Extra Processors versus Future Information in Optimal Deadline Scheduling*

Chiu-Yuen Koo† Tak-Wah Lam‡§
Tsuen-Wan "Johnny" Ngan¶§ Kar-Keung To‡

Abstract

This paper is concerned with the design of online scheduling algorithms that exploit extra resources. In particular, it studies how to make use of multiple processors to counteract the lack of future information in online deadline scheduling. Our results extend the previous work that are primarily based on using a faster processor to obtain a performance guarantee. The challenge arises from the fact that jobs are sequential in nature and cannot be executed on more than one processor at the same time. Thus, a faster processor can speed up a job while multiple unit-speed processors cannot.

1 Introduction

Online algorithms for scheduling jobs with deadlines on a processor have been studied extensively in the literature. A typical example is the earliest deadline first (EDF) algorithm, which has been widely used in many real-time systems (see [19] for a survey). For scheduling underloaded systems, EDF is optimal, i.e., whenever there exists a schedule meeting the deadlines of all jobs released, EDF can always do so [7]. However, when the system is possibly overloaded, no algorithm has a worst-case performance guarantee in the sense that the performance cannot match or be competitive against the offline adversary [2]. In recent years, a plausible approach to achieving better performance guarantee for online scheduling (without restricting the inputs) is to allow the online scheduler to use a faster processor than the offline adversary [3, 5, 8, 11, 15, 17]. Intuitively, we use a faster processor to compensate the online scheduler for the lack of future information. The key question is whether a moderate amount of extra speed can lead to satisfactory competitiveness. Kalyanasundaram and Pruhs [11] were the first to exploit a faster processor to derive an online algorithm whose competitive ratio is bounded by a constant. Subsequently, it has been shown that even a 1-competitive algorithm can be constructed [16].

*A preliminary version of this paper appeared in the Proceedings of the 14th ACM Annual Symposium on Parallel Algorithms and Architectures, 2002.

†Department of Computer Science, University of Maryland, College Park, MD 20742, USA (cykoo@cs.umd.edu).

‡Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong ({twlam,kkto}@cs.hku.hk).

§This research was supported in part by Hong Kong RGC Grant HKU-7024/01E.

¶Department of Computer Science, Rice University, Houston, TX 77005, USA (twngan@cs.rice.edu).

An alternative to using a faster processor is exploiting multiple unit-speed processors. Note that a job, in general, is not parallelizable and cannot be executed by more than one processor at a time. While a faster processor can speed up a job, multiple unit-speed processors cannot. In other words, we cannot use m unit-speed processors to simulate an m-times faster processor, yet the reverse is possible (using time-sharing). In this paper we show in the affirmative that multiple unit-speed processors can be used to counteract the lack of future information. The number of processors required for a 1-competitive deadline scheduling algorithm is of the same order of magnitude as the extra speed requirement given in the previous work. Our new result holds whether job migration among processors is allowed or not. Details are as follows.

Problem definition: We study the following scheduling problem on a processor, which is known as the *firm-deadline scheduling problem* in the literature. Jobs are released at unpredictable times, each being sequential in nature (i.e., cannot be executed by more than one processor at a time) and independent from others. The processing time, deadline, and value of a job are known when the job is released. Deadlines are firm in the sense that completing a job after its deadline gives no value. A scheduler aims to maximize the total *value* of jobs that are completed by their deadlines. Preemption is allowed at no cost, and a preempted job can be resumed at the point of preemption on any processor. We consider both settings where job migration is allowed at no cost and disallowed, respectively. In general, a system may be *overloaded* in the sense that there is no schedule meeting the deadlines of all jobs released. For more details of firm-deadline scheduling, one can refer to [19].

The value of a job reflects its importance and is not necessarily related to the processing time. The *value density* of a job is defined to be its value divided by its processing time, and the *importance ratio* k of a system is the ratio of the largest possible value density to the smallest possible one. When $k = 1$, it means that every job has a value proportional to its computation time.

We analyze the performance of online algorithms with respect to their competitiveness (see, e.g., [4, 18]). For any $c \geq 1$, an online algorithm $\mathcal{A}$ is said to be c-competitive if for any job sequence, $\mathcal{A}$ guarantees to obtain at least a factor $1/c$ of the total value obtained by any offline algorithm. When $c = 1$, $\mathcal{A}$ guarantees to match the value obtained by any offline algorithm.

Previous work: The early work of Dertouzos [7] showed that for underloaded systems, the Earliest Deadline First (EDF) strategy is 1-competitive. But in general, no $O(1)$-competitive firm-deadline scheduler exists; indeed, the best possible competitive ratio is $(1 + \sqrt{k})^2$ (Baruah et al. [2], Koren and Shasha [14]). To obtain better performance guarantees, one can allow online schedulers to use a faster processor. Specifically, one compares an online scheduler that is given a faster processor but has no knowledge about the future against an offline scheduler that uses only a unit-speed processor but has complete information about the jobs. For the firm-deadline scheduling problem, Kalyanasundaram and Pruhs [11] showed that the competitive ratio can be improved from $(1 + \sqrt{k})^2$ to a constant if the online scheduler is given a slightly faster processor (e.g., 31.9 with a double-speed processor); more recently, Lam and To [16] gave an algorithm that is 1-competitive

using a processor of $4\lceil \log k \rceil$ times faster.

This paper investigates online algorithms that use multiple unit-speed processors instead of a faster processor to counteract the lack of future information. We say that an algorithm $\mathcal{A}$ is m-processor c-competitive if $\mathcal{A}$ using m unit-speed processors can obtain at least a fraction $1/c$ of the value obtained by any offline algorithm using one unit-speed processor. It was not known before how to exploit multiple unit-speed processors (instead of a faster processor) to derive an $O(1)$- or 1-competitive algorithm. Nevertheless, there are two previous results for restricted cases. Baruah [1] considered jobs with uniform value density (i.e., $k = 1$) and gave an m-processor $m/(m-1)$-competitive algorithm (note that without extra resources, the best competitive ratio is 4). If the concern is to maximize the total number of job completions, Kalyanasundaram and Pruhs [9] gave a two-processor $O(1)$-competitive algorithm. *

Summary of results: In this paper we resolve in the affirmative that using only extra processors can give a 1-competitive algorithm for the firm deadline scheduling problem. Assuming that migration is allowed, we give a two-processor 1-competitive algorithm for $k = 1$, and a $4\lceil \log k \rceil$-processor 1-competitive algorithm for general k. The processor bound is asymptotically tight as it is relatively easy to show that any m-processor 1-competitive algorithm requires $m \geq \lfloor \log k \rfloor$ (see the Appendix). † Eliminating migration via more processors has been an interesting problem even in the offline setting (e.g., [10, 12]). In this paper, we show that when migration is not allowed, a 1-competitive algorithm can still be attained with a slight increase in the number of processors — three processors for $k = 1$ and $6\lceil \log k \rceil$ processors for general k.‡

Organization of the paper: The remainder of this paper is organized as follows. Section 2 shows a new way to enhance EDF for the case when $k = 1$, obtaining a two-processor 1-competitive algorithm called EDF-Plus. Section 3 extends EDF-Plus to handle the cases for general k. The $\lfloor \log k \rfloor$ processor lower bound is given in Section 4.2. Section 4 gives a non-migratory version of EDF-Plus, which is three-processor 1-competitive for $k = 1$ and $6\lceil \log k \rceil$-processor 1-competitive for general k.

