

Optimizing Throughput and Energy in Online Deadline Scheduling

HO-LEUNG CHAN

University of Hong Kong

and

JOSEPH WUN-TAT CHAN

King's College London

and

TAK-WAH LAM AND LAP-KEI LEE AND KIN-SUM MAK

University of Hong Kong

and

PRUDENCE W.H. WONG

University of Liverpool

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 (the rate is believed to be a cubic function of the speed). As the speed is upper bounded, the processor may be overloaded with jobs and no scheduling algorithms can guarantee to 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. If we ignore energy usage, the best possible online algorithm is 4-competitive on throughput [Koren and Shasha 1995]. On the other hand, existing work on energy-efficient scheduling focuses on a setting where the processor speed is unbounded and the concern is on minimizing the energy to complete all jobs; $O(1)$ -competitive online algorithms with respect to energy usage have been known [Yao et al. 1995; Bansal et al. 2007a; Li et al. 2006]. 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 $O(1)$ -competitiveness on energy usage.

A preliminary version of this paper appeared in the proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2007.

The work of Tak-Wah Lam is partly supported by HKU Grant 7176104. The work of Prudence W.H. Wong is partly supported by EPSRC Grant EP/E028276/1.

Authors' addresses: Ho-Leung Chan, Tak-Wah Lam, Lap-Kei Lee, Kin-Sum Mak, Department of Computer Science, University of Hong Kong, Hong Kong, email: {hlchan, twlam, lkleee, ksmak}@cs.hku.hk; Joseph Wun-Tat Chan, Department of Computer Science, King's College London, UK, email: joseph.chan@kcl.ac.uk; Prudence W.H. Wong, Department of Computer Science, University of Liverpool, UK, email: pwong@liverpool.ac.uk

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 0000-0000/20YY/0000-0001 \$5.00

Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*; G.2.1 [Discrete Mathematics]: Combinatorics—*Combinatorial algorithms*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Power saving, speed scaling, deadline scheduling, online algorithms, competitive analysis

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., [Baruah et al. 1991; Dertouzos 1974; Koren and Shasha 1995]). 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 [1995] 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 admit a (offline) schedule that completes all jobs on time (i.e., the underloaded setting), the algorithm EDF (earliest deadline first) guarantees to complete all jobs in time and is thus 1-competitive [Dertouzos 1974].

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., [Grunwald et al. 2000; Pillai and Shin 2001; Weiser et al. 1994]). As the rate of energy usage P required to run a processor at speed s is believed to be roughly s^α where $\alpha \geq 2$ [Brooks et al. 2000], 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 [Pillai and Shin 2001], 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. [1995]. 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 online and offline algorithms aim to complete the entire job sequence, the only concern is the speed and energy usage. Yao et al. [1995] gave a simple online algorithm called AVR for determining the speed. When

coupled with EDF, AVR gives an online algorithm that 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. [2007a] to be α^α -competitive. Bansal et al. [2007a] 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. [2006] 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 $(\alpha^\alpha + \alpha^{24^\alpha})$ -competitive on energy usage. If $T = \infty$, then this algorithm is 1-competitive on throughput 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 for 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.

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 intriguing 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 constant fraction of the largest possible amount of work that can be completed for the remaining jobs.

We also apply our results on FSA and OAT 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 [Kwon and Kim 2005; Li and Yao 2005]; in particular, assuming the underloaded setting, Li and Yao [2005] 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

Table I. The performance guarantee (in terms of competitive ratios) for three different settings

	unbounded max speed	fixed max speed	fixed discrete speed levels
Throughput	1	14	14
Energy usage	$2^\alpha \alpha^\alpha$ [Yao et al. 1995] α^α [Yao et al. 1995; Bansal et al. 2007a] $2(\alpha/(\alpha-1))^\alpha e^\alpha$ [Bansal et al. 2007a]	$(\alpha^\alpha + \alpha^2 4^\alpha)$	$\Delta^\alpha (\alpha^\alpha + \alpha^2 4^\alpha) + 2$

level. A careful analysis would show that this algorithm is still 14-competitive on throughput and $(\Delta^\alpha (\alpha^\alpha + \alpha^2 4^\alpha) + 2)$ -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 $\Delta = 2$). Table I gives a summary of the competitive ratios under different speed models.

- 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 [Koren and Shasha 1995]. In this more general case, a straightforward adaptation of FSA can give an online algorithm that is $(12\psi + 2)$ -competitive on throughput and $(\alpha^\alpha + \alpha^2 (4\psi)^\alpha)$ -competitive on energy usage, where ψ is the importance ratio (i.e., the ratio of the largest possible value per unit work to the smallest possible one).
- Better throughput via resource augmentation.** Note that even if the energy concern is ignored, no online algorithm can be better than 4-competitive on throughput [Koren and Shasha 1995]. 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.

Remarks. The literature also contains results on other interesting aspects of energy efficient scheduling [Irani and Pruhs 2005]. Irani et al. [2007] extended the result on AVR [Yao et al. 1995] 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. [2008a] have studied the offline problem of minimizing the total flow time subject to a fixed amount of energy, while Albers and Fujiwara [2007] and Bansal et al. [2007b] have studied the online problem of minimizing a cost consisting of the energy usage and the total flow time. Furthermore, the offline problem of minimizing the makespan subject to a fixed amount of energy has been studied in [Bunde 2006; Pruhs et al. 2008b]. 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 [Bansal et al. 2007a].

Organization of paper. The rest of the paper is organized as follows. Section 2 shows that if FSA is coupled with a speed function that allows FSA to finish all jobs admitted, then FSA is 14-competitive. Section 3 proves that OAT is a speed function as required. Section 4 analyzes the energy usage of OAT. The last two sections discuss the results on discrete speed levels, jobs with arbitrary value, and

relaxation of the maximum speed.

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 $span(J)$, 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 and 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 admitted for scheduling, 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, \dots, J_n be the jobs currently in the admitted list, where $d(J_1) \leq d(J_2) \leq \dots \leq d(J_n)$.

- J is admitted if J together with J_1, J_2, \dots, J_n is full-speed admissible. [A set S of jobs is said to be *full-speed admissible* at a certain time if, using the maximum speed onwards, the remaining work of every job in S can be completed by its deadline.]
- Otherwise, J can still be admitted if $w(J) > 2(w(J_1) + w(J_2) + \dots + w(J_k))$ and $\{J, J_{k+1}, J_{k+2}, \dots, J_n\}$ is full-speed admissible, where k is the smallest possible integer in $[1, n]$. In this case, J_1, \dots, 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 the overdue condition is defined for completeness; we will show later that it does not occur in the algorithm $FSA(OAT)$ we are going to use.)

Note that FSA does not guarantee a job admitted at release time to be completed; the job 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.

If $\text{FSA}(f)$ is honest, then no job becomes overdue; furthermore, $\text{FSA}(f)$ is constant competitive on throughput as shown in the following theorem.

THEOREM 2.2. *If $\text{FSA}(f)$ is honest, then $\text{FSA}(f)$ is 14-competitive on throughput.*

The rest of this section is devoted to proving Theorem 2.2. In the next section, we will describe a speed function OAT and show that $\text{FSA}(\text{OAT})$ is honest. Assume that $\text{FSA}(f)$ is honest and consider any job sequence I . Without loss of generality, we assume that every job in I is full-speed admissible on its own. We define several subsets of I , namely, A , N , C and E representing jobs that are admitted, not admitted, completed and expelled. Precisely, let $A \subseteq I$ be the set of jobs that have been admitted by $\text{FSA}(f)$ at their release times, 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 $\text{FSA}(f)$ is honest, $\text{FSA}(f)$ completes exactly the jobs in C and expels all jobs in E . The competitiveness of $\text{FSA}(f)$ (i.e., Theorem 2.2) stems from the following upper bounds on E and N .

LEMMA 2.3. $w(E) \leq w(C)$.

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 $w(E) \leq w(C)$. \square

Consider the union of the spans of all jobs in N , which may cover one or more intervals. Let $\ell = |\bigcup_{X \in N} \text{span}(X)|$ 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 on ℓT .

LEMMA 2.4. $\ell T \leq 6w(A)$.

With Lemmas 2.3 and 2.4, proving Theorem 2.2 is straightforward.

