Performance Guarantee for Online Deadline Scheduling in the Presence of Overload

Tak-Wah Lam Kar-Keung To
Department of Computer Science, University of Hong Kong, Hong Kong

Abstract

Earliest deadline first (EDF) is a widely-used online algorithm for scheduling jobs with deadlines in real-time systems. Yet, existing results on the performance guarantee of EDF are limited to underloaded systems [6, 12, 14]. This paper initiates the study of EDF for overloaded systems, attaining similar performance guarantees as in the underloaded setting. Specifically, we show that EDF with a simple form of admission control is optimal for scheduling on both uniprocessor and multiprocessors when moderately faster processors are available (our analysis actually admits a tradeoff between speed and extra processors). This is the first result attaining optimality under overload. Another contribution of this paper is an improved analysis of the competitiveness for weighted deadline scheduling.

1 Introduction

This paper is concerned with online algorithms for deadline scheduling. A typical example is the earliest deadline first (EDF) algorithm, which is widely used in many real-time systems (see [15] for a survey). Yet, from a theoretical viewpoint, EDF except in some simple settings has no performance guarantee, i.e., its performance cannot match or even be competitive against the offline adversary. It is indeed known that in many settings of deadline scheduling, no online algorithm has this sort of performance guarantee [2, 7]. In recent years, a plausible approach to studying performance guarantee is to allow the online scheduler to use faster processors than the offline adversary [1, 3, 5, 8, 9, 12, 14]. Intuitively, using faster processors compensates the online scheduler for the lack of future information. In particular, for deadline scheduling on multiprocessors, Phillips et al. [14] were able to show that EDF using speed-2 processors is *optimal* for hard-deadline systems. This means that if the system is underloaded, allowing the offline adversary to schedule all the jobs to meet the deadlines, then EDF can always do so with double-speed processors. This result extends the only optimality result of EDF in the literature—For hard-deadline scheduling on a single processor, EDF using a speed-1 processor is already optimal [6].

In this paper we study deadline scheduling for overloaded systems, in which the offline adversary may not be able to schedule all the jobs, and job deadlines are firm in the sense that there is no credit to completing a job after its deadline. Our aim is again to attain "optimality" with the general meaning of matching the total work of jobs completed by the adversary. Scheduling under overload is more difficult as the online algorithm has to be smart in selecting the right jobs to schedule. Even for the single processor setting, no online algorithm using a speed-1 or faster processor is known to be optimal, let alone the multiprocessor setting. In fact, it has been shown that if the processor speed is less than 1.5, any algorithm is not optimal [13]. This result should be contrasted with the fact that in the underloaded setting, EDF using a speed-1 processor is already optimal [6]. In this paper we resolve in the affirmative the following open questions about deadline scheduling under overload:

- For scheduling on a single processor, can EDF (or any other online algorithm) using a faster (say, speed-2) processor be *optimal*?
- For scheduling on multiprocessors, is EDF a good policy? Are speed-O(1) processors sufficient to guarantee optimality?
- Can we reduce the speed requirement by using more processors? Such a tradeoff result is known in the underloaded setting; e.g., let m be the number of processors available to the

offline adversary, then EDF using $2m$ speed-1.5 processors is also optimal [12].

- Suppose that jobs have arbitrary values (or weights) and the aim is to maximize the total value of the jobs meeting the deadlines. The scheduling becomes more complicated. Can we exploit faster processors to ensure that the total value attained is competitive or even optimal with respect to the offline adversary?

Problem and definitions: We are given a pool of $m \geq 1$ processors and a stream of jobs which are released at arbitrary times, with varying work (processing time) requirements and deadlines. Every job is sequential in nature and is processed by at most one processor at a time. Preemption is allowed. For a hard deadline (or equivalently, an underloaded) system, we aim at obtaining a schedule in which all jobs are completed before their deadlines, assuming that can be done by the offline adversary. In general, the system may be overloaded, i.e., even the offline adversary may not be able to meet all the deadlines. In this case, our aim is to maximize the total work of jobs completed before the deadlines. There is no credit to completing a job after its deadline. An online algorithm is said to be *optimal* if it can match the total work of jobs completed by the offline adversary. Note that an online algorithm that is optimal for overloaded systems is also optimal for underloaded systems. We consider settings in which the on-line algorithm is equiped with processors that are faster than those available to the off-linc adversary. A processor is speed-s if it can process s times the work that can be processed by a processor of the off-line adversay in the same amount of time.

EDF with admission control: The way EDF schedules jobs on $m \geq 1$ processors is as follows: Whenever a job is released or completed, EDF examines the remaining jobs to be completed. If there are at most m such jobs, each job is scheduled to run in one processor; otherwisc, EDF chooses m jobs with earliest deadlines for execution. When the system is overload, EDF may be too aggressive in preempting jobs; thus, in practice, EDF is often supplemented with some kind of admission control. We consider the following simple form of admission control (Figure 1). At release, every job has to go through a feasibility test in order to get admitted for EDF scheduling. The test simply checks whether the new job together with the previously admitted jobs can all be completed before their deadlines using a EDF schedule.[1] We call this algorithm EDF-AC. Our first result on scheduling under overload is that for a single processor system, EDF-AC using speed-2 processor is optimal. (We can easily show that EDF-AC cannot attain optimality if the speed increase is less than a factor of two.) When we are scheduling with $m \geq 2$ processors, we find that speed-3 processors suffice to guarantee EDF-AC to be optimal, no matter how big m is. To our knowledge, this is the first result attaining optimality for scheduling under overload.