Notations: Throughout this paper, we denote the release time, processing time, deadline, and value of a job J as $r(J), p(J), d(J)$, and $v(J)$, respectively. For any set $\mathcal{S}$ of jobs, $p(\mathcal{S})$ denotes the total processing time of the jobs in $\mathcal{S}$. For a system with importance ratio k, we assume that all jobs have their value densities normalized to the range $[1, k]$. Furthermore, we assume that jobs have distinct release times and deadlines (ties can be

*The past few years have also witnessed the application of extra-resource analysis to other difficult online scheduling problems. In particular, for multi-processor scheduling in the underloaded setting, Phillips et al. [17] have shown that using two times faster processors or $\log \Delta$ times more processors can lead to optimal online scheduling, where Δ is the ratio of the processing time of the longest job to that of the shortest job. Note that job values are not a concern in this context as in the underloaded setting, it is always feasible to schedule all jobs by their deadlines and an optimal online scheduling must complete all jobs.

†Recently Chrobak et al. [6] have independently obtained a similar lower bound.

‡Following previous work, this paper assumes that the importance ratio k is known in advance. Such an assumption can actually be removed by allocating processors on a needed basis (see, e.g., [17]); our algorithms remain valid, with the processor complexity increasing by an additive constant (precisely, the $\lceil \log k \rceil$ factor increases to $(\lceil \log k \rceil + 1)$).

broken using their identification numbers).

All algorithms in this paper are based on EDF, which refers to the strategy of scheduling the job with the earliest deadline. Note that the current job will be preempted when a new job with an earlier deadline is released. EDF is often supplemented with some kind of admission control to avoid excessive preemption when the system is overloaded. Below EDF-Ac denotes EDF enhanced with the following simple form of admission control: Upon release, a job must pass a test to get admitted for EDF scheduling. The test simply checks whether the new job together with the previously admitted jobs can all be completed by their deadlines using (plain) EDF. EDF-Ac is 1-competitive using a $4\lceil \log k \rceil$ times faster processor [16].

2 The EDF-Plus algorithm

In this section we discuss a new algorithm called EDF-Plus which is two-processor 1-competitive for scheduling jobs with value density equal to one. It is known that EDF (and EDF-Ac) is one-processor 1-competitive for underloaded systems [7]. Yet this is not true for overloaded systems. Intuitively, it is too difficult for EDF and EDF-Ac to select the right jobs so as to maximize the overall processing time. For example, EDF-Ac can make a mistake in rejecting a long job due to the earlier admission of a shorter job with tight deadline. We improve EDF-Ac based on a simple idea. When EDF-Ac mistakenly rejects a job, we give the job a second chance by scheduling it on another processor temporarily; after a while, the remaining processing time will get smaller and hopefully, the job can get admitted by EDF-Ac. Thus, the enhanced EDF-Ac will be more productive.

The above observation leads us to design an algorithm using two processors, denoted M_e and M_p (the subscripts carry the meaning that these two processors select jobs with earliest deadlines and largest processing times). M_e schedules jobs using EDF-Ac. Once M_e admits a job, the job is guaranteed to be completed. A rejected job is considered by M_p immediately. M_p aims at deferring the time EDF-Plus discards some jobs by temporarily scheduling them. The job in M_p will repeatedly attempt to migrate to M_e by going through the admission control of M_e whenever M_e completes a job. This makes EDF-Plus inherently migratory. Note that at any time, there may be more than one job rejected by M_e. Yet M_p needs to schedule only one using some simple greedy strategy. Below we give two such strategies, both attempt to capture the currently biggest job in accordance to a certain definition.

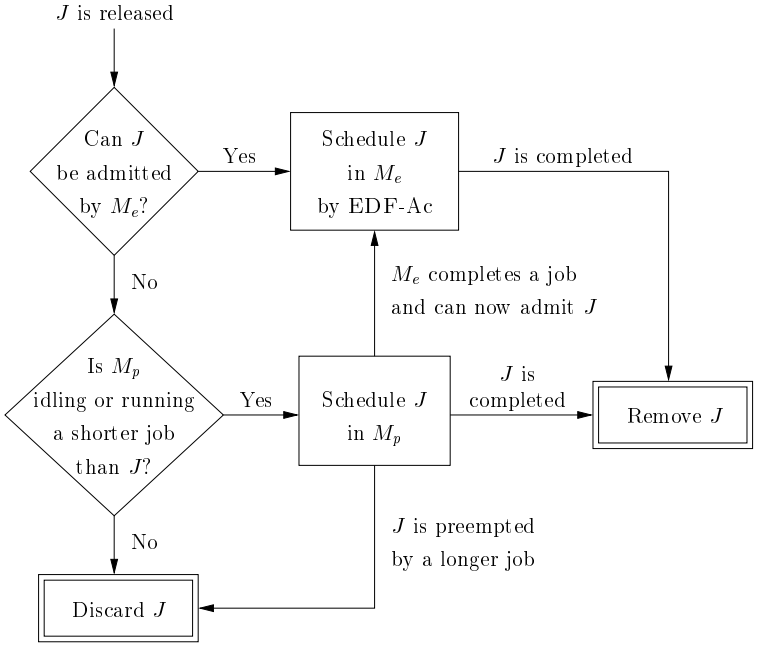
- **maximum processing time**: M_p works on a job as soon as it is rejected by M_e. In case there are more than one such job, M_p works on the one with the longest processing time and discards all other jobs.