PROOF OF THEOREM 2.2. 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 2.4, is at most $6w(A)$. Thus, the total amount of work completed is at most $7w(A)$, or equivalently, $7(w(C) + w(E))$. By Lemma 2.3, $w(E) \leq w(C)$ and $7(w(C) + w(E)) \leq 14w(C)$. Recall that $\text{FSA}(f)$ completes all jobs in C , attaining a throughput of $w(C)$. Thus, Theorem 2.2 follows. \square

It remains to prove Lemma 2.4, i.e., the upper bound on ℓ . Recall that $\ell = |\bigcup_{X \in N} \text{span}(X)|$. We will define, for each job $X \in N$, an interval $\text{span}^*(X)$ that encloses $\text{span}(X)$ (see Definition 2.5). Then $\ell \leq |\bigcup_{X \in N} \text{span}^*(X)|$. The way span^* is defined allows us to upper bound each $|\text{span}^*(X)|$ in terms of the jobs in A whose deadlines are in $\text{span}^*(X)$ (see Lemma 2.6).

Definition 2.5. At time $r(X)$ (i.e., when X is released), let $S(X) = \{J_1, J_2, \dots, J_n\}$ be the jobs in the admitted list, where $d(J_1) \leq d(J_2) \leq \dots \leq d(J_n)$. Because X is in N , X and $S(X)$ together are not full-speed admissible. Note that

X alone is full-speed admissible. Let $k \leq n$ be the smallest integer such that X, J_{k+1}, \dots, J_n are full-speed admissible, but $X, J_k, J_{k+1}, \dots, J_n$ are not. Furthermore, let $m \leq n$ be the smallest integer such that $X, J_k, J_{k+1}, \dots, J_m$ are not full-speed admissible. Let $s = \max\{d(X), d(J_m)\}$ and let $\text{span}^*(X) = [r(X), s]$. Denote $S_1(X) = \{J_1, \dots, J_k\}$ and $S_2(X) = \{J_k, \dots, J_m\}$. Note that each job in $S_1(X)$ or $S_2(X)$ has deadline in $\text{span}^*(X)$.

LEMMA 2.6. *For any job $X \in N$, $|\text{span}^*(X)|T < 2w(S_1(X)) + w(S_2(X)) \leq 3w(A|_{\text{span}^*(X)})$, where $A|_{\text{span}^*(X)}$ denotes the subset of jobs in A whose deadlines are in $\text{span}^*(X)$.*

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 $S_2(X) = \{J_k, \dots, J_m\}$ 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 $(\max\{d(X), d(J_m)\} - r(X))T = |\text{span}^*(X)|T$. Thus, we have $|\text{span}^*(X)|T < w(X) + w(S_2(X)) \leq 2w(S_1(X)) + w(S_2(X))$. Since jobs in $S_1(X)$ or $S_2(X)$ are all in A and have deadlines in $\text{span}^*(X)$, we have $2w(S_1(X)) + w(S_2(X)) \leq 3w(A|_{\text{span}^*(X)})$. The lemma follows. \square

By definition, $\ell \leq |\bigcup_{X \in N} \text{span}^*(X)|$. The following simple observation shows that $|\bigcup_{X \in N} \text{span}^*(X)|$ can be upper bounded by considering a subset M of N whose elements X have disjoint $\text{span}^*(X)$. Then we can easily make use of Lemma 2.6 to show that $\ell T \leq 6w(A)$.

LEMMA 2.7. *N contains a subset M such that all elements X of M have mutually disjoint $\text{span}^*(X)$, and $|\bigcup_{X \in N} \text{span}^*(X)| \leq 2 \sum_{X \in M} |\text{span}^*(X)|$.*

PROOF. Let N' be a minimal subset of N such that $\bigcup_{X \in N'} \text{span}^*(X) = \bigcup_{X \in N} \text{span}^*(X)$. I.e., N' and N define the same union of time intervals. Note that no three jobs in N' have their intervals overlapping at a common time. We can further partition N' 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 $\ell \leq |\bigcup_{X \in N'} \text{span}^*(X)| \leq 2|\bigcup_{X \in M} \text{span}^*(X)| = 2 \sum_{X \in M} |\text{span}^*(X)|$. \square

Finally we are ready to give an upper bound on ℓ in terms of the work of A and prove Lemma 2.4.

PROOF OF LEMMA 2.4. By Lemma 2.6, for each job $X \in N$, $|\text{span}^*(X)|T < 3w(A|_{\text{span}^*(X)})$. By Lemma 2.7, N contains a subset M with all elements X having disjoint $\text{span}^*(X)$, and $\ell T \leq 2 \sum_{X \in M} |\text{span}^*(X)|T \leq \sum_{X \in M} 6w(A|_{\text{span}^*(X)})$. For any two jobs X, X' in M , $\text{span}^*(X)$ and $\text{span}^*(X')$ do not overlap, and the sets $A|_{\text{span}^*(X)}$ and $A|_{\text{span}^*(X')}$ are also disjoint. Thus, $\sum_{X \in M} 6w(A|_{\text{span}^*(X)}) \leq 6w(A)$. \square

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) [Yao et al. 1995]. 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 OA and OAT do 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 does not use maximum speed all the time and may be too slow to allow FSA to complete the jobs admitted. Nevertheless, we have a non-trivial observation that when OA (and hence OAT) is using a speed below T , the jobs admitted by FSA must already have better progress in the schedule of $FSA(OAT)$ than in the schedule of OA. Note that the latter can complete all jobs. Thus, $FSA(OAT)$ does not require a higher speed to get completed.

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

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

Roughly speaking, $\frac{\sum_{J: d(J) \leq t'} w_{OA}(J, t)}{t' - t}$ measures the “density” of the interval $[t, t']$, which is a lower bound on the speed required to complete the remaining jobs with deadlines falling into $[t, t']$. (Section 4.2 will further discuss the schedule of OA calculated at time t , but those details are not needed in this section.)

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_1) 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 $OA_I(t)$ and $OA_{I'}(t)$ denote the current speed of the OA schedule of I and I' , respectively. Similarly, we define $OAT_I(t)$ or $OAT_{I'}(t)$ for OAT. The following properties of OA can be proven based on the existing knowledge of OA [Yao et al. 1995; Bansal et al. 2007a]. Intuitively, Fact 3.1 means that the speed of the OA schedule is sufficient to finish all the jobs by their deadlines, while Fact 3.2 means that OA does not decrease speed when more jobs are released.

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

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 $t \geq r(J)$, $OA_{I'}(t) \leq OA_{I''}(t)$. Furthermore, $OA_{I''}(t) \leq OA_I(t)$.*

At any time t , let $w_{FSA}(J, t)$ 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 $t' > t$,

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

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_1 when a job is released. Let I_1 be the set of jobs arrived at or before t_1 . OAT updates its admitted list at time t_1 ; let $S_1 \subseteq I_1$ be the set of jobs in the admitted list of FSA(OAT). By definition, the jobs in S_1 are full-speed admissible at t_1 . In Lemmas 3.3 and 3.5 below, we show that if no more job arrives after t_1 , the remaining work of all jobs in S_1 can be completed using the speed function OAT_{I_1} , or equivalently, for any $t' > t_1$,

$$\sum_{J \in S_1 \text{ and } d(J) \leq t'} w_{\text{FSA}}(J, t_1) \leq \int_{t_1}^{t'} \text{OAT}_{I_1}(x) dx.$$

Consider the OA schedule of I_1 . Let $t_2 \geq t_1$ be the latest time this OA schedule runs at a speed higher than T (if t_2 does not exist, then we set t_2 to be the time immediately before t_1). In other words, for any time t in $[t_1, t_2]$, $\text{OA}_{I_1}(t) > T$ and $\text{OAT}_{I_1}(t) = T$. And for any $t > t_2$, $\text{OA}_{I_1}(t) = \text{OAT}_{I_1}(t)$. Lemma 3.3 derives an upper bound on the remaining work in S_1 with deadlines at most t_2 , and Lemma 3.5 considers those with deadlines beyond t_2 .

LEMMA 3.3. *For any $t_1 \leq t' \leq t_2$,*

$$\sum_{J \in S_1 \text{ and } t_1 \leq d(J) \leq t'} w_{\text{FSA}}(J, t_1) \leq T(t' - t_1) = \int_{t_1}^{t'} \text{OAT}_{I_1}(x) dx.$$

PROOF. The lemma is true since S_1 is full-speed admissible at t_1 and $\text{OAT}_{I_1}(x) = T$ for $t_1 \leq x \leq t'$. \square

We then consider the remaining work in S_1 with deadlines beyond t_2 . This case depends on an interesting observation that FSA(OAT) has made more progress than OA for every job J in S_1 with deadline greater than t_2 .

