpha^2 4^\alpha)$ times that of *OPT*, where *OPT* is the optimal schedule with maximum speed T that maximizes the throughput and minimizes the energy subject to this throughput.

Overview of the proof. Given a job sequence I, we will first partition the time line into intervals, which we call *overloaded periods* and *underloaded periods* (see below for formal definitions). Roughly speaking, a processor with maximum speed T cannot complete all the jobs whose spans fall completely in an overloaded period. Since *OPT* maximizes the throughput and there is too much work available during an overloaded period, we can show that the amount of work done by *OPT* during the overloaded periods must be at least a constant fraction of the maximum work of a speed-T processor can do. Thus, *OPT* does not use much less energy than OAT during the overloaded periods.

We then show that *OPT* completes all jobs whose spans do not fall completely in an overloaded period. We adapt the potential function in [2] (which works for the setting where the maximum speed is unbounded) to bound the energy usage of *OPT* and OAT on these jobs. Note that *OPT* may spread the work of these jobs over both the overloaded and non-overloaded periods, while OAT may accumulate these work to the non-overloaded periods, leading to higher speed and energy usage. We observe that the amount of such work accumulated is at most proportional to the length of the overloaded periods, so we use another potential function to account for such accumulation, showing that the energy usage of OAT on these jobs is bounded by the total energy usage of *OPT*.

4.1 Overloaded periods and properties of *OPT*

For any $S \subseteq I$, S is said to be *feasible* if a processor with maximum speed T can complete all jobs in S; and S is *infeasible* otherwise. Furthermore, $S \subseteq I$ is a *minimally infeasible job set* if (1) S is infeasible, and (2) for any job $J \in S$, $S - \{J\}$ is feasible. The *span* of a minimally infeasible job set S, denoted *span*(S), is the union of *span*(J) over all jobs $J \in S$. Note that *span*(S) is a single time interval.

Let $\mathcal{M}$ be the collection of all minimally infeasible job sets of I. Note that elements of $\mathcal{M}$ may overlap. The union of *span*(S) over all $S \in \mathcal{M}$ defines a number of disjoint time intervals $\lambda_1, \lambda_2 \ldots$. We call each such time interval an *overloaded period*. We call the remaining time intervals the *underloaded periods*. Let P_o and P_u be the set of overloaded periods and underloaded periods, respectively.

We divide I into two groups: $I_o = \{J \in I \mid \mathit{span}(J) \subseteq \lambda_i$ for some overloaded period $\lambda_i\}$ and $I_u = I - I_o$. That is, a job J is in I_u if its span overlaps with any underloaded period. Below we prove a property about I_u, followed by a useful property about *OPT*.

Lemma 4.1. (1) *I_u is feasible.* (2) *For any feasible job set $S \subseteq I_o$, $I_u \cup S$ is feasible.*

Proof. We first show that for any set $S \subseteq I$, if S is feasible, then for any $J \in I_u - S$, $S \cup \{J\}$ is feasible. Assume for contradiction that there is a set $S \subseteq I$ such that for some $J \in I_u - S$, S is feasible and $S \cup \{J\}$ is infeasible. Let S be the one with the smallest number of jobs. Let $J' \in S$ be a job such that $(S - \{J'\}) \cup \{J\}$ is infeasible. If no such J' is found, then $S \cup \{J\}$ is a minimally infeasible job set and J cannot be in I_u. The latter is a contradiction. If such J' exist, then $(S - \{J'\})$ is feasible and $(S - \{J'\}) \cup \{J\}$ is infeasible. It contradicts that S is the smallest set with such property.