- **zero slack time**: M_p works on a rejected job only when the job has zero slack time (i.e., deadline = current time + processing time); in case there is more than one such job, preference will be given to the one with the latest deadline.

```
(1)   Initialization: Q ← ∅
(2)
(3)   When job J is released:
(4)     if M_e can complete all jobs in Q ∪ {J} using EDF
(5)       Q ← Q ∪ {J};
(6)       M_e runs the job with the earliest deadline job in Q
(7)     else if M_p is idle or p(J) > p(J_{M_p}), where J_{M_p} is the job running in M_p
(8)       J_{M_p} is discarded and M_p runs J
(9)     else
(10)      J is discarded
(11)
(12)  When M_e completes job J:
(13)    Q ← Q − {J};
(14)    let J_{M_p} be the job running in M_p;
(15)    if M_e can complete all jobs in Q ∪ {J_{M_p}} using EDF
(16)      Q ← Q ∪ {J_p}   // M_p becomes idle
(17)    M_e runs the job with the earliest deadline in Q
```

Algorithm 1: The EDF-Plus algorithm

The first strategy gives us an extra property which is useful in extending the algorithm for general k in the migratory setting, while the second strategy favors the non-migratory setting in Section 4. In the following we only prove the correctness of EDF-Plus based on the first strategy. The details of EDF-Plus are shown in Algorithm 1. It makes use of a queue called Q to store all admitted jobs to be completed by M_e. The life cycle of a job is depicted in the following figure.



Theorem 1. *For scheduling jobs with uniform value density,* EDF-Plus *is two-processor* 1*-competitive (against one-processor offline schedulers).*

In the remainder of this section, we prove Theorem 1 by contradiction. Assume that EDF-Plus is not 1-competitive for some job sequence. Let $\mathcal{I}$ be such a sequence containing the fewest jobs. Without loss of generality, we assume the release time of the first job is 0. In Lemma 1, we establish that M_e is busy over exactly one continuous period in the course of scheduling $\mathcal{I}$. Then in Lemma 2, we show an interesting property of the job J_ℓ in $\mathcal{I}$ that has the latest deadline. Based on these lemmas, we can argue that the total processing time of jobs completed by M_e is more than $d(J_\ell)$ (see Lemma 3). Note that jobs of $\mathcal{I}$ can only be scheduled within the period $[0, d(J_\ell)]$. Thus, an offline algorithm, using one processor, obtains a total value (processing time) of at most $d(J_\ell)$. This contradicts that EDF-Plus is not optimal for $\mathcal{I}$ and Theorem 1 is proved.

Fact 1. *At any time, if M_e is idle, then Q is empty and M_p is idle.*

Lemma 1. *In the course of scheduling $\mathcal{I}$ by* EDF-Plus*, M_e is busy over exactly one continuous period.*

Proof. Assume that M_e is busy over two or more disjoint periods. Let t_ℓ be the start time of the last busy period. By Fact 1, both M_e and M_p are idle immediately before t_ℓ. Therefore, if we partition $\mathcal{I}$ into two parts, one for jobs with release time before t_ℓ and one for the rest, and schedule them using EDF-Plus separately, the schedule produced is not changed. Since EDF-Plus is not 1-competitive for input $\mathcal{I}$, at least one of the two parts gives a job sequence where EDF-Plus is not 1-competitive. This contradicts the assumption that $\mathcal{I}$ is the smallest counterexample. □

We need the following notion to analyze J_ℓ, the job with the latest deadline.

Definition 1. Consider any time t when M_e (in general, any processor using EDF for admission control) fails to admit a job J. That is, if M_e uses EDF to schedule J together with the jobs admitted before t, then some job J_o (which could be J itself) will miss its deadline. Such a job J_o is said to *repudiate* J at t.

Fact 2. *A job can repudiate itself as well as jobs with earlier deadlines, but it cannot repudiate jobs with later deadlines.*

Lemma 2. *In the course of scheduling $\mathcal{I}$ by* EDF-Plus*, there is at least one time when J_ℓ repudiates itself or another job.*

Proof. Suppose to the contrary that J_ℓ never repudiates any job including itself. Then, when J_ℓ is considered by M_e at its release time, no job could repudiate J_ℓ as J_ℓ does not repudiates itself and any other job has a deadline earlier than $d(J_\ell)$. Thus, J_ℓ must be admitted by M_e. Consider any moment after J_ℓ is admitted. Any newly released job, if rejected by M_e, must be repudiated by a job other than J_ℓ. Recall that M_e is running EDF-Ac and J_ℓ has the latest deadline. If we remove J_ℓ from $\mathcal{I}$, M_e will not admit more jobs and EDF-Plus loses exactly the processing time of J_ℓ. On the other hand, the

optimal offline algorithm loses at most the processing time of J_ℓ. Thus, $\mathcal{I} - \{J_\ell\}$ is a job sequence for which EDF-Plus is not 1-competitive. This contradicts the assumption that $\mathcal{I}$ is the smallest counterexample. □

Lemma 3. *The value obtained by* EDF-Plus *in scheduling* $\mathcal{I}$ *is more than* $d(J_\ell)$.

Proof. By Lemma 2, J_ℓ repudiates some job J at some time t, where $r(J_\ell) \le t \le d(J_\ell)$. Thus M_e must be busy at time t. Furthermore, by Lemma 1, M_e is busy throughout the period $[0, t]$. By the definition of repudiation, at time t, using EDF to schedule the jobs currently found in Q and J will cause J_ℓ to miss its deadline. In other words, M_e is committed to process admitted jobs up to a time later than $d(J_\ell) - p(J)$, attaining a total value of more than $d(J_\ell) - p(J)$.

Next, we show that J or another even longer job that has not yet been admitted by M_e will be completed by EDF-Plus. After M_e rejects J at time t, there are three possible scenarios: (1) J is scheduled to completion on M_p; (2) J is scheduled on M_p and later migrates to M_e; or (3) J is discarded before its deadline by M_p due to the presence of another rejected job with longer processing time. In the last case, EDF-Plus guarantees that a rejected job with longer processing time will eventually be completed. The value obtained in scheduling rejected jobs is at least $p(J)$.

Therefore, the total value obtained by EDF-Plus in scheduling $\mathcal{I}$ is more than $d(J_\ell) - p(J) + p(J) = d(J_\ell)$. □

3 Non-uniform value densities

In this section, we first present an algorithm called EDF-MSp which is 4-processor 1-competitive for scheduling jobs with importance ratio at most 2. Then we show that for jobs with importance ratio at most k (≥ 2), a simple extension of EDF-MSp can give a $4\lceil \log k \rceil$-processor 1-competitive algorithm. EDF-MSp uses four processors, which are divided into two *bands*, each containing two processors. When a job is released, it is first considered by Band 1, which is running EDF-Plus. If Band 1 discards the job (at line 8 or line 10 in Algorithm 1), the job is passed to Band 2, which runs a different algorithm to be described later.