LEMMA 3.4. *$w_{\text{FSA}}(J, t_1) \leq w_{\text{OA}}(J, t_1)$ for any $J \in S_1$ with $d(J) > t_2$.*

PROOF. Note that some jobs might have released before t_1 . We prove the lemma by induction on the number of distinct times before t_1 when jobs are released. Base case: If t_1 is indeed the first time when a job arrives, neither FSA(OAT) nor OA has processed any work at t_1 , and the lemma is trivially true.

Induction step: Suppose the lemma is true if there are $k - 1$ distinct release times before t_1 . Below we consider the case for k distinct release times before t_1 . Let r be the last release time before t_1 . Recall that I_1 and S_1 denote the set of jobs released up to t_1 and the set of admitted jobs at t_1 , respectively. Similarly, we define I_r and S_r for r . Note that $r < t_1$, $I_r \subseteq I_1$, and $S_1 \subseteq S_r \cup (I_1 - I_r)$.

Recall that at time t_1 , assuming no more jobs arrive afterwards, we define $t_2 \geq t_1$ to be the latest time such that $\text{OA}(t_2) > T$ (precisely, $\text{OA}_{I_1}(t_2) > T$). Similarly, assuming no more jobs arrive after time r , we define $r_1 \geq r$ to be the latest time such that $\text{OA}(r_1) > T$ (precisely, $\text{OA}_{I_r}(r_1) > T$). By Fact 3.2, we have $r_1 \leq t_2$.

At time t_1 , let J be a job in S_1 with $d(J) > t_2$. If J is released at time t_1 , then both FSA(OAT) and OA haven't processed J at t_1 . Thus, $w_{\text{FSA}}(J, t_1) = w_{\text{OA}}(J, t_1) = w(J)$, and the lemma follows. The non-trivial case is when J is

released at or before r . In this case, J is also in S_r and $d(J) > t_2 \geq r_1$. The induction hypothesis guarantees that $w_{\text{FSA}}(J, r) \leq w_{\text{OA}}(J, r)$.

To complete the proof, we consider whether $t_1 \leq r_1$.

- Assume that $t_1 \leq r_1$. The remaining work of J under FSA(OAT) cannot increase from time r to t_1 . Thus, $w_{\text{FSA}}(J, t_1) \leq w_{\text{FSA}}(J, r) \leq w_{\text{OA}}(J, r)$. The definition of r_1 implies that at time r , OA is committed to schedule only jobs with deadlines at most r_1 within $(r, r_1]$ (and jobs with deadlines beyond r_1 after time r_1). Yet such commitment may change when new jobs arrive at t_1 . For J , since $d(J) > t_2 \geq r_1$, OA does not schedule J during $[r, \min\{t_1, r_1\}] (= (r, t_1])$, and $w_{\text{OA}}(J, r) = w_{\text{OA}}(J, t_1)$. In conclusion, $w_{\text{FSA}}(J, t_1) \leq w_{\text{OA}}(J, t_1)$.
- Assume that $r_1 < t_1$. Let us consider the schedule during the period $(r_1, t_1]$. By definition, at any time $t \in (r_1, t_1]$, $\text{OA}(t) \leq T$ and FSA(OAT) runs at the speed $\text{OA}(t)$. On the other hand, OA and FSA(OAT) both use EDF to select jobs from I_r and S_r , respectively. Since $S_r \subseteq I_r$, we can be sure that for every job J' in S_r (including J), if $w_{\text{FSA}}(J', r_1) \leq w_{\text{OA}}(J', r_1)$, then $w_{\text{FSA}}(J', t_1) \leq w_{\text{OA}}(J', t_1)$. Therefore, with respect to J , $w_{\text{FSA}}(J, t_1) \leq w_{\text{OA}}(J, t_1)$.

Combining the two cases, the lemma follows. \square

We now present and prove Lemma 3.5.

LEMMA 3.5. *For any $t' > t_2$,*

$$\begin{aligned} \sum_{J \in S_1 \text{ and } t_2 < d(J) \leq t'} w_{\text{FSA}}(J, t_1) &\leq \sum_{J \in I_1 \text{ and } t_2 < d(J) \leq t'} w_{\text{OA}}(J, t_1) \\ &\leq \int_{t_2}^{t'} \text{OA}_{I_1}(x) dx = \int_{t_2}^{t'} \text{OAT}_{I_1}(x) dx. \end{aligned}$$

PROOF. The first inequality follows from Lemma 3.4 that at time t_1 , for any job $J \in S_1$ with $d(J) > t_2$, FSA(OAT) has made more progress than OA. I.e., $w_{\text{FSA}}(J, t_1) \leq w_{\text{OA}}(J, t_1)$.

The second inequality follows from the definition of OA and Fact 3.1. Consider the OA schedule of I_1 . By the definition of OA, in interval $[t_1, t_2]$, the processor must only work on jobs with deadline $\leq t_2$ without being idle. Therefore, $\sum_{J \in I_1 \text{ and } d(J) \leq t_2} w_{\text{OA}}(J, t_1) = \int_{t_1}^{t_2} \text{OA}_{I_1}(x) dx$. Together with Fact 3.1, for any $t' > t_2$,

$$\begin{aligned} \sum_{J \in I_1 \text{ and } t_2 < d(J) \leq t'} w_{\text{OA}}(J, t_1) &= \sum_{J \in I_1 \text{ and } d(J) \leq t'} w_{\text{OA}}(J, t_1) - \sum_{J \in I_1 \text{ and } d(J) \leq t_2} w_{\text{OA}}(J, t_1) \\ &\leq \int_{t_1}^{t'} \text{OA}_{I_1}(x) dx - \int_{t_1}^{t_2} \text{OA}_{I_1}(x) dx = \int_{t_2}^{t'} \text{OA}_{I_1}(x) dx. \end{aligned}$$

The last equality is due to the definition of t_2 , which asserts that $\text{OA}_{I_1}(t) \leq T$ for any $t > t_2$. \square

THEOREM 3.6. *FSA(OAT) is honest.*

PROOF. We prove by contradiction. 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 3.3 and 3.5, no job in the admitted list should miss deadline before any new job arrives, which contradicts the assumption that J misses its deadline. \square

4. ENERGY USAGE OF FSA(OAT)

This section shows that OAT is $(\alpha^\alpha + \alpha^{24\alpha})$ -competitive on energy when compared to any algorithm that maximizes the throughput. In the previous section, OAT refers to a speed function without a schedule, while OA defines both. In this section, we overload OAT to refer to an imaginary schedule which at any time t , processes the same job as OA at the speed $\min\{\text{OA}(t), T\}$. Note that OAT, as re-defined, is running the same speed as before, but it may not process a job till completion (as the speed is capped at T). This capped schedule is, however, very helpful in analyzing the energy usage of OAT.

First of all, let us review how Bansal et al. [2007a] make use of a potential function of unfinished work to analyze the energy usage of OA when the processor has no maximum speed. In this case, OA as well as the optimal schedule Opt can complete all jobs. At any time t , let $E_{\text{OA}}(t)$ and $E_{\text{OPT}}(t)$ be the energy used so far by OA and Opt, respectively. It was proven in [Bansal et al. 2007a] that $E_{\text{OA}}(t) + \phi_{\text{OA}}(t) \leq \alpha^\alpha E_{\text{OAT}}(t) + \phi_{\text{OPT}}(t)$, where $\phi_{\text{OA}}(t)$ [resp. $\phi_{\text{OPT}}(t)$] denotes the unfinished work of OA [resp. Opt] “weighted” by a special function to make it compatible with energy. Let t_e be the termination time, i.e., the first time when all deadlines have passed. Then $\phi_{\text{OA}}(t_e)$ and $\phi_{\text{OPT}}(t_e)$ both equal zero, and $E_{\text{OA}}(t_e) \leq \alpha^\alpha E_{\text{OPT}}(t_e)$. Thus, OA is α^α -competitive.

Below we show how to extend the above analysis to the setting where the processor has a maximum speed T . First of all, we need the following observations and notations.