Thus, for any set $S \subseteq I$, if S is feasible, we can add each job in I_u to S continuously, and the resulting set $S \cup I_u$ remains feasible. (1) is the case when S is an empty set. (2) is the case when S is a subset of I_o. ☐

Lemma 4.2. *OPT completes all jobs in I_u.*

Proof. Let $S \subseteq I$ be the set of jobs *OPT* completes. Let $S' = S - I_u$, which is feasible. If *OPT* does not complete some jobs in I_u, then $S' \cup I_u$ has larger total work than S, and $S' \cup I_u$ is feasible by Lemma 4.1. It contradicts the fact that *OPT* maximizes the throughput. ☐

Lemma 4.3. *Let $|P_o|$ be the total length of all overloaded periods. Then, the jobs in I_o that OPT completes have a total work at least $\frac{1}{4}T|P_o|$.*

Proof. We will show that there is a set of jobs $S_o \subseteq I_o$ such that S_o is feasible and total work of all jobs in S_o is at least $\frac{1}{4}T|P_o|$. Then, by Lemma 4.1, throughput of *OPT* is at least the total work of $S_o \cup I_u$, which is at least $\frac{1}{4}T|P_o|$ plus the total work of I_u. So *OPT* completes at least $\frac{1}{4}T|P_o|$ units of work belonging to jobs in I_o.

From the collection $\mathcal{M}$ of all minimally infeasible job sets, we can select greedily a sub-collection $N = \{S_1, S_2, \ldots, S_r\}$, ordered by their starting time, such that the union of the spans of all S_i is exactly P_o, and the span of each S_i does not overlap with other job sets in N, except S_{i-1} and S_{i+1}. Let $N_1 = \{S_i \in \mathcal{S} \mid i \text{ is odd}\}$ and $N_2 = \{S_i \in \mathcal{S} \mid i \text{ is even}\}$. The span of at least one of the above two groups has a total length at least $|P_o|/2$. W.L.O.G., let that group be N_1. Note that no two job sets in N_1 overlap.

For each $S_i \in N_1$, we remove the smallest-size job from S_i to make a feasible job set. Let S_o be the union of these feasible job sets. Then, S_o is feasible and the total work of S_o is at least T times half the total span of all S_i in T_1. Thus, the total work of S_o is at least $\frac{1}{4}T|P_o|$. □

4.2 Potential functions relating OAT and *OPT*

We are now ready to compare the energy usage of OAT and *OPT* using two potential functions. Recall that OAT gives a speed function without a schedule, while OA defines both. To ease the discussion, let us define an imaginary schedule OA_{cut} which at any time t, processes the same job as OA at the speed $\min\{\mathrm{OA}(t), T\}$. Note that OA_{cut} is running at the same speed as OAT, so they use the same amount of energy. In the following, we will focus at OA_{cut} and analyze its energy usage compared with *OPT*.

Defining the potential functions. We need some notations to define our potential functions. Let t be the current time. For any $t', t'' \geq t$, let $w_{\mathrm{OA}}(t', t'')$ be the amount of work remaining in the simulated schedule of OA, for jobs released on or before t and have deadlines in $(t', t'']$. For any interval $(t', t'']$, the *density* of the interval, denoted $den_{\mathrm{OA}}(t', t'')$, is $w_{\mathrm{OA}}(t', t'')/(t'' - t')$. We let $t_0 = t$. For $i \geq 1$, define t_i to be the earliest time greater than t_{i-1} such that $den_{\mathrm{OA}}(t_{i-1}, t_i) = \max_{t' > t_{i-1}} den_{\mathrm{OA}}(t_{i-1}, t')$. We call each of the interval $(t_i, t_{i+1}]$ a *critical interval.*

OA_{cut} schedules jobs according to OA. At any time t, let us consider the complete (future) schedule of OA, assuming no more jobs will arrive. If OA schedules a job J to run during $[t', t'']$ at speed s, then, OA_{cut} will run J during $[t', t'']$ at speed $\min\{s, T\}$. We say that OA *assigns* $(t'' - t') \cdot \min\{s, T\}$ units of work of J to OA_{cut} in $(t', t'']$. For any t_i, t_j, let $w_{i,j}$ be the amount of work OA assigns to OA_{cut} in $[t_i, t_j]$. Let $s_i = w_{i,i+1}/(t_{i+1} - t_i)$. Then, s_0 is the speed of OA_{cut} at time t, and s_i will be the speed of OA_{cut} from t_i to t_{i+1}.