For any job sequence $\mathcal{I}$, let $\mathcal{A}_1$ and $\mathcal{O}$ be the sets of jobs completed by EDF-Plus and an optimal offline algorithm Opt, respectively. Recall that EDF-Plus guarantees that $p(\mathcal{A}_1) \ge p(\mathcal{O})$, i.e., $p(\mathcal{A}_1 - \mathcal{O}) \ge p(\mathcal{O} - \mathcal{A}_1)$. Though jobs in $\mathcal{O} - \mathcal{A}_1$ may have higher value, the importance ratio is at most two and $\|\mathcal{O} - \mathcal{A}_1\|$, the total value of $\mathcal{O} - \mathcal{A}_1$, is at most $2p(\mathcal{O} - \mathcal{A}_1)$. EDF-MSp is 1-competitive if the Band 2 processors can complete a subset $\mathcal{A}_2$ of jobs discarded by EDF-Plus with sufficient processing time, say, $p(\mathcal{A}_2) \ge p(\mathcal{O} - \mathcal{A}_1)$. Then we can conclude that $\|\mathcal{A}_1\| + \|\mathcal{A}_2\| \ge \|\mathcal{O}\|$. Yet to achieve such a lower bound on $p(\mathcal{A}_2)$ seems to be very difficult. (This is mainly due to the fact that some jobs are not passed to Band 2 immediately upon their release.) In fact, our algorithm takes advantage of a less demanding requirement, namely, $p(\mathcal{A}_2) \ge p(\mathcal{O}')$, where $\mathcal{O}' = \mathcal{O} - \mathcal{A}_1 - \mathcal{A}_2$.

Lemma 4. *Suppose* $p(\mathcal{A}_2) \ge p(\mathcal{O}')$. *Then* $\|\mathcal{A}_1\| + \|\mathcal{A}_2\| \ge \|\mathcal{O}\|$.

```
(1)   Initialization:
(2)     Q' ← ∅   // A job is kept in Q' until its slack time is zero.
(3)
(4)   When job J is passed to band 2:
(5)     if M_r is idle or r(J) < r(J_{M_r}) where J_{M_r} is the job running in M_r
(6)       Q' ← Q' ∪ {J_{M_r}};
(7)       M_r runs J
(8)     else
(9)       Q' ← Q' ∪ {J}
(10)
(11)  When M_r completes job J:
(12)    if Q' ≠ ∅
(13)      Q' ← Q' − {J'} where J' is chosen arbitrarily from Q';
(14)      M_r runs J'
(15)    else if M_d is working on a job J_{M_d}
(16)      M_r runs J_{M_d}   // M_d becomes idle
(17)
(18)  When job J ∈ Q' has zero slack time:
(19)    Q' ← Q' − {J}
(20)    if M_d is idle or d(J) > d(J_{M_d}) where J_{M_d} is the job running in M_d
(21)      M_d runs J; J_{M_d} is discarded
(22)    else
(23)      J is discarded
```

Algorithm 2: MSp — the algorithm for Band 2.

Proof. Let $\mathcal{S}_1 = \mathcal{A}_1 \cap \mathcal{O}$ and $\mathcal{S}_2 = \mathcal{A}_2 \cap \mathcal{O}$. Note that $\mathcal{O}' = \mathcal{O} - \mathcal{S}_1 - \mathcal{S}_2$ and $\|\mathcal{O}\| = \|\mathcal{S}_1\| + \|\mathcal{S}_2\| + \|\mathcal{O}'\|$. Furthermore, let $\mathcal{A}'_1 = \mathcal{A}_1 - \mathcal{S}_1$, and let $\mathcal{A}'_2 = \mathcal{A}_2 - \mathcal{S}_2$. Since $p(\mathcal{A}_1) \geq p(\mathcal{O})$, we have $p(\mathcal{A}'_1) = p(\mathcal{A}_1) - p(\mathcal{S}_1) \geq p(\mathcal{O}) - p(\mathcal{S}_1) = p(\mathcal{O}') + p(\mathcal{S}_2)$. Moreover, $p(\mathcal{A}_2) \geq p(\mathcal{O}')$ and thus $p(\mathcal{A}'_2) = p(\mathcal{A}_2) - p(\mathcal{S}_2) \geq p(\mathcal{O}') - p(\mathcal{S}_2)$. In summary, we have $p(\mathcal{A}'_1) + p(\mathcal{A}'_2) \geq 2p(\mathcal{O}') \geq \|\mathcal{O}'\|$. Therefore, $\|\mathcal{A}_1\| + \|\mathcal{A}_2\| \geq \|\mathcal{S}_1\| + \|\mathcal{S}_2\| + p(\mathcal{A}'_1) + p(\mathcal{A}'_2) \geq \|\mathcal{O}\|$. □

Below we show an algorithm called MSp for Band 2 which satisfies the requirement that $p(\mathcal{A}_2) \geq p(\mathcal{O}')$. Then by Lemma 4, we can conclude that EDF-MSp is 4-processor 1-competitive.

First of all, we note that a job J passed to MSp is discarded by EDF-Plus either at $r(J)$ or strictly after $r(J)$. For the latter case, the definition of EDF-Plus gives us the following property.

Fact 3. If a job J is discarded by EDF-Plus at time $t > r(J)$, it has been running on M_p during $[r(J), t]$.

Denote the two processors of MSp as M_r and M_d (the subscripts carry the meaning that these processors select jobs based on release times and deadlines, respectively). De-

tails of MSp are shown in Algorithm 2. Intuitively, for every job J discarded by Band 1, we hope that either J is completed in Band 2, or M_r is busy during the entire period $[r(J), d(J)]$. M_r attempts to schedule and complete any job discarded by Band 1. However, to guarantee productivity as much as possible, a discarded job from Band 1 with an earlier release time can preempt the current job in M_r. On the other hand, for any job J that is not scheduled by M_r, we give J a second chance by scheduling it in M_d temporarily and J may migrate to M_r whenever M_r becomes idle. This keeps M_r busy as long as possible. M_d uses the zero slack time strategy, i.e., M_d runs a job J only when J's slack time is zero; ties are broken using the latest deadline. Intuitively, such a strategy allows M_d to retain a job (for future migration) as long as possible. Similar to the two processors of EDF-Plus, M_r and M_d have the following relationship.

Fact 4. At any time, if M_r is idle, then Q' is empty and M_d is idle.