- Let I be a job sequence. Let Opt be a schedule (using maximum speed T) that maximizes the throughput, while using the smallest possible energy. Opt may complete only a subset of I and never schedules the other jobs. For the sake of analysis, jobs in I that are completed by Opt are referred to as type-1 jobs and other jobs in I are called type-0 jobs. The work due to a type-0 job is called type-0 work, and similarly type-1 work for type-1 jobs. Note that the online algorithm doesn’t know such a classification.
- Neither OAT nor Opt intend to complete all jobs in I . At any time t , the unfinished work no longer means the unfinished work of all jobs. For Opt, we confine the unfinished work to type-1 jobs only. OAT follows OA to schedule every job in I using a speed capped at T . At time t , the unfinished work of OAT refers to the amount of work to be processed by OAT in accordance with the capped schedule of OA calculated at time t . See Section 4.1 for a formal definition.
- OAT attempts to schedule every job including type-0 jobs. This makes it very difficult to relate the energy and remaining work of OAT and Opt. For example, when a type-0 job J arrives, the unfinished work of Opt remains unchanged, yet J will boost the unfinished work of OAT. We resolve this problem by adding another potential function to discount the effect of type-0 jobs in the amortization analysis.

More specifically, we want to show that $E_{OAT}(t)$ and $\phi_{OAT}(t)$, the energy and the weighted unfinished work of OAT at time t , satisfy the following inequality:

$$E_{OAT}(t) + \phi_{OAT}(t) - \beta_{OAT}(t) \leq \alpha^\alpha E_{OPT}(t) + \phi_{OPT}(t), \quad (1)$$

where $\beta_{OAT}(t)$ denotes a weighted sum of the type-0 work that OAT has already processed and the unfinished type-0 work that OAT has yet to process (the weight is to be defined).

At the termination time t_e , $\phi_{OAT}(t_e)$ and $\phi_{OPT}(t_e)$ both equal zero. Yet $\beta_{OAT}(t_e)$ would have accumulated a lot of finished work. As to be shown in Corollary 4.6, an interesting finding in this section is that the work processed by OAT on type-0 jobs can be shown to be at most 4 times the work processed by Opt on I , and, in particular, $\beta_{OAT}(t_e) \leq \alpha^2 4^\alpha E_{OPT}(t_e)$. Thus, $E_{OAT}(t_e) \leq (\alpha^\alpha + \alpha^2 4^\alpha) E_{OPT}(t_e)$, and OAT is $(\alpha^\alpha + \alpha^2 4^\alpha)$ -competitive on energy.

The rest of this section is divided into two parts. Section 4.1 derives an upper bound for the type-0 work that OAT can process (i.e., $\beta_{OAT}(t_e)$). Section 4.2 is devoted to the technical details of the potential function ϕ and the proof of inequality (1). To simplify our discussion, we will omit the subscripts in the potential functions and rewrite inequality (1) as

$$E_{OAT}(t) + \phi(t) - \beta(t) \leq \alpha^\alpha E_{OPT}(t), \quad (2)$$

where $\phi(t) = \phi_{OAT}(t) - \phi_{OPT}(t)$. It is worth mentioning that ϕ and β give different weights to work; for β , work is weighted using a simple multiplier but for ϕ , the weight is more complicated and not uniform; basically, we apply the weighting function in [Bansal et al. 2007a] to our new definition of unfinished work.

4.1 Overloaded periods, properties of Opt, and upper bound of $\beta(t_e)$

We now define $\beta(t)$ formally. Let I be a job sequence. At any time t , we define $\beta(t)$ to be $\alpha^2 T^{\alpha-1}$ times the sum of the type-0 work that OAT has already processed and the unfinished type-0 work that OAT has yet to process. We make use of the fact that Opt maximizes its throughput to show that the amount of type-0 work that any algorithm (including OAT) can process is at most 4 times of the total work Opt can complete for I (or equivalently, all type-1 work). Then we show that $\beta(t_e) \leq \alpha^2 4^\alpha E_{OPT}(t_e)$.

The above results stem from a slightly complicated notion of *overloaded periods*. As to be shown later, this notion has a nice property of enclosing the span of every type-0 job. The rough idea is as follows. Let $|P_o|$ be the total length of such periods. The amount of type-0 work that OAT or any algorithm can process is at most $T|P_o|$. More interestingly, since Opt maximizes the throughput, it cannot be too lazy during the overloaded periods and we can show that the amount of work it completes is at least $\frac{1}{4}T|P_o|$. Details are given below.

For any subset of jobs $S \subseteq I$, 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 the elements of \mathcal{M} may overlap. The union of $\text{span}(S)$ over all $S \in \mathcal{M}$ defines a number of disjoint time intervals $\lambda_1, \lambda_2, \dots$. 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: $I_o = \{J \in I \mid \text{span}(J) \subseteq \lambda_i \text{ for some overloaded period } \lambda_i\}$ 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. **(i)** I_u is feasible. **(ii)** 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 a J' exists, 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 repeatedly, and the resulting set $S \cup I_u$ remains feasible.

If we consider S being an empty set, then (i) follows. If S is a subset of I_o , then (ii) follows. \square

LEMMA 4.2. Opt completes all jobs in I_u .

PROOF. Let $S \subseteq I$ be the set of jobs Opt completes. Let $S' = S - I_u$. Note that $S' \subseteq I_o$ and S' is feasible. By Lemma 4.1, $S' \cup I_u$ is feasible. Since Opt achieves the maximum throughput, S cannot have less work than $S' \cup I_u$. In other words, S must include every job in I_u . \square

COROLLARY 4.3. All jobs in I_u are type-1, and all type-0 jobs are in I_o .

LEMMA 4.4. 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 the total work of S_o is at least $\frac{1}{4}T|P_o|$. Then, by Lemma 4.1, $S_o \cup I_u$ is feasible, and the throughput of Opt is at least the work of $S_o \cup I_u$, which is at least $\frac{1}{4}T|P_o|$ plus the total work of I_u . Thus, 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 $N = \{S_1, S_2, \dots, S_r\}$, ordered by their starting time, such that the union of the span of all $S_i \in N$ 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 $N_1 = \{S_i \in N \mid i \text{ is odd}\}$ and $N_2 = \{S_i \in N \mid i \text{ is even}\}$. 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 N_1 . Thus, the total work of S_o is at least $\frac{1}{4}T|P_o|$. \square

COROLLARY 4.5. *The amount of type-0 work that OAT can process is at most $\frac{1}{4}$ times of the amount of work in I that Opt completes.*

PROOF. All type-0 jobs are in I_o , and they must be processed entirely within P_o . Thus, the maximum amount of type-0 work that OAT can process is $T|P_o|$. By Lemma 4.4, the amount of work that Opt has processed is at least $\frac{1}{4}T|P_o|$. The corollary follows. \square

COROLLARY 4.6. $\beta(t_e) \leq \alpha^2 4^\alpha E_{OPT}(t_e)$.

PROOF. Recall that $\beta(t_e)$ is defined as $\alpha^2 T^{\alpha-1}$ times of the amount of type-0 work that OAT has processed, which is at most $\alpha^2 T^\alpha |P_o|$. To complete $\frac{1}{4}T|P_o|$ units of work during the overloaded periods, the minimum energy required by Opt is $(\frac{T}{4})^\alpha |P_o|$. Thus, $E_{OPT}(t_e) \geq (\frac{T}{4})^\alpha |P_o|$, or equivalently, $|P_o| \leq (\frac{4}{T})^\alpha E_{OPT}(t_e)$. The corollary follows. \square

4.2 Potential functions and energy usage of OAT and Opt

This section proves inequality (2). First of all, we formally define the unfinished work of OAT and show how to apply the weighting function in [Bansal et al. 2007a] to such unfinished work so as to define the potential function ϕ . Note that the weighting function in [Bansal et al. 2007a] is based on the notion of critical intervals of an OA schedule [Yao et al. 1995; Bansal et al. 2007a], detailed as follows. Let t be the current time.

- For any time $t', t'' \geq t$, let $w(t', t'')$ be the unfinished work under OA that is currently available with deadlines in $(t', t'']$. Define $\rho(t', t'') = w(t', t'') / (t'' - t')$, which is also called the density of the interval.
- Furthermore, define a sequence of times as follows: Let $c_0 = t$. For $i \geq 1$, define c_i to be the earliest time after c_{i-1} such that $\rho(c_{i-1}, c_i) = \max_{t' > c_{i-1}} \rho(c_{i-1}, t')$. Each interval $(c_{i-1}, c_i]$ is called a *critical interval*. To ease our discussion, we use ρ_i to denote $\rho(c_{i-1}, c_i)$. It is useful to note that $\rho_1 \geq \rho_2 \geq \rho_3 \geq \dots$. By definition, if no more job arrives after time t , OA runs at speed ρ_i during the entire i -th critical interval $(c_{i-1}, c_i]$ and processes the jobs with deadlines in $(c_{i-1}, c_i]$ to completion.