Finally, we assume that when a job J arrives, *OPT* admits J if *OPT* will complete J, and discards J immediately otherwise. Let $w'_{i,j}$ be the amount of work remaining in *OPT* at time t for the admitted jobs with deadline in $(t_i, t_j]$. We define the potential function

$$\Phi(t) = \alpha \sum_{i \geq 0} s_i^{\alpha-1}(w_{i,i+1} - \alpha w'_{i,i+1}) \ .$$

We call the work for jobs in I_o the *busy work.* At any time t, let w_{busy} be the total amount of busy work OA assigns to OA_{cut}, plus the amount of busy work OA_{cut} has done up to now. We define a second potential function

$$\Gamma(t) = \alpha^2 T^{\alpha-1} w_{busy} \ .$$

LEMMA 4.4. *At any time t, let $E(t)$ be the total energy usage of* OA_{cut} *up to time t. Let $E'(t)$ be that for OPT. Let $\Phi(t)$ and $\Gamma(t)$ be the values of the potential functions calculated at time t. Then,*

$$E(t) + \Phi(t) \leq \alpha^\alpha E'(t) + \Gamma(t) \ .$$

Lemma 4.4 is true before any job is released. In the Appendix, we will show that

- Between job arrivals, the rate of change of $E(t) + \Phi(t)$ is at most that of $\alpha^\alpha E'(t) + \Gamma(t)$.
- When a job arrives, the change in Φ is at most the change in Γ.

Then we can prove by induction on time that Lemma 4.4 is true throughout the scheduling of I.

THEOREM 4.1. FSA(OAT) *is $(\alpha^\alpha + \alpha^2 4^\alpha)$-competitive on energy.*

Proof. Let t be the time equal to the latest deadline of jobs in I. At t, $\Phi(t) = 0$ and $\Gamma(t) \leq \alpha^2 T^\alpha |P_o|$, where $|P_o|$ is the total length of the overloaded periods. So the total energy usage of OA_{cut} is at most $\alpha^\alpha E' + \alpha^2 T^\alpha |P_o|$, where E' is the total energy usage of *OPT*. By Lemma 4.3, $E' \geq (T/4)^\alpha |P_o|$. Thus, energy usage of OA_{cut} is at most $(\alpha^\alpha + \alpha^2 4^\alpha)$ times that of *OPT*.

Energy usage of FSA(OAT) is the same as that of OA_{cut}, so the theorem follows. □

5 Speed relaxation

In this section we study the case that the maximum processor speed for the optimal offline algorithm is T while for the online algorithm is $(1 + \epsilon)T$. We give another FSA algorithm, denoted as FSA$'$ and show that, when coupled with a speed function $\mathrm{OAT}'(t) = \min\{\mathrm{OA}(t), (1 + \epsilon)T\}$, FSA$'$ is $(1 + 1/\epsilon)$-competitive on throughput and $(1 + \epsilon)^\alpha(\alpha^\alpha + \alpha^2 4^\alpha)$-competitive on energy usage. FSA$'$ uses a simple admission test that admits a new job if and only if the new job plus the admitted list of jobs is $(1 + \epsilon)T$-speed admissible. In other words, FSA$'$ always guarantees that after each job arrival the admitted list of jobs is full-speed admissible. Therefore, by Theorem 3.1 in Section 3, we can see that FSA$'$ is honest. The remaining of the section is devoted to prove, as in Theorem 5.1, that FSA$'$ has the claimed competitive ratio.