Tradeoff between speed and processor: Other than using faster processors, another way to facilitate the online scheduler is to use more processors. As jobs are assumed to be sequential in nature, just increasing the number of processors by even a constant factor cannot lead to optimality. Nevertheless, we find that the speed requirement for EDF-AC can be reduced if more processors are available. In particular, we show that EDF-AC is optimal when given $M > m$ speed-$(1 + \frac{(\sqrt{m}+1)^2}{M+1})$ processors. For example, when $m = 4$, EDF-AC can attain optimality if using four speed-3 processors, or five speed-2.5 processors, or seventeen speed-1.5 processors.

Scheduling jobs with values: In general, the credit awarded to complete a job before its deadline may not be related to its processing time. Jobs may be given arbitrary values, and the objective becomes to maximize the total value of the jobs completed before their deadlines. For this more general setting, there are several online algorithms in the literature. These algorithms are not optimal, though. Define the value density of a job to be the ratio of its value to its required work. An online algorithm A is said to be c-competitive for some $c > 0$ if, for any job sequence, the total value of jobs completed by the offline adversary is at most

[1] For implementation purpose, it suffices to check jobs admitted and not completed.

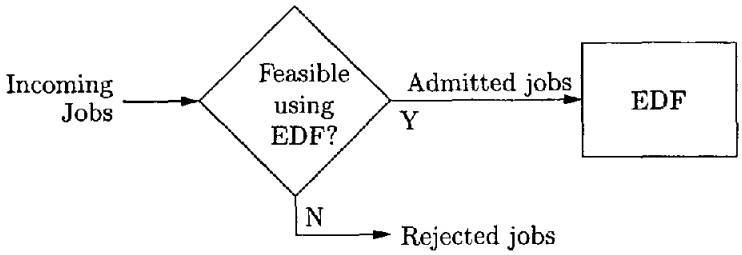

Figure 1: **The scheduling of EDF-AC.** A job is scheduled using EDF only if it passes a feasibility test.

c times of that of A. [4] In the single processor setting, Koren and Shasha [10] generalized the work of Koren et al. [2] to give an $(1+\sqrt{k})^2$-competitive algorithm, where jobs are assumed to have value density in the range $[1, k]$. Kalyanasundaram and Pruhs [9] gave another algorithm called SLACKER, which is $O(1)$-competitive when a faster processor is used. Specifically, for any $\delta > 0$, SLACKER is $(1+\delta^{-1})(1+\delta^{-1/2})(1+\delta^{-1/2}+\delta^{-1})$-competitive when given a speed-$(1+2\delta)$ processor. This paper gives a simpler analysis of SLACKER, showing that it is $(1+2\delta^{-1}+4\delta^{-2})$-competitive. For example, given a speed-2 processor, our analysis reveals that SLACKER is actually 21-competitive instead of 31.9-competitive.

In the multiprocessor setting, Koren et al. [11] gave a $(1+m(k^{1/\psi}-1))$-competitive algorithm[2], where $\psi = m\ln k/2(\ln k+1)$; yet no result has been heard on using faster processors to improve the competitiveness. In fact, SLACKER admits a natural extension to the multiprocessor setting, but it is non-trivial to extend the analysis of SLACKER in [9]. Based on our new analytical tool, we show that the multiprocessor version of SLACKER is $(1+2\delta^{-1}+4\delta^{-2})$-competitive when it is given speed-$(1+2\delta)$ processors. Table 1 gives a summary of these results.

Note that none of the above algorithms achieves optimality. We can actually show that EDF-AC achieves optimality even if jobs have different values. For the single processor setting, it suffices to use a speed-$(k+1)$ processor to achieve optimality; for the multiprocessor setting, speed-$(k+2)$ processors are required. Moreover, a simple adaptation of EDF-AC reduces the speed factor to $O(\log k)$. The idea is to divide the m processors into $\lceil \log k \rceil$ groups such that group i handles jobs with density in the range $[k^{(i-1)/\lceil \log k \rceil}, k^{i/\lceil \log k \rceil}]$.

Organization of the paper: The remainder of the paper is organized as follows. Section 2 shows the optimality of EDF-AC in the single processor setting. It serves as a warm-up to Section 3, which extends this result to the multiprocessor setting. In Section 4, we adapt EDF-AC defined in Section 3 to reduce the speed factor to $O(\log k)$. Section 5 discusses how to use extra processors to reduce the speed requirement. Finally, Section 6 presents the multiprocessor extension of SLACKER, which schedules jobs with different value densities. Due to space limitation, details of the optimality of EDF-AC for scheduling jobs with different value densities are left in the full paper.

In the rest of the paper, whenever we refer to a sequence of jobs, we assume that the first job is released at time 0. The release time, work request and deadline of a job J are denoted by $r(J)$, $p(J)$ and $d(J)$ respectively. We measure $p(J)$ in terms of the time required to process J on a speed-1 processor. We only consider jobs satisfying $p(J) \geq d(J) - r(J)$. Recall that completing a job after its deadline gives no credit. Below, whenever we say a job is completed, it is meant that the job is completed before its deadline.

2 Optimality of EDF-AC for uni-processor scheduling

This section shows that for scheduling on a single processor, increasing the processor speed by a factor of two makes EDF-AC optimal, i.e., matching the total work of jobs completed by the offline

[2] $1+m(k^{1/\psi}-1)$ approaches $2\ln k+3$ when m is large.