OAT schedules every job in I in accordance with OA, but using a speed capped at T . Unlike OA, OAT does not aim to complete every job. For any currently available job J , if J 's deadline is in the critical interval $(c_{i-1}, c_i]$ and OA plans to schedule J for x time units, then OAT is committed to process $\min\{\rho_i, T\} \times x$ units of work for J . Thus, we define the unfinished work of J under OAT to be $\min\{\rho_i, T\}x$. Note that the unfinished work under OA with deadlines in a critical interval $(c_{i-1}, c_i]$ is exactly $\rho_i(c_i - c_{i-1})$, while the unfinished work under OAT with deadlines in $(c_{i-1}, c_i]$ is $\min\{\rho_i, T\}(c_i - c_{i-1})$.

The potential function $\phi(t)$ weights the unfinished work of the currently available jobs according to which critical intervals their deadlines fall into. At time t , with respect to the i -th interval $(c_{i-1}, c_i]$,

- let $w_{OAT}(i)$ be the unfinished work under OAT with deadlines in $(c_{i-1}, c_i]$;
- let $w_{OPT}(i)$ be the unfinished type-1 work under Opt with deadlines in $(c_{i-1}, c_i]$.

Adapting the work in [Bansal et al. 2007a], we apply the weight of $\alpha(\min\{\rho_i, T\})^{\alpha-1}$ to the unfinished work with deadlines in the i -th critical interval and define

$$\phi(t) = \alpha \sum_{i \geq 1} (\min\{\rho_i, T\})^{\alpha-1} (w_{OAT}(i) - \alpha w_{OPT}(i)) .$$

We are ready to state the main result of this section. Recall that $\beta(t)$ has already been defined as $\alpha^2 T^{\alpha-1}$ times the sum of the type-0 work that OAT has processed up to time t and the unfinished type-0 work under OAT.

LEMMA 4.7. *At any time t , $E_{OAT}(t) + \phi(t) - \beta(t) \leq \alpha^\alpha E_{OPT}(t)$.*

To prove Lemma 4.7, we follow the framework in [Bansal et al. 2007a] to consider how the potential functions change in two different scenarios, namely, at the arrival time of jobs and at any other time. The analysis to be presented in Section 4.2.1 and 4.2.2 is indeed a natural but tedious extension of [Bansal et al. 2007a]. For example, in many cases, β remains unchanged, and ϕ changes in the same way as in [Bansal et al. 2007a] and is sufficient to offset the different growth rate of E_{OAT} and E_{OPT} . The most noticeable case is when a type-0 job J arrives. J would drastically increase the unfinished work under OAT, but not for Opt; in other words, ϕ would go up. Nevertheless, J also boosts β , balancing out the increase of ϕ .

4.2.1 *Change of potential between job arrivals.* Let t be the current time. Assume that no job arrives at t . Let s and s_{opt} be the current speed of OAT and Opt, respectively. By definition, the rate of change of E_{OAT} and E_{OPT} are s^α and s_{opt}^α , respectively. Let ϕ' and β' be the current rate of change of ϕ and β . The following lemma gives a bound on $\phi' - \beta'$.

LEMMA 4.8. *Assume that no job arrives at time t . Then $s^\alpha + \phi' - \beta' \leq \alpha^\alpha s_{opt}^\alpha$.*

PROOF. We first observe the following bounds for ϕ' and β' .

- Recall that no job arrives at time t . Under OAT, the rate of type-0 work to be processed exactly equals the rate of unfinished type-0 work to decrease. Thus, $\beta' = 0$.
- Next, we upper bound ϕ' . At time t , OAT is going to process a job with deadline in the first critical interval (as defined at time t). In other words, $w_{OAT}(1)$ is decreasing at the rate of s . The deadline of the job to be processed by Opt is not necessarily in the first critical interval. Suppose it is in the k -th critical interval for some $k \geq 1$. Then $w_{OPT}(k)$ is decreasing at the rate of s_{opt} . Note that $s = \min\{\rho_1, T\} \geq \min\{\rho_k, T\}$, where ρ_i denotes the density of the i -th critical interval. By definition, ϕ is changing at the rate of

$$-\alpha \min\{\rho_1, T\}^{\alpha-1} s + \alpha^2 \min\{\rho_k, T\}^{\alpha-1} s_{opt} \leq -\alpha s^\alpha + \alpha^2 s^{\alpha-1} s_{opt} .$$

Therefore,

$$s^\alpha + \phi' - \alpha^\alpha s_{opt}^\alpha \leq s^\alpha - \alpha s^\alpha + \alpha^2 s^{\alpha-1} s_{opt} - \alpha^\alpha s_{opt}^\alpha .$$

Let $f(z) = (1 - \alpha)z^\alpha + \alpha^2 z^{\alpha-1} - \alpha^\alpha$. Then $f(s/s_{opt})s_{opt}^\alpha = s^\alpha - \alpha s^\alpha + \alpha^2 s^{\alpha-1} s_{opt} - \alpha^\alpha s_{opt}^\alpha$. Now we argue that $f(z) \leq 0$ for all $z \geq 0$. Note that $f(z) = -\alpha^\alpha$ if $z = 0$,

and $f(z) = -\infty$ if $z = \infty$. If we set the derivative of $f(z)$ to 0, we have $z = \alpha$ and $f(\alpha) = 0$. Therefore, $f(z) \leq 0$ for all $z \geq 0$, and $s^\alpha + \phi' - \alpha^\alpha s_{opt}^\alpha \leq 0$. \square

Lemma 4.8 implies that if Lemma 4.7 (i.e., $E_{OAT}(t) + \phi(t) - \beta(t) \leq \alpha^\alpha E_{OPT}(t)$) holds at a particular job arrival time, then it remains true up to the moment just before the next job arrives because the rate of change $E_{OAT}(t) + \phi(t) - \beta(t)$ is always upper bounded by that of $\alpha^\alpha E_{OPT}(t)$. In the next section, we consider the scenario when a job arrives.

4.2.2 Change of potential when a job arrives - a simple case. Assume that a job J is released at time t . The arrival of J immediately changes the schedule of OA and OAT, the boundaries of critical intervals, unfinished work of OAT and Opt, as well as ϕ and β . Denote the change in ϕ and β due to the arrival of J at t as $\Delta\phi$ and $\Delta\beta$, respectively. Just before J arrives, denote the critical intervals as $C_i = (c_{i-1}, c_i]$ and the unfinished work under OAT and Opt with deadlines in C_i as $w_{OAT}(i)$ and $w_{OPT}(i)$, respectively. Suppose that $d(J)$ falls into C_x . Note that J can be a type-1 job (i.e., Opt schedules J) or a type-0 job (i.e., Opt doesn't schedule J).

Simple Case. Following [Bansal et al. 2007a], we first consider a special case where the critical intervals of OA has a minimal change and then consider the general case in Section 4.2.3. Assume that immediately after J arrives, C_x remains a critical interval in the OA schedule, and J only increases the density of C_x but not other critical intervals. Suppose that the density of C_x increases from ρ to ρ' . Depending on whether ρ and ρ' exceed T , we give a different argument for proving $\Delta\phi - \Delta\beta \leq 0$.

LEMMA 4.9. (Simple Case 1) *Assume $\rho < \rho' \leq T$. (i) If J is a type-1 job then $\Delta\phi \leq 0$ and $\Delta\beta = 0$. (ii) If J is a type-0 job then $\Delta\phi \leq \alpha^2 T^{\alpha-1} w(J)$ and $\Delta\beta = \alpha^2 T^{\alpha-1} w(J)$.*

PROOF. In this case, $\rho = \frac{w_{OAT}(x)}{|C_x|}$ and $\rho' = \frac{w_{OAT}(x) + w(J)}{|C_x|}$, where $|C_x| = c_x - c_{x-1}$.