The crux of the analysis of Band 2 is captured by the following lemma and theorem. Recall that with respect to a given job sequence $\mathcal{I}$, we denote the set of jobs completed by Band 1 and Band 2 as $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Let $\mathcal{I}'$ be the set of jobs not completed by EDF-MSp (i.e., $\mathcal{I}' = \mathcal{I} - \mathcal{A}_1 - \mathcal{A}_2$). We need the following notion of *span*.

Definition 2. The span of a job J is the period $[r(J), d(J)]$, and the span of a set $\mathcal{S}$ of jobs is the union of the spans of all the jobs in $\mathcal{S}$. (E.g., the union of the spans $[3, 6]$ and $[5, 8]$ is $[3, 8]$.) Furthermore, let $sp(\mathcal{S})$ be the total time included in the span of $\mathcal{S}$.

Definition 3. At any time t, EDF-MSp is said to be *productive* on $\mathcal{A}_2$ if a job $J \in \mathcal{A}_2$ is running on one of the four processors of EDF-MSp.

Lemma 5. *Let J be any job discarded by* EDF-MSp. *Then* EDF-MSp *is productive on $\mathcal{A}_2$ throughout the span of J, i.e., $[r(J), d(J)]$.*

Before giving a proof, we note that Lemma 5 can easily lead to the conclusion that that EDF-MSp is a four-processor 1-competitive algorithm. See the following corollary and theorem.

Corollary 1. *(i) $p(\mathcal{A}_2) \geq sp(\mathcal{I}')$; (ii) $p(\mathcal{A}_2) \geq p(\mathcal{O}')$.*

Proof. Statement (i): $p(\mathcal{A}_2)$ is at least the total time during which EDF-MSp is productive on $\mathcal{A}_2$. Lemma 5 ensures that for any job $J \in \mathcal{I}'$, EDF-MSp is productive on $\mathcal{A}_2$ during the span of J. In other words, EDF-MSp is productive on $\mathcal{A}_2$ during the span of $\mathcal{I}'$. Therefore, $p(\mathcal{A}_2) \geq sp(\mathcal{I}')$.

Statement (ii): By Theorem 1, $p(\mathcal{A}_2) \geq sp(\mathcal{I}')$. As $\mathcal{O}' \subseteq \mathcal{I}'$, we have $sp(\mathcal{I}') \geq sp(\mathcal{O}')$. Since Opt schedules at most one job at a time, it follows that $p(\mathcal{O}') \leq sp(\mathcal{O}')$. In summary, $p(\mathcal{A}_2) \geq sp(\mathcal{I}') \geq sp(\mathcal{O}') \geq p(\mathcal{O}')$. □

By Lemma 4, we can further conclude that $\|\mathcal{A}_1\| + \|\mathcal{A}_2\| \geq \|\mathcal{O}\|$. Thus, we have the following theorem.

Theorem 2. *For scheduling jobs with value densities in the range* $[1, 2]$, EDF-Plus *is* $4\lceil \log k \rceil$*-processor* 1*-competitive (against one-processor offline schedulers).*

The rest of this section is devoted to proving Lemma 5.

Lemma 6. *Let J be a job passed to Band 2 at time $t > r(J)$. Then any job J' with $r(J') \leq r(J)$ (i.e., released no later than J), if passed to Band 2, must be passed on or before $r(J)$.*

Proof. The lemma is trivial if J' is passed to Band 2 at $r(J')$. It remains to consider the case where J' is passed to Band 2 at time $t' > r(J')$. By Fact 3, M_p schedules J during the entire nonempty interval $[r(J), t]$, and M_p also schedules J' during the entire interval $[r(J'), t']$. Note that these two intervals cannot overlap. Since $r(J') \leq r(J)$, $[r(J'), t']$ must precede $[r(J), t]$ and thus, $t' \leq r(J)$. □

Lemma 7. *At any time, if M_r is busy, then* EDF-MSp *is productive on $\mathcal{A}_2$.*

Proof. Consider any time t_1 when M_r runs a job J. The lemma holds if J is completed on M_r. It remains to consider the case that J is preempted by another job J' passed to Band 2 at time $t_2 \geq t_1$. By definition of MSp, $r(J') < r(J)$. We want to show that J' is in $\mathcal{A}_2$. Note that $r(J') < r(J) \leq t_1 \leq t_2$. By Fact 3, J' is running on M_p during the period $[r(J'), t_2]$. By Lemma 6, all jobs passed to Band 2 after $t_2 \geq r(J')$ are released later than J'. Therefore, J' cannot be preempted by these jobs and can run up to completion on M_r. Thus, J' is in $\mathcal{A}_2$ and EDF-MSp is productive on $\mathcal{A}_2$ during $[r(J'), t_2]$ and in particular at time t_1. □

Proof of Lemma 5. By Lemma 7, it suffices to show that M_r is busy during the span of J. We divide the span into at most three periods and argue M_r is busy in each period.

- Consider the period from $r(J)$ to the time t_o when J is passed to Band 2. Suppose $t_o > r(J)$. Assume, for the sake of contradiction, that M_r is idle at a certain time $t \in [r(J), t_o]$. Then all jobs passed to Band 2 before t were completed or discarded by t; otherwise M_r would schedule one at t. In other words, any job J' found in Band 2 after time t must be passed to Band 2 at time after $t > r(J)$. By Lemma 6, if $J \neq J'$, then $r(J') > r(J)$. When J is passed to Band 2 at t_o, it can preempt the job currently in M_r (if exist) and will not be preempted afterward. Therefore, J can run up to completion on M_r, contradicting that J is discarded by EDF-MSp.

- As J is discarded eventually, it must have been put into Q'. Consider the period from t_o to the last time t_ℓ when J is removed from Q' for consideration of M_d. At any time within this period, J is in Q' or is processed by M_d or M_r. In both cases, M_r cannot be idle because of Fact 4 (i.e., J is eligible for scheduling on M_r).

- At time t_ℓ, M_d attempts to schedule J. Recall that J is discarded by EDF-MSp. J must be preempted before its deadline. By definition of M_d, this must be due to a job J' with a later deadline. Note that J' may possibly be further preempted or migrated to M_r. In all cases, at any time within the period $[t_\ell, d(J)]$, there is at least one job with deadline on or after $d(J)$ scheduled by either M_r or M_d. In the latter case, by Fact 4, M_r must be busy with some other job. □

Arbitrary importance ratio

EDF-MSp can serve as building block for handling importance ratio of any $k > 1$.