THEOREM 5.1. FSA$'$ *is $(1 + 1/\epsilon)$-competitive on throughput and $(1 + \epsilon)^\alpha(\alpha^\alpha + \alpha^2 4^\alpha)$-competitive.*

Competitive on throughput. We state, precisely, the criteria for admitting a job. Let $w_{\mathrm{FSA}'}(J, t)$ denote the remaining work of job J at time t in the schedule by FSA$'$. Let S denote the admitted list of jobs at time t. For any t and t' where $t < t'$, define $w[t, t'] = \sum_{J \in S \text{ and } d(J) \leq t'} w_{\mathrm{FSA}'}(J, t)$, which is the total remaining work at t for jobs in S with deadlines at most t'. When a job J arrives, if J plus the jobs in S is full-speed admissible, J is admitted. Precisely, J is admitted if and only if $\max_{t' \geq d(J)} \left\{ \frac{w[r(J),t'] + w(J)}{t' - r(J)} \right\} \leq (1+\epsilon)T$. Lemma 5.1 follows directly from fact that FSA$'$ is honest.

Lemma 5.1. *For any t and t' where $t < t'$, FSA$'$ must complete at least $w[t, t']$ work in $[t, t']$.*

Lemma 5.2. *If a job J is not admitted by FSA$'$ at $r(J)$, the total work completed by FSA$'$ in any set of disjoint intervals $\{[x_1, y_1], [x_2, y_2], \ldots, [x_k, y_k]\}$, with $r(J) \leq x_1 < x_2 < \ldots < x_k$ and $y_k \leq d(J)$ and $\sum_{i=1}^{k} (y_i - x_i) \geq w(J)/T$, is at least $\epsilon \cdot w(J)$.*

Proof. We prove by contradiction. Suppose there is a set of disjoint intervals $V = \{[x_1, y_1], [x_2, y_2], \ldots, [x_k, y_k]\}$, with $r(J) \leq x_1 < x_2 < \ldots < x_k$ and $y_k \leq d(J)$ and $\sum_{i=1}^{k} (y_i - x_i) \geq w(J)/T$, such that FSA$'$ completes less than $\epsilon \cdot w(J)$ within the intervals. Since J is not admitted, there exists a t' such that $(w[r(J), t'] + w(J))/(t' - r(J)) > T(1 + \epsilon)$, i.e., $w[r(J), t'] > T(1 + \epsilon)(t' - r(J)) - w(J)$. By Lemma 5.1, FSA$'$ must complete at least $w[r(J), t'] > T(1 + \epsilon)(t' - r(J)) - w(J)$ work in $[r(J), t']$. However, since $t' \geq d(J)$, $V \subseteq [r(J), t']$. Then the total work that FSA$'$ can complete in $[r(J), t'] - V$ is at most $T(1 + \epsilon)(t' - r(J) - w(J)/T)$. Since FSA$'$ completes less than $\epsilon \cdot w(J)$ work in V, the total work that FSA$'$ can complete in $[r(J), t']$ is at most $T(1+\epsilon)(t' - r(J) - w(J)/T) + \epsilon \cdot w(J) = T(1 + \epsilon)(t' - r(J)) - w(J)$, which is a contradiction. □

Lemma 5.3. *FSA$'$ is $(1 + 1/\epsilon)$-competitive on throughput.*

Proof. Let *OPT* denote an optimal schedule that maximizes the throughput with maximum speed T. Let N be the set of jobs that is completed in *OPT* but not admitted by FSA$'$. For each job $J \in N$, let J be scheduled in the disjoint intervals $[x_1, y_1], [x_2, y_2], \ldots, [x_k, y_k]$ in *OPT* with $r(J) \leq x_1 < x_2 < \ldots < x_k$. Then $y_k \leq d(J)$ and $\sum_{i=1}^{k} (y_i - x_i) \geq w(J)/T$. By Lemma 5.2, FSA$'$ completes at least $\epsilon \cdot w(J)$ work in these disjoint intervals. Since the sets of corresponding intervals for all jobs completed in *OPT*, in particular those jobs in N, are disjoint, the total work completed by FSA$'$, denoted by w_f, is at least $\epsilon \cdot \sum_{J \in N} w(J)$. On the other hand, the total work completed in *OPT* is at most $w_f + \sum_{J \in N} w(J) \leq w_f + w_f/\epsilon = (1 + 1/\epsilon)w_f$. Thus, the theorem follows. □