Processor speed	1	$1+2\delta$ for any $\delta > 0$
Single processor	$(1+\sqrt{k})^2$ [2]	$(1+\delta^{-1})(1+\delta^{-\frac{1}{2}})(1+\delta^{-\frac{1}{2}}+\delta^{-1})$ [9] $1+2\delta^{-1}+4\delta^{-2}$ [*]
Multiprocessor	$1+m(k^{1/\psi}-1)$ [11]	$1+2\delta^{-1}+4\delta^{-2}$ [*]

Table 1: Competitive ratios of algorithms for scheduling jobs with arbitrary values. In the result of [11], $\psi = m\ln k/2(\ln k+1)$. Results in this paper are denoted with asterisks.

adversary.

Notation: For any sequence L of jobs, let $\mathcal{E}_L$ be the subset of jobs in L that can be completed by EDF-AC (using a speed-2 processor), and $\mathcal{O}_L$ for the offline adversary (using a speed-1 processor); let $\|L\|$ be the total work of the jobs in L.

THEOREM 2.1. *For any sequence L of jobs, $\|\mathcal{E}_L\| \geq \|\mathcal{O}_L\|$.*

We prove Theorem 2.1 by contradiction. Suppose there exists a job sequence L such that $\|\mathcal{O}_L\| > \|\mathcal{E}_L\|$. Let us consider L to be the job sequence containing fewest jobs.

If jobs in $\mathcal{E}_L$ are scheduled to run in two or more continuous periods, we can immediately derive a contradiction since one of such periods contains a smaller job sequence violating Theorem 2.1. Below, we assume that all jobs in $\mathcal{E}_L$ are scheduled in one continuous period, say, of length ℓ. The contradiction stems from a property of the jobs admitted by EDF-AC, which is stated in the Lemma 2.1. Roughly speaking, any *late-dead* job (with deadline after 2ℓ), if present in L, must be admitted into $\mathcal{E}_L$, and would not affect the admittance of any *early-dead* jobs (with deadline at or before 2ℓ) released subsequently. The latter means that even if we remove all late-dead jobs from L, we cannot schedule more early-dead jobs.

LEMMA 2.1. *Let L_e and L_t be the sets of early-dead and late-dead jobs in L, respectively. Then $L_t \subseteq \mathcal{E}_L$ and $\mathcal{E}_{L_e} = \mathcal{E}_L - L_t$.*

With Lemma 2.1, we can prove Theorem 2.1 by deriving a contradiction as follows: Recall that jobs in $\mathcal{E}_L$ are scheduled in one continuous period of length ℓ. Since $\|\mathcal{O}_L\| > \|\mathcal{E}_L\|$, the offline adversary using a speed-1 processor takes more than 2ℓ time to complete $\mathcal{O}_L$, and some jobs in $\mathcal{O}_L$ have deadlines after 2ℓ. In other words, L contains some late jobs. By Lemma 2.1, $\|\mathcal{E}_{L_e}\| = \|\mathcal{E}_L\| - \|L_t\| < \|\mathcal{O}_L\| - \|L_t\|$. Note that $\|\mathcal{O}_{L_e}\| \geq \|\mathcal{O}_L - L_t\| \geq \|\mathcal{O}_L\| - \|L_t\|$. In summary, L_e contains fewer jobs than L, yet $\|\mathcal{E}_{L_e}\| < \|\mathcal{O}_{L_e}\|$; this contradicts the definition of L. The proof of Theorem 2.1 is completed.

Proof. [Lemma 2.1] To see $L_t \subseteq \mathcal{E}_L$, it suffices to observe that it is always feasible to schedule a late-dead job J within the period $[\ell, d(J)]$ since $\ell \leq d(J)/2$ and the processing time of J on a speed-2 processor is $p(J)/2 \leq d(J)/2$. Since EDF can complete any feasible job sets [6], this guarantees that J always passes the feasibility test of EDF-AC.

We prove $\mathcal{E}_{L_e} = \mathcal{E}_L - L_t$ by contradiction. Suppose the equality does not hold. Then $\mathcal{E}_{L_e}$ and $\mathcal{E}_L$ do not contain the same set of early-dead jobs. Let $t \leq \ell$ be the release time of the first early-dead job J on which $\mathcal{E}_{L_e}$ and $\mathcal{E}_L$ disagree. Denote by $(\mathcal{E}_L)^t$ the set of jobs in $\mathcal{E}_L$ released before t, and similarly for $(\mathcal{E}_{L_e})^t$. Note that $(\mathcal{E}_L)^t$ comprises all the jobs in $(\mathcal{E}_{L_e})^t$ plus possibly some late-dead jobs. At time t, the only possible scenario for $\mathcal{E}_{L_e}$ and $\mathcal{E}_L$ to disagree on J is that J is admitted into $\mathcal{E}_{L_e}$ but rejected from $\mathcal{E}_L$. That means, $(\mathcal{E}_{L_e})^t \cup \{J\}$ can be completed using a EDF schedule, but $(\mathcal{E}_L)^t \cup \{J\}$ cannot be. This can only happen when admitting J causes some late-dead job $J' \in (\mathcal{E}_L)^t$ to miss its deadline. By definition of ℓ, all jobs in $\mathcal{E}_L$ including J' can be completed before time ℓ. To cause J' to miss the deadline which is after 2ℓ, J must request more than ℓ units of time on a speed-2 processor, or equivalently, $p(J) > 2\ell$. This is impossible because J is an early-dead job and $p(J) \leq d(J) \leq 2\ell$. Thus,

J cannot exist and $\mathcal{E}_{L_e} = \mathcal{E}_L - L_t$. □

3 Optimality of EDF-AC for multiprocessor scheduling

We extend the result of Section 2 to the multiprocessor setting. Specifically, we show that for deadline scheduling on $m \geq 2$ processors, EDF-AC using speed-3 processors is optimal.