Theorem 3. *For scheduling jobs with value densities in the range $[1, k]$ where $k > 1$, there exists a $4\lceil \log k \rceil$-processor 1-competitive algorithm (against one-processor offline schedulers).*

Proof. Consider the following $4\lceil \log k \rceil$-processor algorithm. Partition the jobs into $\lceil \log k \rceil$ groups, where the i^{th} group contains all the jobs with value density in the range $[2^{i-1}, 2^i]$. Each group is given four processors executing EDF-MSp independently. Within each group, the value densities differ by at most a factor of 2, so the four processors match the value obtained by any offline algorithm for jobs of this group. Therefore, the $4\lceil \log k \rceil$ processors together can match the value obtained by any offline algorithm for jobs with value densities in $[1, k]$. □

4 Non-migratory scheduling

In this section we discuss a non-migratory algorithm N-EDF-Plus(η), which is parameterized by an integer $\eta \geq 1$. We first show that N-EDF-Plus(1) is 3-processor 1-competitive for scheduling jobs with uniform value density (i.e., $k = 1$). Then we show that N-EDF-Plus(2) is 6-processor 1-competitive for scheduling jobs with importance ratio at most 2. Using the technique in Theorem 3, we can make use of N-EDF-Plus(2) to construct a $6\lceil \log k \rceil$-processor 1-competitive algorithm for scheduling jobs with importance ratio at most k, where $k \geq 2$.

N-EDF-Plus(η) uses 3η processors, denoted Me_i, Md_i, and Mu_i where $1 \leq i \leq \eta$. N-EDF-Plus(η) works as follows. Each Me_i is using EDF-Ac with its own queue. When a job J is released, it will be admitted by any one of the Me_i's if possible. If J is rejected by all Me_i's, it is put into a common pool shared by all other processors. Whenever an Md_i is idle, it removes the job with the latest deadline from the pool and works on it until it is completed.

If a job J in the pool has never been picked by an Md_i, its slack time will become zero and it is then said to be *urgent*. In this case, we will try to retain J in the pool by scheduling it on any available Mu_j temporarily. I.e., each Mu_j is using the zero slack time strategy and a job J running on an Mu_j will migrate to an Md_i once Md_i completes a job and J has the latest deadline among all jobs currently in the pool. At any time, up to η urgent jobs in the pool are processed by the Mu_j's; preference is given to those with latest deadlines (and the remaining ones are removed from the pool as they will miss their deadlines). Details of N-EDF-Plus(η) is depicted in Algorithm 3.

Notice that N-EDF-Plus(η) involves job migrations from Mu_j's to Md_i's. Yet these 2η processors, unlike Me_i's, do not commit to any jobs that have been partially executed. Migrating a job from an Mu_j to an Md_i can be avoided by switching the role of the two processors. Thus, N-EDF-Plus(η) is non-migratory in nature.

```
(1)   Initialization: P ← ∅
(2)
(3)   When job J is released:
(4)     if some Me_i can admit J
(5)       Me_i admits J to its EDF queue
(6)       Me_i schedules the job with the earliest deadline in its EDF queue
(7)     else if some Md_i is idling
(8)       Md_i runs J
(9)     else
(10)      P ← P ∪ {J}
(11)
(12)  When Me_i completes a job J:
(13)    Me_i removes J from its EDF queue
(14)    Me_i schedules the job with the earliest deadline in its EDF queue, if any
(15)
(16)  When Md_i completes a job:
(17)    if P ≠ ∅
(18)      Let J be the job in P with the latest deadline
(19)      P ← P − {J}   // if some Mu_j is running J, it becomes idle
(20)      Md_i runs J
(21)
(22)  When J ∈ P becomes urgent (slack time becomes zero):
(23)    if some Mu_i is idling
(24)      Mu_i runs J   // J remains in P
(25)    else
(26)      Let J' be the urgent job other than J in P with the earliest deadline
(27)      Let Mu_j be the processor running J'
           // Mu_j exists; otherwise J' would miss its deadline
(29)      if d(J) > d(J')
(30)        P ← P − {J'}; Mu_j runs J
(31)      else
(32)        P ← P − {J}
```

Algorithm 3: The N-EDF-Plus(η) algorithm

4.1 Analysis of N-EDF-Plus(1)

First, we focus on the case where $\eta = 1$. As there is only one processor of each type, we omit the subscript i and use the notations *Me*, *Md*, and *Mu*. Intuitively, N-EDF-Plus(1) uses two processors *Me* and *Md* to simulate the processor M_e of EDF-Plus so as to avoid any job migration to *Me*.

N-EDF-Plus has the property that a job, once started in *Me* or *Md*, will eventually be completed. We define the *safe processing time* produced by N-EDF-Plus(1) to be the total length of the busy periods of *Me* and the busy periods of *Md*. Then the total processing time of jobs N-EDF-Plus(1) completes is at least the safe processing time. The result that N-EDF-Plus(1) is 1-competitive is based on the following theorem.

Theorem 4. *For scheduling any job set $\mathcal{I}$ with uniform value density, the safe processing time produced by* N-EDF-Plus(1) *is no less than the total processing time of the jobs completed by an optimal offline algorithm using one processor.*

The proof of Theorem 4 is analogous to the proof of EDF-Plus being 1-competitive. Assume on the contrary that Theorem 4 fails for some job sequences. Let $\mathcal{I}$ be such a sequence with the fewest jobs. We suppose the first job is released at time 0. At any time, if *Me* or *Md* is busy, we say that N-EDF-Plus(1) is *busy*. Note that by definition, whenever *Mu* is busy, *Md* and thus N-EDF-Plus(1) are also busy. The following two lemmas are analogous to Lemmas 1 and 2 and their proofs are omitted.

Lemma 8. *In the course of scheduling $\mathcal{I}$,* N-EDF-Plus(1) *is busy over exactly one continuous period.*

Lemma 9. *There exists a moment when J_ℓ, the latest deadline job in $\mathcal{I}$, repudiates some job in Me.*

Based on the above two lemmas, we can prove that N-EDF-Plus(1) can complete jobs with a total time at least $d(J_\ell)$ (see Lemma 10). Jobs in $\mathcal{I}$ can only be processed within the period $[0, d(J_\ell)]$ and therefore any offline algorithm, using one processor, can obtain a value at most $d(J_\ell)$. This contradicts N-EDF-Plus(1) is not 1-competitive for $\mathcal{I}$, and we complete the proof of Theorem 4.

Lemma 10. *The safe processing time is at least $d(J_\ell)$.*

Proof. By Lemma 9, J_ℓ repudiates some job J. Since *Me* only considers a job at its release time, this must happen at $r(J)$, and *Me* must be busy at $r(J)$. By Lemma 8, N-EDF-Plus(1) is busy during the period $[0, r(J)]$. Thus, the safe processing time up to $r(J)$ is at least $r(J)$.