Competitive on energy usage. It is clear that the energy incurred by OAT$'$ is at most $(1 + \epsilon)^{\alpha}$ times that incurred by OAT defined in Section 3. By Theorem 4.1 in Section 4, we have proved that the energy usage incurred by OAT is at most $\alpha^{\alpha} + \alpha^2 4^{\alpha}$ times that of the optimal schedule with maximum speed at T. Thus, the energy usage by FSA$'$(OAT$'$) is at most $(1 + \epsilon)^{\alpha}(\alpha^{\alpha} + \alpha^2 4^{\alpha})$ times that of the optimal schedule with maximum speed at T. In other words, FSA$'$ is $(1 + \epsilon)^{\alpha}(\alpha^{\alpha} + \alpha^2 4^{\alpha})$-competitive on energy usage.

Appendix. Analysis of the potential functions

This appendix proves Lemma 4.4. We will show that

- At any time between job arrivals, the rate of change of $E(t) + \Phi(t)$ is at most the rate of change of $\alpha^{\alpha} E'(t) + \Gamma(t)$.
- When a job arrives, the change in Φ is at most the change in Γ.

Thus, we can prove by induction on time that Lemma 4.4 is true at any time. In the following, we focus at a time t and consider the complete (future) schedule of OA. Recall that $t_0, t_1, \ldots$ are the time calculated at time t such that $t_0 = t$ and t_i, $i \geq 1$ are the boundaries of the critical intervals. $w_{i,j}$ is the amount of work OA assigns to OA$_{cut}$ in $[t_i, t_j]$ and $w'_{i,j}$ is the amount of work remaining in *OPT* for the admitted jobs with deadline in $(t_i, t_j]$. Finally, $s_i = w_{i,i+1}/(t_{i+1} - t_i)$.

Change of potential between job arrivals Note that at time t, OA$_{cut}$ is working at speed s_0. Let s_{opt} be the speed of *OPT*. Let Φ' be the rate of change of Φ, let Γ' be that for Γ. We have the following lemma.

Lemma 5.4. *At any time t between job arrivals, $(s_0)^{\alpha} + \Phi' \leq \alpha^{\alpha}(s_{opt})^{\alpha}$, and $\Gamma' = 0$.*

Proof. At the time t, OA$_{cut}$ is processing jobs with deadlines in the first critical interval. Thus, $w_{0,1}$ is decreasing at the rate of s_0. Let k be the smallest number such that $w'_{k,k+1} > 0$. We can assume WLOG that *OPT* always executes the job with earliest deadline. Thus, $w'_{k,k+1}$ is decreasing at a rate of s_{opt}. Hence,

$$\begin{aligned}
&(s_0)^{\alpha} + \Phi' - \alpha^{\alpha}(s_{opt})^{\alpha} \\
&= s_0^{\alpha} + (-\alpha s_0^{\alpha-1} s_0 + \alpha^2 s_k^{\alpha-1} s_{opt}) - \alpha^{\alpha} s_{opt}^{\alpha} \\
&\leq (1 - \alpha)s_0^{\alpha} + \alpha^2 s_0^{\alpha-1} s_{opt} - \alpha^{\alpha} s_{opt}^{\alpha} \\
&= (1 - \alpha)z^{\alpha} + \alpha^2 z^{\alpha-1} - \alpha^{\alpha} \qquad \text{where } z = s_0/s_{opt}
\end{aligned}$$

For the last expression, it is $-\alpha^{\alpha}$ when $z = 0$, and it is $-\infty$ when $z = \infty$. If we set the derivative to 0, we have $z = \alpha$, where the expression equals 0. So, the expression is non-positive.

For Γ, we note that when no job arrives, the amount of busy work assigned to OA$_{cut}$ plus the amount of busy work done is a constant, and $\Gamma' = 0$. □

Change of potential when a job arrives Assume that a job arrives at time t. Let $\Delta\Phi$ be the change in Φ due to J and let $\Delta\Gamma$ be that for Γ.