Consider any sequence L of jobs. Denote by $\mathcal{E}_L$ the set of jobs in L completed by EDF-AC using m speed-3 processors. With respect to the schedule of $\mathcal{E}_L$, a period is said to be *busy* if all the m processors are working throughout this period. Similar to Section 2, the core of the proof of optimality is on analyzing the case when the schedule of $\mathcal{E}_L$ contains only one busy period, say, $[0, \ell]$, and all jobs in L are released within this period. We say that a job in L is *early-dead* if its deadline is at or before 3ℓ, and *late-dead* otherwise. We will show that the early-dead and late-dead jobs in L satisfy the same properties as in the single-processor setting (see Lemma 3.2 below); the proof is slightly more complicated due to the multiprocessor setting. First of all, we make a more basic observation, which reveals the need of speed-3 processors.

LEMMA 3.1. *Assume that the schedule of $\mathcal{E}_L$ contains only one busy period. Consider any job $J \in L$. Let X be the set of jobs in $\mathcal{E}_L$ released before J. Let $X' = X \cup \{J\}$ and let $X'_e \subseteq X \cup \{J\}$ comprise all the early-dead jobs. Then, using an EDF schedule, X' can be completed if and only if X'_e can be completed.*

Proof. The "only if" direction is obvious. It remains to prove the "if" direction. For the sake of contradiction, suppose that using an EDF schedule, X'_e can be completed but not X'. Among the jobs in X' not meeting the deadlines, let J_0 be the one with the earliest deadline. Since X'_e can be completed, J_0 must be a late-dead job, i.e., $d(J_0) > 3\ell$, or equivalently, $\ell < d(J_0)/3$. On the other hand, as X, a subset of $\mathcal{E}_L$, can be completed using an EDF schedule, we have $d(J_0) \geq d(J)$.

Let us have a close look of the EDF schedules of X and $X' = X \cup \{J\}$. Recall that the EDF schedule of $\mathcal{E}_L$ contains only one busy period. Thus, the schedule of X also contains a single busy period, say, ending at h. A basic property of EDF is that at any particular time, the schedule of $X \cup \{J\}$ uses at least as many processors as the schedule of X. Therefore, the EDF schedule of $X \cup \{J\}$ contains a busy period ending at $\hat{h} \geq h$. Moreover, the schedule of $X \cup \{J\}$ outperforms the schedule of X by at least $3(\hat{h} - h)$ units of work. Since the EDF schedule of X completes all jobs in X, we have $p(J) \geq 3(\hat{h} - h)$. Thus, $\hat{h} \leq h + p(J)/3 \leq \ell + p(J)/3 < d(J_0)/3 + d(J)/3$. Using EDF to schedule $X \cup \{J\}$, we will complete J_0 if $d(J_0) \geq \hat{h} + p(J_0)/3$. The latter is true because $\hat{h} + p(J_0)/3 < d(J_0)/3 + d(J)/3 + d(J_0)/3 \leq d(J_0)$. Thus, J_0 should have met its deadline in the EDF schedule of X'. A contradiction occurs. □

With Lemma 3.1, we can easily show the key properties of the late-dead jobs in L.

LEMMA 3.2. *Let L_e and L_t be the sets of the early-dead jobs and late-dead jobs in L. Then $L_t \subseteq \mathcal{E}_L$ and $\mathcal{E}_{L_e} = \mathcal{E}_L - L_t$.*

Proof. Since EDF schedules jobs in the order of their deadlines, admitting a late-dead job J does not cause any early-dead jobs previously admitted to miss the deadline. By Lemma 3.1, J is always admitted by EDF-AC into $\mathcal{E}_L$. Thus, $L_t \subseteq \mathcal{E}_L$.

Suppose $\mathcal{E}_{L_e} \neq \mathcal{E}_L - L_t$ and consider the first job J in L_e which is scheduled differently when EDF-AC schedules L and L_e. J must be admitted when EDF-AC schedules L_e and rejected when EDF-AC schedules L. This implies that, when using EDF-AC to schedule L, we should find that J together with the early-dead jobs admitted before J can be completed using an EDF schedule. By Lemma 3.1, J should be admitted when using EDF-AC to schedule L. A contradiction occurs. □

It remains to prove a theorem, in the spirit of Theorem 2.1, to show that for scheduling on $m \geq 2$ processors, EDF-AC is optimal when given speed-3 processors. Unlike Theorem 2.1, the case when the schedule of EDF-AC contains more than one busy periods is complicated to analyze. The problem is that the execution of a job can span more than one busy period, making it difficult to

split L for contradiction purpose. To deal with this, we prove a stronger version of Theorem 2.1; roughly speaking, $\|\mathcal{E}_L\|$ not only matches $\|\mathcal{O}_L\|$, but is in excess by a significant amount: Suppose $\mathcal{O}$ is any subset of L that is feasible for scheduling. Define a function $f(L,\mathcal{O})$ to be the portion of the work of the jobs in $\mathcal{E}_L - \mathcal{O}$ scheduled by EDF-AC after its last busy period. We show that $\|\mathcal{E}_L\|$ is in excess of $\|\mathcal{O}\|$ by at least $f(L,\mathcal{O})$. Technically speaking, this allows us to, in the case when there are more than one busy periods, split L into two parts both containing those jobs spanning two busy periods. Intuitively, the overlapping is compensated by the increased $f(L,\mathcal{O})$ after the split. Details are as follows:

THEOREM 3.1. *For any job set L, $\|\mathcal{E}_L\| - f(L,\mathcal{O}) \geq \|\mathcal{O}\|$ for all $\mathcal{O} \subseteq L$ that is feasible for the offline adversary.*

Proof. Suppose, for the sake of contradiction, there exists a job sequence L such that the theorem is violated. That is, there exists $\mathcal{O} \subseteq L$ that causes $\|\mathcal{E}_L\| - f(L,\mathcal{O}) < \|\mathcal{O}\|$. Without loss of generality, we consider L to be the job sequence containing the fewest jobs.