(i) Suppose that J is a type-1 job. Then $\Delta\beta = 0$ because there is no increase in type-0 work. It remains to consider ϕ . J increases the unfinished work of Opt by exactly $w(J)$. Since $\rho' \leq T$, the increase of the unfinished work of OAT (during C_x) is also $w(J)$. $\Delta\phi$ is upper bounded by

$$\begin{aligned} & \alpha \rho'^{\alpha-1} \left((w_{OAT}(x) + w(J)) - \alpha (w_{OPT}(x) + w(J)) \right) - \alpha \rho^{\alpha-1} \left(w_{OAT}(x) - \alpha w_{OPT}(x) \right) \\ &= \frac{\alpha}{|C_x|^{\alpha-1}} \left[(w_{OAT}(x) + w(J))^{\alpha-1} \left((w_{OAT}(x) + w(J)) - \alpha (w_{OPT}(x) + w(J)) \right) \right. \\ & \quad \left. - w_{OAT}(x)^{\alpha-1} \left(w_{OAT}(x) - \alpha w_{OPT}(x) \right) \right]. \end{aligned}$$

Bansal et al. [2007a] has shown that a general form of the above expression is non-positive:

$$\text{For any } q, r, \delta \geq 0 \text{ and } \alpha \geq 1, (q + \delta)^{\alpha-1} (q + \delta - \alpha(r + \delta)) - q^{\alpha-1} (q - \alpha r) \leq 0.$$

Thus, putting $q = w_{OAT}(x)$, $r = w_{OPT}(x)$, and $\delta = w(J)$, we conclude that $\Delta\phi \leq 0$.

(ii) Suppose that J is a type-0 job. J increases the type-0 unfinished work of OAT. Because $\rho' \leq T$, the increase is exactly $w(J)$, and $\Delta\beta = \alpha^2 T^{\alpha-1} w(J)$.

J , being a type-0 job, does not increase the unfinished work of Opt. $\Delta\phi$ can be bounded as follows.

$$\begin{aligned}
 \Delta\phi &\leq \alpha\rho'^{\alpha-1}(w_{OAT}(x) + w(J) - \alpha w_{OPT}(x)) - \alpha\rho^{\alpha-1}(w_{OAT}(x) - \alpha w_{OPT}(x)) \\
 &\leq \alpha\rho'^{\alpha-1}(w_{OAT}(x) + w(J)) - \alpha\rho^{\alpha-1}w_{OAT}(x) \\
 &= \frac{\alpha}{(c_x - c_{x-1})^{\alpha-1}}((w_{OAT}(x) + w(J))^\alpha - w_{OAT}(x)^\alpha) \\
 &= \frac{\alpha}{(c_x - c_{x-1})^{\alpha-1}}w(J)((w_{OAT}(x) + w(J))^{\alpha-1} + w_{OAT}(x)(w_{OAT}(x) + w(J))^{\alpha-2} \\
 &\quad + \dots + w_{OAT}(x)^{\alpha-1}) \\
 &\leq \frac{\alpha}{(c_x - c_{x-1})^{\alpha-1}}w(J)\alpha(w_{OAT}(x) + w(J))^{\alpha-1} \\
 &= \alpha^2w(J)(\rho')^{\alpha-1} \leq \alpha^2w(J)T^{\alpha-1}
 \end{aligned}$$

The second equality uses the fact that $x^\alpha - y^\alpha = (x - y)(x^{\alpha-1} + x^{\alpha-2}y + \dots + xy^{\alpha-1} + y^\alpha)$. \square

LEMMA 4.10. (Simple Case 2.) *Assume $T \leq \rho < \rho'$. (i) If J is a type-1 job then $\Delta\phi = -\alpha^2T^{\alpha-1}w(J)$ and $\Delta\beta \geq -\alpha^2T^{\alpha-1}w(J)$. (ii) If J is a type-0 job then $\Delta\phi = 0$ and $\Delta\beta \geq 0$.*

PROOF. We first consider β . Since $\rho \geq T$, OAT has committed to run at speed T in C_x before J arrives. When J arrives, OA (and hence OAT) will schedule J in C_x and must reduce the time for processing existing jobs in C_x so as to make room for J . Since OAT cannot increase the speed beyond T during C_x , the work of some existing jobs to be processed by OAT during C_x has to be reduced. Some of the work reduced may be type-0; yet the reduction of type-0 work is at most the work OAT commits to J . In summary, if J is type-1, β may decrease but by at most $\alpha^2T^{\alpha-1}w(J)$; If J is type-0, J itself contributes to the type-0 work in C_x , compensating for any possible decrease of the existing jobs, i.e., $\Delta\beta \geq 0$.

Let us consider ϕ . OAT has already committed maximum work in C_x , and J cannot increase the unfinished work of OAT in C_x . If J is a type-1 job, J increases the unfinished work of Opt by $w(J)$. Thus, ϕ decreases by exactly $\alpha \min\{\rho', T\}^{\alpha-1}\alpha w(J) = \alpha^2T^{\alpha-1}w(J)$. If J is a type-0 job, J does not increase the unfinished work of Opt either, and $\Delta\phi = 0$ \square

LEMMA 4.11. (Simple Case 3.) *Assume $\rho < T < \rho'$. Then $\Delta\phi - \Delta\beta \leq 0$.*

PROOF. In this case, we divide J into two jobs J_1 and J_2 , with the same arrival time and deadline; $w(J_1) = (T - \rho)|C_x|$ and $w(J_2) = w(J) - w(J_1)$. The arrival of J can be simulated by J_1 's arrival followed by J_2 's arrival. By Lemmas 4.9 and 4.10, we can conclude that $\Delta\phi - \Delta\beta \leq 0$. \square

4.2.3 *Change of potential when a job arrives - the general case.* In general, when a job J is released at time t , the schedule of OA may change radically. Nevertheless, as observed in [Bansal et al. 2007a], we can consider the change as a sequence of smaller changes. In particular, we can imagine the size of J as increasing from 0 to $w(J)$. Let $u \leq w(J)$ be the smallest size such that one of the following two events occurs. Below, $C_x = (c_{x-1}, c_x]$ again denotes the critical interval containing the

deadline of J ; furthermore, C_y, \dots, C_{x-1} , where $y \leq x$, are all the critical intervals before C_x with the same density as C_x .

- $C_y, C_{y+1}, \dots, C_{x-1}, C_x$ merge into a single critical interval and its density increases to that of C_{y-1} .
- Again, the boundary between $C_y, C_{y+1}, \dots, C_{x-1}, C_x$ dissolve, yet splitting occurs within C_x . I.e., $C_y, C_{y+1}, \dots, C_{x-1}, C_x$ merge and then split into two or more critical intervals $(c_{y-1}, d_{i_1}], (d_{i_1}, d_{i_2}], \dots, (d_{i_r}, c_x]$ all with the same density, where $r \geq 1$ and $c_{x-1} < d_{i_1} < d_{i_2} < \dots < d_{i_r} < c_x$.

The arrival of J can be simulated by the arrival of two jobs J_1 and J_2 , both arriving at time t and having the same deadline. J_1 has size u and J_2 has size $w(J) - u$. We can repeat this process recursively for the job J_2 , until we use up all $w(J)$ units of work of J . Then, the change in the OA schedule due to J is equivalent to a sequence of smaller changes, where each of them triggers one of the above two events. To analyze the change of potential due to J , we follow the above sequence of smaller changes and analyze the change in potential due to J_1 and then repeat this analysis recursively for J_2 .

For the first type of event, we can consider the critical intervals C_y, \dots, C_x as a single critical interval $(c_{y-1}, c_x]$ without affecting the value of the potential functions. Then, the change in potential due to J_1 can be analyzed in the same way as the simple case in Section 4.2.2. I.e., applying Lemmas 4.9–4.11 for J_1 with $(c_{y-1}, c_x]$ as the critical interval in concern, we can show that the change in potential is non-positive.

For the second type of event, we again consider the critical intervals C_y, \dots, C_x as a single critical interval $(c_{y-1}, c_x]$. This does not affect the value of the potential functions. Also, we can consider the final critical intervals $(c_{y-1}, c_{i_1}], (c_{i_1}, c_{i_2}], \dots, (c_{i_r}, c_i]$ as a single critical interval $(c_{y-1}, c_x]$ (with the same density as $(c_{i_r}, c_{i+1}]$), without affecting the value of the potential functions. Again, the change in potential due to J_1 can be analyzed in the same way as the simple case. The change in potential is non-positive.

For each of the smaller change in the sequence, the change in the potential is non-positive. So the change in the potential due to J is non-positive.

4.2.4 Competitiveness of OAT. Using the above results on the potential functions, we can prove Lemma 4.7, which states that at any time t , $E_{OAT}(t) + \phi(t) - \beta(t) \leq \alpha^\alpha E_{OPT}(t)$.