LEMMA 5.5. *At any time t when a job arrives, $\Delta\Phi \le \Delta\Gamma$.*

The arrival of J changes the schedule of OA, and it may changes critical intervals and the work assigned to OA_{cut}. In the following, we first prove the lemma in a special case where the schedule of OA is not changed much. Then, we will prove the lemma in the general case.

Simple case. Recall that $d(J)$ and $w(J)$ are the deadline and work of J, respectively. Let $I_i = (t_i, t_{i+1}]$ be a critical interval before J arrives, such that $t_i < d(J) \le t_{i+1}$. We first consider the simple case that I_i remains a critical interval after J arrives and J only increases the density of I_i but not other intervals. Let d_i be the original density of I_i before J arrives, and d'_i be that after. We first consider the case that $d_i < d'_i \le T$.

LEMMA 5.6. *Assume $d_i < d'_i \le T$.* **(1)** *If time t is in P_u, then $\Delta\Phi \le 0$ and $\Delta\Gamma = 0$. Otherwise, time t is in an overloaded period λ,* **(2a)** *if $d(J)$ is in λ, then $\Delta\Phi \le \alpha^2 T^{\alpha-1} w(J)$ and $\Delta\Gamma = \alpha^2 T^{\alpha-1} w(J)$;* **(2b)** *if $d(J)$ is outside λ, then $\Delta\Phi \le 0$ and $\Delta\Gamma = 0$.*

Proof. **(1)** If the time t is in P_u, then J is in I_u and *OPT* admits J. Therefore,

$$(5.1)\qquad \Delta\Phi = \alpha\Big(\frac{w_{i,i+1} + w(J)}{t_{i+1} - t_i}\Big)^{\alpha-1}\Big((w_{i,i+1} + w(J)) - \alpha(w'_{i,i+1} + w(J))\Big) - \alpha\Big(\frac{w_{i,i+1}}{t_{i+1} - t_i}\Big)^{\alpha-1}\Big(w_{i,i+1} - \alpha w'_{i,i+1}\Big).$$

The above expression is non-positive by putting $q = w_{i,i+1}$, $\delta = w(J)$ and $r = w'_{i,i+1}$ to Lemma 5.7 below. The assignment of busy work in OA_{cut} is not changed, so $\Delta\Gamma = 0$.

(2a) If $d(J)$ is in the same busy period λ as time t, then *OPT* may not admit J.

$$\begin{aligned}
\Delta\Phi &\le \alpha\Big(\frac{w_{i,i+1} + w(J)}{t_{i+1} - t_i}\Big)^{\alpha-1}\big(w_{i,i+1} + w(J) - \alpha w'_{i,i+1}\big) \\
&\quad -\alpha\Big(\frac{w_{i,i+1}}{t_{i+1} - t_i}\Big)^{\alpha-1}\big(w_{i,i+1} - \alpha w'_{i,i+1}\big) \\
&\le \frac{\alpha}{(t_{i+1} - t_i)^{\alpha-1}}\big((w_{i,i+1} + w(J))^{\alpha} - w_{i,i+1}^{\alpha}\big) \\
&= \frac{\alpha}{(t_{i+1} - t_i)^{\alpha-1}} w(J)\big((w_{i,i+1} + w(J))^{\alpha-1} + \dots \\
&\quad + (w_{i,i+1})^{\alpha-1}\big) \\
&\le \alpha^2 T^{\alpha-1} w(J) \qquad \text{since } \tfrac{w_{i,i+1} + w(J)}{t_{i+1} - t_i} \le T
\end{aligned}$$

Consider the busy work originally assigned to OA_{cut} and the new busy work of J. They will be assigned to OA_{cut}. So w_{busy} increases by $w(J)$, and $\Delta\Gamma = \alpha^2 T^{\alpha-1} w(J)$.