Consider the schedule of $\mathcal{E}_L$ defined by EDF-AC. Some jobs may be released outside a busy period. For each of such jobs, we can delay its release time until the next busy period while reducing the required work. This results in another sequence of jobs with the same busy periods and the same set of jobs admitted by EDF-AC. $\|\mathcal{E}_L\|$ is reduced by exactly the total amount of work reduced, while $\|\mathcal{O}\|$ is reduced by at most that amount. On the other hand $f(L,\mathcal{O})$ is not affected. As a result, the modified L and $\mathcal{O}$ still cause the theorem to be violated. Thus, without loss of generality, we can assume that jobs are always released when EDF-AC is busy.

Assume $\mathcal{E}_L$ occupies one single period, say, of length ℓ. Then the offline adversary using speed-1 processors must take more than 3ℓ time to complete $\mathcal{O}$. L thus contains some late-dead jobs. Define the follow subsets of jobs: $L_e \subset L$ and $\mathcal{O}_e \subseteq \mathcal{O}$ contain early-dead jobs; $L_l \subseteq L$ contains late-dead jobs; and L_1 and L_2, subsets of L_l, contain jobs in $\mathcal{O}$ and not in $\mathcal{O}$, respectively. By Lemma 3.2, $\|\mathcal{E}_{L_e}\| = \|\mathcal{E}_L\| - \|L_l\|$. It is clear that $\|\mathcal{O}_e\| = \|\mathcal{O}\| - \|L_1\|$. Since the scheduling of jobs in $\mathcal{E}_{L_e}$ is not affected by the removal of the late-dead jobs, $f(L_e,\mathcal{O}_e) \geq f(L,\mathcal{O}) - \|L_2\|$. Therefore, $\|\mathcal{E}_{L_e}\| - f(L_e,\mathcal{O}_e)$ is smaller than $\|\mathcal{O}_e\|$. In summary, $\mathcal{O}_e$ causes L_e to violate the theorem, contradicting the definition of L.

Assume $\mathcal{E}_L$ are scheduled to run in two or more continuous periods. We construct two smaller job sets L_a and L_b as follows. Let t denote the starting time of the second busy period. L_a contains all jobs in L that are released earlier than t. L_b contains all other jobs plus some new jobs derived from L_a as follows. For each job J in $\mathcal{E}_L \cap L_a$, if J has not yet completed at time t by EDF-AC, leaving $w > 0$ units of work, we add to L_b a new job with release time t, deadline $d(J)$, and required work w. Denote by N the set of new jobs added to L_b. It is easy to see that $\mathcal{E}_{L_a} = \mathcal{E}_L \cap L_a$, while $\mathcal{E}_{L_b}$ contains exactly those jobs in L_b that are either in $\mathcal{E}_L$ or derived from jobs in $\mathcal{E}_L$. The sum of $\|\mathcal{E}_{L_a}\|$ and $\|\mathcal{E}_{L_b}\|$ is exactly the sum of $\|\mathcal{E}_L\|$ and $\|N\|$.

It remains to show that L_a or L_b must violate the theorem. Consider the following sets: $\mathcal{O}_a \subseteq L_a$ contains those jobs also in $\mathcal{O}$, and $\mathcal{O}_b \subseteq L_b$ contains those jobs also in $\mathcal{O}$ or derived from jobs in $\mathcal{O}$. The sum of $\|\mathcal{O}_a\|$ and $\|\mathcal{O}_b\|$ is exactly the sum of $\|\mathcal{O}\|$ and $\|N \cap \mathcal{O}_b\|$. Note that $f(L,\mathcal{O})$ is exactly $f(L_b,\mathcal{O}_b)$, while $f(L_a,\mathcal{O}_a)$ is no less than $\|N - \mathcal{O}_b\|$. We conclude that $\|\mathcal{E}_{L_a}\| + \|\mathcal{E}_{L_b}\| - f(L_a,\mathcal{O}_a) - f(L_b,\mathcal{O}_b)$ is smaller than $\|\mathcal{O}_a\| + \|\mathcal{O}_b\|$, so either $\mathcal{O}_a$ causes L_a to violate the theorem, or $\mathcal{O}_b$ causes L_b to violate the theorem. Since both L_a and L_b contain less jobs than L, this again contradicts the definition of L. □

The analysis can be extended to cover the case when jobs have arbitrary values, and the objective is to maximize the value obtained by completing jobs within deadlines. We have the following theorem, which shows that EDF-AC using speed-$(2+k)$ processors is optimal in this case. Due to space limitation, the proof is left to the full paper.