Now we examine the situation at $r(J)$. Since J_ℓ repudiates J, using EDF to schedule the jobs currently found in the queue of *Me* and J causes J_ℓ to miss its deadline. Thus, *Me* must have committed to process admitted jobs up to a time later than $d(J_\ell) - p(J)$, and *Me* is busy over the period $[r(J), d(J_\ell) - p(J)]$.

After *Me* rejects J, J will either be scheduled to completion by *Md* or *Mu*, or be discarded. For the former case, J cannot be completed earlier than $r(J) + p(J)$, so *Md*

must be busy during the period $[r(J), r(J) + p(J)]$. For the latter case, let $t \geq r(J)$ be the time when J is discarded. During $[r(J), t]$, Md is busy. At time t, Mu is running a job with deadline later than $d(J)$, which implies Md must be busy until $d(J)$. Hence Md is busy during the whole span of J.

In summary, after $r(J)$, Me is busy for a period of at least $d(J_\ell) - p(J) - r(J)$, and Md is busy for a period of at least $p(J)$. Therefore, the safe processing time after $r(J)$ is at least $d(J_\ell) - r(J)$, and the overall safe processing time is at least $d(J_\ell)$. □

4.2 Analysis of N-EDF-Plus(2)

Next, we show that N-EDF-Plus(2) is a 6-processor 1-competitive algorithm for scheduling jobs with importance ratio at most 2. The basic idea of N-EDF-Plus(2) is similar to EDF-MSp — Whenever a job J fails to complete, N-EDF-Plus(2), at any time during the span of J, must have scheduled two jobs that can produce useful work.

Consider any sequence $\mathcal{I}$ of jobs. Let $\mathcal{A}$ and $\mathcal{O}$ be the set of jobs completed by N-EDF-Plus(2) and an optimal offline algorithm respectively. Let $\mathcal{O}_o = \mathcal{O} - \mathcal{A}$. For the purpose of analysis, we consider Me_1 and Md_1 together, and compare the jobs completed by them against $\mathcal{O}$. Similarly, we consider Me_2 and Md_2 together. Precisely, we partition $\mathcal{A}$ into five groups:

- $\mathcal{A}_1$: the set of jobs not in $\mathcal{O}$ and completed in either Me_1 or Md_1.
- $\mathcal{B}_1$: the set of jobs in $\mathcal{O}$ and completed in either Me_1 or Md_1.
- $\mathcal{A}_2$: the set of jobs not in $\mathcal{O}$ and completed in either Me_2 or Md_2.
- $\mathcal{B}_2$: the set of jobs in $\mathcal{O}$ and completed in either Me_2 or Md_2.
- $\mathcal{U}$: the set of jobs completed in either Mu_1 or Mu_2.

Note that $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{B}_1 \cup \mathcal{A}_2 \cup \mathcal{B}_2 \cup \mathcal{U}$, while $\mathcal{O} \subseteq \mathcal{O}_o \cup \mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{U}$. We show the following.

Lemma 11. *(i) $p(\mathcal{A}_1) \geq p(\mathcal{O}_o)$; and (ii) $p(\mathcal{A}_2) \geq p(\mathcal{O}_o)$.*

Once we establish Lemma 11, we can conclude that N-EDF-Plus(2) is 1-competitive.

Theorem 5. $\|\mathcal{A}\| \geq \|\mathcal{O}\|$.

Proof. By Lemma 11, $p(\mathcal{A}_1) + p(\mathcal{A}_2) \geq p(\mathcal{O}_o) + p(\mathcal{O}_o) \geq \|\mathcal{O}_o\|$, so

$$\begin{aligned} \|\mathcal{A}\| &\geq p(\mathcal{A}_1) + p(\mathcal{A}_2) + \|\mathcal{B}_1\| + \|\mathcal{B}_2\| + \|\mathcal{U}\| \\ &\geq \|\mathcal{O}_o\| + \|\mathcal{B}_1\| + \|\mathcal{B}_2\| + \|\mathcal{U}\| \\ &\geq \|\mathcal{O}\| \ . \end{aligned}$$

□

It remains to prove Lemma 11. As the two parts are symmetric, we only show the proof of $p(\mathcal{A}_1) \geq p(\mathcal{O}_o)$. Intuitively, we analyze the behavior of N-EDF-Plus(1) for the set of jobs $\mathcal{I}' = \mathcal{A}_1 \cup \mathcal{B}_1 \cup \mathcal{O}_o$, and show that it is very similar to the schedule produced by N-EDF-Plus(2) for $\mathcal{I}$. Then we make use of a result in the last section to show that $p(\mathcal{A}_1) \geq p(\mathcal{O}_o)$. Precisely, Lemma 12 shows that the scheduling of *Me* and *Md* in N-EDF-Plus(1) with $\mathcal{I}'$ as input is exactly the same as the scheduling of Me_1 and Md_1 in N-EDF-Plus(2) with $\mathcal{I}$ as input. This means that the set of jobs that have ever been processed (and thus completed) by *Me* and *Md* in N-EDF-Plus(1) is exactly $\mathcal{A}_1 \cup \mathcal{B}_1$. The safe processing time of N-EDF-Plus(1) (i.e., the total amount of time when *Me* and *Md* are busy) is at most $p(\mathcal{A}_1) + p(\mathcal{B}_1)$. By Theorem 4, the safe processing time is at least the total processing time of jobs that an optimal offline algorithm can obtain from scheduling $\mathcal{I}'$. Note that the latter is at least $p(\mathcal{B}_1) + p(\mathcal{O}_o)$. Thus, $p(\mathcal{A}_1) + p(\mathcal{B}_1) \geq p(\mathcal{B}_1) + p(\mathcal{O}_o)$ and Lemma 11(i) follows.

Lemma 12. *Consider the schedules produced when* N-EDF-Plus(1) *schedules* $\mathcal{I}'$*, and when* N-EDF-Plus(2) *schedules* $\mathcal{I}$*. At any time, (i) the job scheduled by Me in* N-EDF-Plus(1) *is exactly the same as the job scheduled by* Me_1 *in* N-EDF-Plus(2)*; and (ii) the job scheduled by Md in* N-EDF-Plus(1) *is exactly the same as the job scheduled by* Md_1 *in* N-EDF-Plus(2)*.*

The remainder of the section proves Lemma 12. We first show Lemma 12(i) by proving a stronger version of it.