(2b) If $d(J)$ is outside λ, then J is in I_u and *OPT* admits J. By Equation (5.1), $\Delta\Phi$ is non-positive. Since the final density of I_i is at most T, any busy work originally assigned to OA_{cut} in $(t_i, t_{i+1}]$ remains. So, $\Delta\Gamma = 0$. ☐

LEMMA 5.7. [2] *Let $q, r, \delta \ge 0$ and $\alpha \ge 1$. Then $(q+\delta)^{\alpha-1}(q + \delta - \alpha(r+\delta)) - q^{\alpha-1}(q - \alpha r) \le 0$.*

We also consider the case that J only increases the density of I_i, but the original density is already at least T.

LEMMA 5.8. *Assume $T \le d_i < d'_i$.* **(1)** *If time t is in P_u, then $\Delta\Phi \le 0$ and $\Delta\Gamma = 0$. Otherwise, time t is in a overloaded period λ,* **(2a)** *if $d(J)$ is in λ, then $\Delta\Phi \le 0$ and $\Delta\Gamma \ge 0$;* **(2b)** *if $d(J)$ is outside λ, then $\Delta\Phi = -\alpha^2 T^{\alpha-1} w(J)$ and $\Delta\Gamma \ge -\alpha^2 T^{\alpha-1} w(J)$.*

Proof. **(1)** If time t is in P_u, then J is in I_u. *OPT* admits J and $w'_{i,i+1}$ is increased by $w(J)$. Since the density of I_i is at least T, $w_{i,i+1}$ is not changed by J, and s_i remains T. Thus, $\Delta\Phi < 0$. Γ is not changed because the assignment of busy work in OA_{cut} is not changed.

(2a) Since the density of I_i is at least T, $w_{i,i+1}$ is not changed by J, and s_i remains T. $d(J)$ is in P_o, so *OPT* may or may not admit J, but in either case, $\Delta\Phi \le 0$. J is in I_o and the amount of busy work assigned to OA_{cut} may only increase. So $\Delta\Gamma \ge 0$.

(2b) Since the density of I_i is at least T, $w_{i,i+1}$ is not changed by J, and s_i remains T. $d(J)$ is outside λ, so J is in I_u and *OPT* admits J. $w'_{i,i+1}$ increases by $w(J)$, so $\Delta\Phi = -\alpha^2 T^{\alpha-1} w(J)$. Due to the increase in density of I_i, amount of busy work assigned to OA_{cut} may decrease. But the decrease in busy work is at most $w(J)$. Thus, $\Delta\Gamma \ge -\alpha^2 T^{\alpha-1} w(J)$. ☐

So, we can show that Lemma 5.5 is true in the following simple case.

LEMMA 5.9. *At any time when a job arrives, if it only increases the density of a single critical interval, then $\Delta\Phi \le \Delta\Gamma$.*

Proof. Let I_i be that critical interval, with initial density d_i and final density d'_i. If $d_i < d'_i \le T$ or $T \le d_i < d'_i$, then Lemma 5.6 and Lemma 5.8 show that $\Delta\Phi \le \Delta\Gamma$. If $d_i < T < d'_i$, we can divide J into two jobs J_1 and J_2, with same arrival time and deadline, and $w(J_1) = (T - d_i)|I_i|$, $w(J_2) = w(J) - w(J_1)$. The arrival of J can be simulated by J_1's arrival followed by J_2's arrival. By Lemma 5.6 and 5.8, we know that $\Delta\Phi \leq \Delta\Gamma$. □

General case. With Lemma 5.9, we can show that whenever a job arrives, $\Delta\Phi \leq \Delta\Gamma$.

Proof of Lemma 5.5. Assume a job J is released. Let $(t_i, t_{i+1}]$ be a critical interval before J arrives, such that $t_i < d(J) \leq t_{i+1}$. If J only increases the density of this critical interval, $\Delta\Phi \leq \Delta\Gamma$ by Lemma 5.9.

Otherwise, we can imagine the size of J as increasing from 0 to $w(J)$. For some size $x \leq w(J)$, one of the following events must occur.