THEOREM 3.2. *Suppose* EDF-AC *using speed-$(2+k)$ processors is used for any job set L that contains jobs of value densities in $[1,k]$. Let $\mathcal{E}_L$ be the set of*

jobs completed by EDF-AC. *Then* $\|\mathcal{E}_L\| - f(L, \mathcal{O}) \geq \|\mathcal{O}\|$ *for all* $\mathcal{O} \subseteq L$ *that is feasible for the offline adversary.*

4 Improving EDF-AC with value density groups

When k is large, EDF-AC does not give a good performance guarantee since EDF-AC is optimal only when using a lot of extra speed, namely when using speed-$(2+k)$ processors. In this section, we use EDF-AC as a black box and derive another algorithm λ-EDF-AC for any integer $\lambda > 1$. By grouping jobs with vastly different value densities into different group of processors, the algorithm needs only λm speed-$(2 + k^{1/\lambda})$ processors to achieve optimality (Theorem 4.1). Putting $\lambda = \lceil \log_2 k \rceil$, this implies $\lceil \log_2 k \rceil$-EDF-AC is optimal using $\lceil \log_2 k \rceil m$ speed-4 processors. Using time sharing, we can simulate $\lceil \log_2 k \rceil$-EDF-AC using m speed-$(4\lceil \log_2 k \rceil)$ processors and thus obtain an optimal algorithm under such situation. This provides a much better guarantee than that provided by EDF-AC.

The algorithm is defined as follows:

> **λ-EDF-AC:** Divide λm processors into λ clusters, each with m processors. For each $1 \leq i \leq \lambda$, only jobs with value density between $k^{(i-1)/\lambda}$ and $k^{i/\lambda}$ will run in a processor of the i-th cluster. When a job is released, the cluster of processors that can serve the job is identified, *and* EDF-AC is used to schedule the jobs in that cluster.

THEOREM 4.1. λ-EDF-AC *is optimal using* λm *speed-*$(2 + k^{1/\lambda})$ *processors.*

Proof. Consider an input job sequence L. Let $L(i)$ be the set of jobs that are allowed to use the i-th cluster. For a set of jobs $\mathcal{X} \subseteq L$, let $\mathcal{O}(\mathcal{X})$ be the set of jobs that meet deadlines when the optimal off-line algorithm schedules $\mathcal{X}$ using m speed-s processors. Note that $\|\mathcal{O}(L(1)) \cup \cdots \cup \mathcal{O}(L(\lambda))\|$ is no less than $\|\mathcal{O}(L)\|$, since the optimal off-line algorithm can always choose to run jobs in $\mathcal{O}(L) \cap L(1), \ldots, \mathcal{O}(L) \cap L(\lambda)$ when scheduling $L(1)$, $\ldots$, $L(\lambda)$ respectively.

The value obtained by λ-EDF-AC is the sum of values obtained by each cluster, i.e. $\|\mathcal{E}_{L(1)}\| + \cdots + \|\mathcal{E}_{L(\lambda)}\|$. Jobs in the same cluster have value densities which differ by at most a factor of $k' = k^{1/\lambda}$. For scheduling $L(i)$, EDF-AC uses m speed-$(2+k^{1/\lambda})$ processors. Applying Theorem 3.2, we have $\|\mathcal{E}_{L(i)}\| \geq \|\mathcal{O}(L(i))\|$. This results in that the value obtained by λ-EDF-AC is at least $\|\mathcal{O}(L(1))\| + \cdots + \|\mathcal{O}(L(\lambda))\| \geq \|\mathcal{O}(L)\|$. □

5 Trading processors for speed

We extend the result in Section 3 to the case when EDF-AC has more processors, apart from that its processors are faster than those of the offline adversary. Our result shows that when the number of processors increases, the speed required for EDF-AC to achieve optimality can be reduced arbitrarily close to 1. In particular, we have the following lemma.

THEOREM 5.1. *Suppose* $M \geq m$ *processors are available to* EDF-AC. *For each integer* M_0 *from* m *to* M, EDF-AC *using speed-*s *processors is optimal, where* $s = 1 + \frac{m}{M_0} + \frac{1}{M - M_0 + 1}$.

By choosing M_0 to be the integer closest to $\frac{\sqrt{m}}{\sqrt{m}+1}(M+1)$, the speed requirement for EDF-AC to be optimal is about $(1 + \frac{(\sqrt{m}+1)^2}{M+1})$.

We can prove Theorem 5.1 based on the framework in Section 3 (i.e., Lemma 3.1, Lemma 3.2, and Theorem 3.1). However, several notions have to be generalized. Instead of keeping track of continuous time periods when EDF-AC is busy, we keep track of continuous time periods when EDF-AC is M_0-*busy*, defined as follows. Consider the scheduling of a set of jobs L. At any time, we say EDF-AC is M_0-busy $(m \leq M_0 \leq M)$ if at least M_0 processors are working throughout this period. Again, we focus on the case when the schedule of EDF-AC contains only one M_0-busy period, say, $[0, \ell]$, and all jobs are released within this period. A job is said to be early-dead if its deadline is at or before $\frac{M_0}{m} sl$, and late-dead if otherwise.

In proving EDF-AC to be optimal when given more but not so fast processors, the most non-trivial part is why Lemma 3.1 still holds in this modified setting. The new proof of Lemma 3.1 requires a relationship between *busy* periods and M_0-busy periods. It is quite straightforward to show

that the properties in Lemma 3.2 continues to hold. Adapting Theorem 3.1 is however slightly more complicated. In particular, we redefine $f(L, \mathcal{O})$ to be the portion of work of the jobs in $\mathcal{E}_L - \mathcal{O}$ scheduled by EDF-AC after the last M_0-busy period. Details will appear in the full paper.