PROOF OF LEMMA 4.7. We prove the lemma by induction on time. Let $t_0 = 0$ be the time before any job arrives. Obviously, $\phi(t_0), \beta(t_0), E_{OAT}(t_0)$, and $E_{OPT}(t_0)$ all equal 0, and the lemma is true. With respect to a job sequence I , let t_1, t_2, \dots be the arrival times of the jobs. Consider any $i \geq 1$. Assume that the lemma is true at time t_{i-1} . Then by Lemma 4.8, the lemma remains true for all time before the job arrives at t_i . Furthermore, by Lemmas 4.9–4.11, when the job arrives at t_i , $\phi(t) - \beta(t)$ cannot increase and the lemma holds again. This completes the induction. \square

Let t_e be the time when the last job in the input sequence expires. The above lemma implies that $E_{OAT}(t_e) + \phi(t_e) - \beta(t_e) \leq \alpha^\alpha E_{OPT}(t_e)$. As both Opt and OAT

has no more unfinished work at t_e , $\phi = 0$. By Corollary 4.6, $\beta(t_e) \leq \alpha^2 4^\alpha E_{OPT}(t_e)$. Thus, $E_{OAT}(t_e) \leq (\alpha^\alpha + \alpha^2 4^\alpha) E_{OPT}(t_e)$. We conclude this section with the following theorem.

THEOREM 4.12. *FSA(OAT) is $(\alpha^\alpha + \alpha^2 4^\alpha)$ -competitive on energy.*

5. DISCRETE SPEED LEVELS AND JOBS WITH ARBITRARY VALUE

In this section, we show how to extend the algorithm FSA(OAT) to solve two other related scheduling problems, namely, scheduling with discrete speed levels and scheduling jobs with arbitrary value.

5.1 Discrete speed levels

Up to now, we assume a continuous setting where the processor can run arbitrarily at any speed between 0 and the maximum speed T . In the following, we consider a more realistic *discrete setting* where the processor can only run at a fixed number of discrete speed levels. Let $0 = s_0 < s_1 < \dots < s_d = T$ be the possible speeds of the processor.

We can easily extend FSA(OAT) to the discrete setting and the algorithm becomes 14-competitive on throughput and $(\Delta^\alpha(\alpha^\alpha + \alpha^2 4^\alpha) + 2)$ -competitive on energy, where Δ is the maximum ratio between two consecutive non-zero speeds. The modification is limited to the speed function. Define a speed function OAT- d : At any time t , OAT- $d(t)$ is the lowest speed level s_i such that $s_i \geq \text{OAT}(t)$. FSA(OAT- d) is the required algorithm.

Analysis. At any time, the speed of OAT- d is no less than that of OAT. Thus, OAT- d also makes FSA honest and FSA(OAT- d) is 14-competitive on throughput in the continuous and thus the discrete setting.

For energy usage, we consider any job sequence I . Let P_1 and P_2 be the collection of time intervals such that FSA(OAT- d) is running at speed s_1 and at speed at least s_2 , respectively. Let E_{P_1} and E_{P_2} be the total energy usage of FSA(OAT- d) during P_1 and P_2 , respectively. Let E_{opt} be the total energy usage of the optimal algorithm in the discrete setting. At any time during P_2 , the speed of OAT- d is at most Δ times that of OAT. Thus, E_{P_2} is at most Δ^α times the total energy used by OAT. We can adapt the analysis in Section 4.2 to show that the total energy used by OAT is also at most $(\alpha^\alpha + \alpha^2 4^\alpha)$ times of E_{opt} . Thus, $E_{P_2} \leq \Delta^\alpha(\alpha^\alpha + \alpha^2 4^\alpha) E_{opt}$.

At any time during P_1 , the speed of OAT- d may be arbitrarily greater than that of OAT. Yet a brute force approach suffices to bound E_{P_1} (since the processor is running at the lowest speed). Let $A \subseteq I$ be the set of jobs admitted by FSA(OAT- d). Let $C \subseteq A$ be the set of jobs completed, and let $B = A - C$. We know that $w(B) \leq w(C)$ (by Lemma 2.3) and hence $w(C) \geq \frac{w(A)}{2}$. Then, E_{P_1} is at most $\frac{w(A)}{s_1} \times s_1^\alpha$. The optimal algorithm completes at least $w(C)$ units of work, so E_{opt} is at least $\frac{w(C)}{s_1} \times s_1^\alpha \geq \frac{1}{2} \frac{w(A)}{s_1} \times s_1^\alpha \geq \frac{1}{2} E_{P_1}$. Thus, the total energy usage of FSA(OAT- d) is at most $(\Delta^\alpha(\alpha^\alpha + \alpha^2 4^\alpha) + 2)$ times E_{opt} . It gives the following theorem.

THEOREM 5.1. *For a processor with discrete speed levels, FSA(OAT- d) is 14-competitive on throughput and $(\Delta^\alpha(\alpha^\alpha + \alpha^2 4^\alpha) + 2)$ -competitive on energy, where Δ is the largest ratio of two consecutive (non-zero) speed levels.*

5.2 Jobs with arbitrary value

Assume that each job J is associated with a value $v(J)$ when it is released. We expect an optimal schedule to maximize the total value of jobs completed, while minimizing the energy usage subject to this total value. In this section, we show that FSA(OAT) can be adapted easily to become $(12\psi + 2)$ -competitive on total value and $(\alpha^\alpha + \alpha^2(4\psi)^\alpha)$ -competitive on energy, where ψ is the importance ratio of the jobs (i.e., the ratio of the largest possible value per unit work to the smallest possible one).

Specifically, the only change to FSA(OAT) is on the condition of expelling a job from the admitted list. We compare job value instead of job size and the new job admission rule is as follows: When a job J arrives, let J_1, \dots, J_n be the jobs in the admitted list, where $d(J_1) \leq d(J_2) \leq \dots \leq d(J_n)$. J is admitted if J together with J_1, \dots, J_n are full-speed admissible. Otherwise, J can still be admitted if for some $k \leq n$, $v(J) > 2(v(J_1) + \dots + v(J_k))$ and $\{J, J_{k+1}, \dots, J_n\}$ are full-speed admissible. In this case, J_1, \dots, J_k , where k is the smallest possible, are expelled. We follow all other parts of FSA and also run according to the speed function OAT. We call the resulting algorithm $vFSA(\text{OAT})$.

Analysis. To show that $vFSA(\text{OAT})$ is $(12\psi + 2)$ -competitive on total value, our analysis is similar to the one for throughput. For a job sequence I , let $A \subseteq I$ be the set of jobs admitted and $N = I - A$. Also, let $C \subseteq A$ be the set of jobs that are admitted perennially and $E = A - C$. Using the same argument as before, we can show that OAT makes $vFSA$ honest, i.e., completing all jobs in C . For any set of jobs S , let $v(S)$ be the total value of jobs in S . It is easy to see that $v(E) \leq v(C)$. Also, consider the union of spans of all jobs in N , which may consist of a number of disjoint intervals. Let ℓ be the total length of these intervals. We can again show that $\ell T \leq 6w(A) \leq 6v(A)$. Then, the total value obtained by the optimal algorithm is at most $v(A) + \ell T \psi \leq v(A) + 6v(A) \psi \leq (12\psi + 2)v(C)$, which is the desired result.

To analyze the energy usage of OAT, we define the overloaded periods P_o and job sets I_o and I_u in the same way as before. We follow the original analysis and show that the optimal algorithm must complete all jobs in I_u . The only change is on the lower bound of the optimal schedule Opt. We can only show that Opt achieves a total value (instead of work) of at least $\frac{1}{4}T|P_o|$ from completing jobs in I_o . Thus, the work done by Opt during the overloaded periods is at least $\frac{1}{4\psi}T|P_o|$, using at least $(\frac{1}{4\psi}T)^\alpha|P_o|$ units of energy. Then, we can define the potential functions $\phi(t)$ and $\beta(t)$ and the energy functions $E_{\text{OAT}}(t)$ and $E_{\text{OPT}}(t)$ as before, and we show that the equation $E_{\text{OAT}}(t) + \phi(t) - \beta(t) \leq \alpha^\alpha E_{\text{OPT}}(t)$ for all t remains true even if jobs have different values. At the termination time t_e (when the deadlines of all jobs have passed), $\beta \leq \alpha^2 T^\alpha |P_o| \leq \alpha^2 (4\psi)^\alpha E_{\text{OPT}}(t_e)$. Thus, $E_{\text{OAT}}(t_e) \leq (\alpha^\alpha + \alpha^2 (4\psi)^\alpha) E_{\text{OPT}}(t_e)$, which is the desired result.

Combining the above analysis, we have the following theorem.