- $(t_i, t_{i+1}]$ remains a critical interval and its density becomes equal to that of $(t_{i-1}, t_i]$.
- $(t_i, t_{i+1}]$ splits into a number of critical intervals $(t_i, t'_1], (t'_1, t'_2], \ldots, (t'_r, t_{i+1}]$.

We can divide J into two jobs J_1 and J_2 such that they have the same arrival time and deadline as J, and $w(J_1) = x$ and $w(J_2) = w(J) - x$. The effect of J's arrival on the potential functions is identical to the arrival of J_1, followed by the arrival of J_2.

In the first case, J_1 affects only a single critical interval, so $\Delta\Phi \leq \Delta\Gamma$ by Lemma 5.9. In the second case, we can consider the change in potential as increasing the density of I_i followed by splitting it into multiple critical intervals. The splitting of I_i does not affect the potential function values, so $\Delta\Phi \leq \Delta\Gamma$ by Lemma 5.9.

We can repeat the process until we used up all the $w(J)$ units of work of J. □

Proof of Lemma 4.4 Finally, we can prove Lemma 4.4 using Lemma 5.4 and 5.5.

Proof of Lemma 4.4. Let $t_0 = 0$ be the time before any job arrives. Obviously, the lemma is true at t_0. For $i \geq 1$, let t_i be the earliest time after t_{i-1} such that a job arrives. Assume the lemma is true at time t_{i-1}, then the lemma remains true for all time before the job arrives at t_i, and it remains true after the job arrives at t_i, by Lemma 5.4 and 5.5. Thus, by induction on time, the lemma is true at any time t. □

References

[1] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. In *Proc. STACS*, 621–633, 2006.

[2] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *Proc. FOCS*, pages 520-529, 2004.

[3] N. Bansal and K. Pruhs. Speed scaling to manage temperature. In *Proc. STACS*, pages 460–471, 2005.

[4] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. On-line scheduling in the presence of overload. In *Proc. FOCS*, pages 100–110, 1991.

[5] D.M. Brooks, P. Bose, S.E. Schuster, H. Jacobson, P.N. Kudva, A. Buyuktosunoglu, J.D. Wellman, V. Zyuban, M. Gupta, and P.W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[6] D. P. Bunde. Power-aware scheduling for makespan and flow. In *Proc. SPAA*, 2006, to appear.

[7] M. L. Dertouzos. Control robotics: the procedural control of physical processes. In *Proc. IFIP Congress*, pages 807–813, 1974.

[8] D. Grunwald, P. Levis, K. I. Farkas, C. B. Morrey, and M. Neufeld. Policies for dynamic clock scheduling. In *Proc. OSDI*, pages 73–86, 2000.

[9] S. Irani, R. K. Gupta, and S. Shukla. Algorithms for power savings. In *Proc. SODA*, pages 37–46, 2003.

[10] S. Irani and K. Pruhs. Algorithmic problems in power management. *SIGACT News*, 2005.

[11] W.C. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Transactions on Embedded Computing Systems*, 4(1):221-230, 2005.

[12] G. Koren and D. Shasha. D^{over}: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM J. Comput.*, 24(2):318–339, 1995.

[13] M. Li, B.J. Liu, and F.F. Yao. Min-energy voltage allocations for tree-structured tasks. In *Proc. COCOON*, pages 283–296, 2005.

[14] M. Li and F. Yao. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM J. Comput.*, 35(3):658–671, 2005.

[15] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. SOSP*, pages 89–102, 2001.

[16] K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best response for your erg. In *Proc. SWAT*, pages 14–25, 2004.

[17] K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. In *Proc. WAOA*, pages 307–319, 2005.

[18] H.S. Yun and J. Kim On energy-optimal voltage scheduling for fixed-priority hard real-time systems. ACM Transactions on Embedded Computing Systems, 2(3): 393-430, 2003.

[19] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proc. OSDI*, pages 13–23, 1994.

[20] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. FOCS*, pages 374–382, 1995.