6 Scheduling jobs with different value densities

In general, the value $v(J)$ obtained by completing a job J may not be proportional to the amount of work $p(J)$ required to complete the job, and the jobs may have varying value density (the *value density* of J, denoted by $\rho(J)$, is the ratio $v(J)/p(J)$). The aim of the scheduler is to maximize the total value instead of the total work. For scheduling jobs with varying value densities on a single processor, the algorithm SLACKER given by Kalyanasundaram and Pruhs [9] is $(1+\delta^{-1})(1+\delta^{-1/2})(1+\delta^{-1/2}+\delta^{-1})$-competitive when using a speed-$(1+2\delta)$ processor for any $\delta > 0$. In this section, we give an extension of SLACKER, denoted MSLACKER, for scheduling jobs on $m \geq 1$ processors, and show that this extension is $(1+2\delta^{-1}+4\delta^{-2})$-competitive when given speed-$(1+2\delta)$ processors. Since SLACKER is a special case of MSLACKER, our result also improves the competitive ratio of SLACKER given by [9]. This improvement is mainly resulted from a new upper bound on the total value of jobs that can be completed by the adversary.

MSLACKER is parameterized by two real values $\delta > 0$ and $c > 1$. MSLACKER is equipped with m speed-s processors where $s = 1 + 2\delta$, and keeps an initially empty set of privileged jobs M. At any time, MSLACKER runs all jobs in M if M has at most m jobs; otherwise, it runs the m highest-value-density jobs in M. When a job J is released, J is added to M if M contains less than m jobs, or $\rho(J) \geq c\,\rho(J_o)$ where J_o is the m-th highest-value-density job in M. If J cannot be immediately added to M, the same checking will be done again whenever a job is completed, until the remaining slack (i.e., $d(J) - \frac{p(J)}{s} - t$, where t is the current time) is less than $\delta\frac{p(J)}{s}$. A job is removed from M if either it is completed, or its remaining slack becomes negative. Figure 2 shows how MSLACKER considers a job for execution.

Below we analyze the performance of MSLACKER, showing that MSLACKER, when given speed-$(1+2\delta)$ processors and when c is chosen as $1+2\delta^{-1}$, is $(1+2\delta^{-1}+4\delta^{-2})$-competitive. Let A, C, and R denote the sets of jobs completed by the adversary, completed by MSLACKER, and ever added into M, respectively. Let $\|C\|$ be the total value of jobs in C, and similar for $\|A\|$ and $\|R\|$. By definition, $\|C\| \leq \|R\|$. A released job J is said to be *fresh* at any time before $d(J) - (1+\delta)\frac{p(J)}{s}$.

To show the competitive ratio of MSLACKER, we need a lower bound of $\|C\|$, as well as an upper bound of $\|A\|$. Intuitively, MSLACKER is very conservative and can complete most of jobs added into M; specifically, we show that $\|C\|$ is at least a significant fraction of $\|R\|$ (see Lemma 6.1). On the other hand, the adversary may be able to pick some more valuable jobs to execute, but the way MSLACKER selects the jobs guarantees that $\|A\|$ cannot exceed $\|R\|$ too much (see Lemma 6.3).

LEMMA 6.1. $\|C\| \geq \frac{\delta(c-1)-1}{\delta(c-1)}\|R\|$.

Proof. The proof is essentially a generalization of the technique for analyzing SLACKER in the single-processor setting [9]. Details are left in the full paper. □

Unlike the proof given in [9], we derive the lower bound of $\|A\|$ by analyzing separately the jobs that are completed by MSLACKER and jobs that are not. This leads to a simpler and tighter analysis. First of all, let us observe a simple property of MSLACKER.

LEMMA 6.2. *At any time t when a job J has not yet completed by* SLACKER *and J is still fresh, then either* MSLACKER *is executing J, or* MSLACKER *is executing m other jobs each with value density at least $\rho(J)/c$.*

Proof. At time t, J may or may not be in M. If J is in M and not being executed, then the value density of any job SLACKER is executing is at least $\rho(J)$. Next, we consider J not in M. Throughout the period $[r(J), t]$, J is not qualified to get into M, i.e., M contains at least m jobs, and $\rho(J) < c\,\rho(J_o)$ where J_o is the m-th highest-value-density job in M. At time t, MSLACKER is

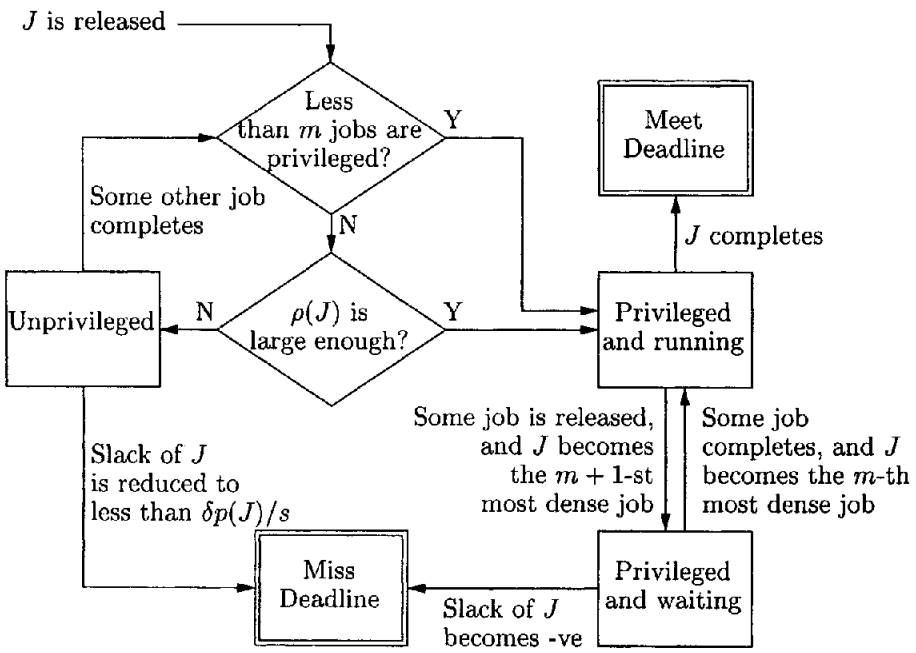

Figure 2: **The life-cycle of a job J as scheduled by MSLACKER.** After J is released, it tries to become a privileged job by going through the two tests. It may run in a processor only after it is privileged. However, since there are only m processors, some privileged jobs may have to wait. Jobs that wait for too long to be privileged or to run in a processor will miss deadlines.

executing m other jobs each with value density at least $\rho(J)/c$. □

We are now ready to show the upper bound of $\|A\|$.