THEOREM 5.2. *For scheduling jobs with arbitrary values, $vFSA(\text{OAT})$ is $(12\psi + 2)$ -competitive on total value and $(\alpha^\alpha + \alpha^2(4\psi)^\alpha)$ -competitive on energy, where ψ is the importance ratio.*

6. RESOURCE AUGMENTATION

In this section, we study the case that the maximum processor speed of the online algorithm is relaxed to $(1 + \epsilon)T$ for some $\epsilon > 0$, while the offline algorithm retains T . We show that a simpler algorithm FSA' , when coupled with the speed function $OAT'(t) = \min\{OA(t), (1 + \epsilon)T\}$, is $(1 + 1/\epsilon)$ -competitive on throughput and $(1 + \epsilon)^\alpha(\alpha^\alpha + \alpha^{24^\alpha})$ -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. Unlike FSA , FSA' does not expel any admitted jobs. Yet FSA' still guarantees that after every job arrival, the admitted list is full-speed (precisely, $(1 + \epsilon)T$ -speed) admissible. The analysis in Section 3 can be simplified to prove that $FSA'(OAT')$ is honest. The rest of this section is devoted to analyzing the throughput and energy usage of $FSA'(OAT')$.

Throughput Analysis. Consider any time t when a job J arrives. Let S be the set of admitted jobs just before J arrives. If $\{J\} \cup S$ is full-speed admissible, J is admitted. With respect to the schedule of $FSA'(OAT')$, define $w(J, t)$ to be the remaining work of a job J at time t . Furthermore, for any $t' > t$, define $w(t, t')$ to be the total remaining work of jobs in S with deadlines at most t' . We have the following observations:

- $FSA'(OAT')$ is honest. For any $t' > t$, $FSA'(OAT')$ completes at least $w(t, t')$ units of work during the time interval $[t, t']$.
- If J is not admitted by FSA' , then there exists $t' \geq d(J)$ such that $\frac{w(t, t') + w(J)}{t' - t} > T(1 + \epsilon)$, or equivalently, $w(t, t') > T(1 + \epsilon)(t' - t) - w(J)$.

Let Opt denote an optimal schedule that maximizes the throughput using maximum speed T . Let N be the set of jobs that is completed in Opt but not admitted by $FSA'(OAT')$. We are ready to consider how much work $FSA'(OAT')$ would perform for each job J in N .

LEMMA 6.1. *Suppose that J is not admitted by $FSA'(OAT')$. Let $\{[x_1, y_1], [x_2, y_2], \dots, [x_k, y_k]\}$ be a set of disjoint intervals within $span(J)$ such that $\sum_{i=1}^k (y_i - x_i) \geq w(J)/T$. Then the total work completed by $FSA'(OAT')$ during these intervals is at least $\epsilon \cdot w(J)$.*

PROOF. We prove by contradiction. Suppose that $V = \{[x_1, y_1], [x_2, y_2], \dots, [x_k, y_k]\}$ is a set of intervals within $span(J)$ such that $\sum_{i=1}^k (y_i - x_i) \geq w(J)/T$, and the work completed by $FSA'(OAT')$ during these intervals is less than $\epsilon \cdot w(J)$. Let $t = r(J)$, and let $w(t, t')$ be defined as above. For any $t' \geq t$, $w(t, t') \leq T(1 + \epsilon)(t' - t) - w(J) + \epsilon \cdot w(J)$ because the work completed by $FSA'(OAT')$ during V is less than $\epsilon \cdot w(J)$. Therefore, $w(t, t') + w(J) \leq T(1 + \epsilon)(t' - t)$ for all $t' \geq t$. Then J should be admitted. A contradiction occurs. \square

LEMMA 6.2. *FSA' is $(1 + 1/\epsilon)$ -competitive on throughput (against any offline algorithm with maximum speed T).*

PROOF. Let N denote the set of jobs completed by Opt but not by $FSA'(OAT')$. Let J be any job in N . Suppose Opt schedules J in the disjoint intervals $[x_1, y_1], [x_2, y_2], \dots, [x_k, y_k]$. Note that $\sum_{i=1}^k (y_i - x_i) \geq w(J)/T$. By Lemma 6.1, $FSA'(OAT')$ completes at least $\epsilon \cdot w(J)$ units of work in these disjoint intervals. Note that the

sets of corresponding intervals for all jobs completed by Opt, and in particular for those jobs in N , are disjoint. Denote the total work completed by FSA'(OAT') as w_f . Then, w_f is at least $\epsilon \cdot w(N)$. On the other hand, the total work completed by Opt is at most $w_f + w(N) \leq w_f + w_f/\epsilon = (1 + 1/\epsilon)w_f$. The lemma follows. \square

Energy analysis. 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 4.12 in Section 4, the energy incurred by OAT is at most $\alpha^\alpha + \alpha^{24\alpha}$ times that of the optimal schedule with maximum speed T . Thus, the energy usage by FSA'(OAT') is at most $(1 + \epsilon)^\alpha(\alpha^\alpha + \alpha^{24\alpha})$ times that of the optimal schedule with maximum speed T . We conclude with the following lemma.

LEMMA 6.3. FSA'(OAT') is $(1 + \epsilon)^\alpha(\alpha^\alpha + \alpha^{24\alpha})$ -competitive on energy usage (against any offline algorithm with maximum speed T).

REFERENCES

- ALBERS, S. AND FUJIWARA, H. 2007. Energy-efficient algorithms for flow time minimization. *ACM Trans. Alg.* 3, 4, 49.
- BANSAL, N., KIMBREL, T., AND PRUHS, K. 2007a. Speed scaling to manage energy and temperature. *J. ACM* 54, 1 (Mar.), 3.
- BANSAL, N., PRUHS, K., AND STEIN, C. 2007b. Speed scaling for weighted flow time. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. ACM, New York, 805–813.
- BARUAH, S. K., KOREN, G., MISHRA, B., RAGHUNATHAN, A., ROSIER, L. E., AND SHASHA, D. 1991. On-line scheduling in the presence of overload. In *Proceedings of the 32nd Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, Washington DC, 100–110.
- BROOKS, D., BOSE, P., SCHUSTER, S., JACOBSON, H., KUDVA, P., BUYUKTOSUNOGLU, A., WELLMAN, J., ZYUBAN, V., GUPTA, M., AND COOK, P. 2000. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro* 20, 6 (Nov.), 26–44.
- BUNDE, D. P. 2006. Power-aware scheduling for makespan and flow. In *Proceedings of the 18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, New York, 190–196.
- DETOUZOS, M. L. 1974. Control robotics: the procedural control of physical processes. In *Proceedings of International Federation for Information Processing (IFIP) Congress 1974*. 807–813.
- GRUNWALD, D., LEVIS, P., FARKAS, K. I., MORREY, C. B., AND NEUFELD, M. 2000. Policies for dynamic clock scheduling. In *Proceedings of the 4th Symposium on Operating System Design & Implementation (OSDI)*. 73–86.
- IRANI, S. AND PRUHS, K. 2005. Algorithmic problems in power management. *ACM SIGACT News* 36, 2, 63–76.
- IRANI, S., SHUKLA, S., AND GUPTA, R. 2007. Algorithms for power savings. *ACM Trans. Alg.* 3, 4 (Feb.), 41.
- KOREN, G. AND SHASHA, D. 1995. *D^{over}*: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM J. Comput.* 24, 2 (Apr.), 318–339.
- KWON, W. C. AND KIM, T. 2005. Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Trans. Embedded Comput. Sys.* 4, 1 (Feb.), 211–230.
- LI, M., LIU, B. J., AND YAO, F. F. 2006. Min-energy voltage allocations for tree-structured tasks. *J. Combin. Opt.* 11, 3 (May), 305–319.
- LI, M. AND YAO, F. F. 2005. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM J. Comput.* 35, 3 (Sept.), 658–671.
- PILLAI, P. AND SHIN, K. G. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. *ACM SIGOPS Operating Systems Review* 35, 5 (Dec.), 89–102.

- PRUHS, K., UTHAISOMBUT, P., AND WOEGINGER, G. 2008a. Getting the best response for your erg. *ACM Trans. Alg.* 4, 3 (June), 38.
- PRUHS, K., VAN STEE, R., AND UTHAISOMBUT, P. 2008b. Speed scaling of tasks with precedence constraints. *Theor. Comp. Sys.* 43, 1 (July), 67–80.
- WEISER, M., WELCH, B., DEMERS, A., AND SHENKER, S. 1994. Scheduling for reduced CPU energy. In *Proceedings of the 1st Symposium on Operating Systems Design & Implementation (OSDI)*. 13–23.
- YAO, F., DEMERS, A., AND SHENKER, S. 1995. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, Washington DC, 374–382.

Received Month Year; revised Month Year; accepted Month Year