LEMMA 6.3. $\|A\| \leq \|C\| + \frac{c}{\delta}\|R\|$.

Proof. Partition A into $A^c = A \cap C$ and $A^u = A - C$. Since $A^c \subseteq C$, $\|A\| \leq \|C\| + \|A^u\|$. For any job J in A^u, define $a_1(J)$ (resp. $a_2(J)$) to be the total amount of time when the adversary executes J while J is fresh (resp. J is no longer fresh). Obviously, $a_1(J) + a_2(J) = p(J)$. For any job $J \in A^u$, $a_2(J) \leq (1+\delta)\frac{p(J)}{s}$, and $a_1(J) \geq \delta\frac{p(J)}{s}$. Thus, $\frac{\delta}{s}v(J) = \frac{\delta}{s}p(J)\rho(J) \leq a_1(J)\rho(J)$, and $\frac{\delta}{s}\|A^u\| \leq \sum_{J\in A^u} a_1(J)\rho(J)$.

To derive an upper bound of $\|A\|$, we consider $a_1(J)$ for each job $J \in A^u$. By definition, every job $J \in A^u$ is not completed by MSLACKER. At any time when the adversary executes a job $J \in A^u$ while J is fresh, Lemma 6.2 tells us that MSLACKER either executes J, or executes m jobs each of value density at least $\rho(J)/c$. In general, at any time t, let X_t be the set of fresh jobs in A^u currently executed by the adversary; then for each job $J \in X_t$, we can identity a distinct job currently executed by MSLACKER with job density at least $\rho(J)/c$. In other words, the total value density of jobs in X_t is at most c times the total value density of jobs currently executed by MSLACKER. To bound $\sum_{J\in A^u} a_1(J)\rho(J)$, it suffices to consider the sum over all time t of the total value density of jobs in X_t, which is at most c times of the sum over all time t of the total value density of jobs executed by MSLACKER at time t. Note that each job $J \in R$ can contribute a quantity of at most $\rho(j)\frac{p(J)}{s}$ to the latter sum. Thus, the latter sum can be expressed as $\sum_{J\in R}\rho(j)\frac{p(J)}{s}$, which is equal to $\frac{1}{s}\|R\|$. In summary, $\sum_{J\in A^u} a_1(J)\rho(J) \leq \frac{c}{s}\|R\|$. Therefore, $\delta\|A^u\| \leq c\|R\|$, and $\|A\| \leq \|C\| + \|A^u\| \leq \|C\| + \frac{c}{\delta}\|R\|$. □

THEOREM 6.1. MSLACKER, *when given speed-*$(1+$

2δ) *processors and when* c *is chosen as* $1 + 2\delta^{-1}$, *is* $(1 + 2\delta^{-1} + 4\delta^{-2})$*-competitive.*

Proof. It follows from Lemmas 6.1 and 6.3. □

References

[1] S. Baruah. *Overload tolerance for single-processor workloads.* In IEEE Symposium on Real time technology and application, pages 2–11, 1998.

[2] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. *On-line scheduling in the presence of overload.* In Proc. 1991 IEEE Real-Time Systems Symposium, pages 101-110, 1991.

[3] P. Berman and C. Coulston. *Speed is more powerful than clairvoyance.* In Proc. 6th SWAT, pages 255–263, 1998.

[4] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[5] M. Brehob, E. Torng, and P. Uthaisombut. *Applying extra-resource analysis to load balancing.* In Proc. 11th ACM-SIAM SODA, pages 560–561, 2000.

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

[7] M. L. Dertouzos and A. K. L. Mok. *Multiprocessor on-line scheduling of hard-real-time tasks.* IEEE Transactions on Software Engineering, 15(12):1497–1506, 1989.

[8] J. Edmonds. *Scheduling in the dark.* In Proc. 31st ACM STOC, pages 179–188, 1999.

[9] B. Kalyanasundaram and K. R. Pruhs. *Speed is as powerful as clairvoyance.* J. ACM, 2000. To appear. Preliminary version appeared in *Proc. 27th FOCS* (1995), pp. 214–221.

[10] 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.

[11] G. Koren, D. Shasha, and S. C. Huang. *MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling.* In Proc. 14th Real-Time Systems Symposium, pages 172–181, 1993.

[12] T. W. Lam and K. K. To. *Trade-offs between speed and processor in hard-deadline scheduling.* In Proc. 10th ACM-SIAM SODA, pages 623–632, 1999.

[13] T. W. Ngan. *Private communication.*

[14] C. A. Phillips, C. Stein, E. Torng, and J. Wein. *Optimal time-critical scheduling via resource augmentation.* In Proc. 29th ACM STOC, pages 140–149, 1997.

[15] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo. *Deadline scheduling for real-time systems: EDF and related algorithms.* Kluwer Academic Publishers, 1998.