Lemma 13. *At any time, the* EDF *queues of Me and* Me_1 *contain exactly the same set of jobs.*

Proof. We prove Lemma 13 by induction on the release time of jobs in $\mathcal{I}$. Assume that the EDF queues of *Me* and Me_1 are exactly the same at any time from time 0 to the time just before the release of a job $J \in \mathcal{I}$. As *Me* and Me_1 use the same strategy to schedule jobs in their queues, their schedule have been exactly the same just before J is released. Let X be the set of jobs in these two queues just before J is released. If J is not in $\mathcal{I}'$, it is rejected by both Me_1 and Md_1. Furthermore, J is not released to N-EDF-Plus(1) and cannot be accepted by *Me* either. If J is in $\mathcal{I}'$, the acceptance of J into the EDF queues of *Me* and Me_1 both depend only on whether $X \cup \{J\}$ can be scheduled using EDF. Thus, *Me* and Me_1 make the same decision and their EDF queues remain the same immediately after J is released. □

We prove the second part of Lemma 12 by contradiction. Let t_o be the first time when the jobs scheduled by *Md* and Md_1 are different. By definition, just before t_o, both *Md* and Md_1 are either idle or working on the same job, say J. The next lemma shows that in the latter case, both *Md* and Md_1 must complete J exactly at t_o. That means, at t_o, either one of *Md* and Md_1 schedules a new job and the other is idle, or these processors schedule different new jobs.

Lemma 14. *If Md and* Md_1 *are working on a job J just before* t_o*, then both Md and* Md_1 *complete J at* t_o*.*

Proof. Both Md and Md_1 never preempt jobs. If they both schedule J prior to t_o, one of them, say Md, must complete J at t_o. If Md has completed J by itself (i.e., without using Mu), Md_1, which has worked on J for the same amount of time as Md, also completes J at t_o. Otherwise, J has been an urgent job from the viewpoint of Md and $t_o = d(J)$. Md_1 must complete J at t_o. □

Proof of Lemma 12 (ii). As explained above, at time t_o, one of Md and Md_1 schedules a new job, and the other is either idle or schedules a different new job. Below we further argue that if Md schedules a new job J_o at t_o, then Md_1 must schedule J_o or a job with a later deadline (see Lemma 15). The reverse can also be proven (see Lemma 16). In other words, the jobs scheduled by Md and Md_1 at t_o have the same deadline. As the deadline of a job is assumed to be unique, we conclude that Md and Md_1 schedule the same job at t_o. This contradicts the definition of t_o, and Lemma 12 (ii) follows. □

Lemma 15. *Suppose Md_1 schedules a new job J_o at t_o. Then Md at t_o cannot idle or schedule a job with a deadline earlier than $d(J_o)$.*

Proof. Note that Me_1 does not admit J_o. By Lemma 13, when N-EDF-Plus(1) schedules I', J_o is also not admitted by Me. Thus, N-EDF-Plus(1) must have put J_o into its common pool. Suppose for the sake of contradiction that at t_o, Md either idles or schedules a job with an earlier deadline. Then N-EDF-Plus(1) must have removed J_o from its pool at some time $t < t_o$. This can happen only if J_o is an urgent job at t and there is another urgent job J' with $d(J') > d(J_o)$. J' cannot be picked up by Md before t_o (otherwise Md will be kept busy beyond t_o, contradicting Lemma 14). J' could also be removed from the pool before t_o if there is another urgent job with an even later deadline. Nevertheless, during the interval $[t, t_o]$, the pool always contains an urgent job with a deadline later than $d(J_o)$. N-EDF-Plus(1) at t_o could schedule a job with deadline at least $d(J_o)$ on Md. A contradiction occurs. □

Lemma 16. *Suppose Md schedules a new job J_o at t_o. Then Md_1 at t_o cannot idle or schedule a job with a deadline earlier than $d(J_o)$.*

Proof. Note that upon release, J_o is not admitted into the EDF queue of Me and thus of Me_1. Furthermore, since J_o is in $\mathcal{I}' = \mathcal{A}_1 \cup \mathcal{B}_1 \cup \mathcal{O}_o$, J_o, when scheduled by N-EDF-Plus(2), is also not admitted by Me_2 and must be put into the common pool of N-EDF-Plus(2). Suppose for the sake of contradiction that Md_1 at t_o idles or schedules a job with a deadline before $d(J_o)$. Then J_o must have been removed from the pool at some time $t < t_o$. As J_o, being in $\mathcal{I}'$, cannot be picked up by Md_2, we can deduce that when J_o is removed from the pool at t, J_o must be urgent and the pool contains two other urgent jobs both with a deadline beyond $d(J_o)$. Each of these two urgent jobs may be subsequently replaced by another urgent job with an even later deadline, and so on. Nevertheless, from t onwards till t_o, the pool contains two such urgent jobs until Md_1 or Md_2 schedules any one of them. Note that Md_1 cannot schedule any such urgent job before t_o (otherwise, Md_1 will be kept busy beyond $d(J_o) > t_o$, contradicting Lemma 14). Md_2 can schedule at most one of the two urgent jobs before t_o and then keeps on working on it beyond t_o. Thus, at t_o, the

pool contains at least one urgent job whose deadline is at least $d(J_o)$ and N-EDF-Plus(2) could schedule this job on Md_1. A contradiction occurs. □

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 *Proceedings of the 32th Annual Symposium on Foundations of Computer Science*, pages 101–110, 1991.

[3] P. Berman and C. Coulston. Speed is more powerful than clairvoyance. In *Proceedings of the Sixth Scandinavian Workshop on Algorithm Theory*, pages 255–263, 1998.

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

[5] M. Brehob, E. Torng, and P. Uthaisombut. Applying extra-resource analysis to load balancing. *Journal of Scheduling*, 3(5):273–288, 2000.

[6] M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichý, and N. Vakhania. Preemptive scheduling in overloaded systems. In *Proceedings of the Twenty-ninth International Colloquium on Automata, Languages, and Programming*, pages 800-811, 2002.

[7] M. L. Dertouzos. Control robotics: the procedural control of physical processes. In *Proceedings of International Federation for Information Processing Congress*, pages 807–813, 1974.

[8] J. Edmonds. Scheduling in the dark. *Theoretical Computer Science*, 235(1):109–141, 2000.

[9] B. Kalyanasundaram and K. R. Pruhs. Maximizing job completions online. In *Proceedings of the Sixth European Symposium on Algorithms*, pages 235–246, 1998.

[10] B. Kalyanasundaram and K. R. Pruhs. Eliminating migration in multi-processor scheduling. *Journal of Algorithms*, 38(1):2–24, 2001.

[11] B. Kalyanasundaram and K. R. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[12] G. Koren, E. Dar, and A. Amir. The power of migration in multiprocessor scheduling of real-time systems. *SIAM Journal on Computing*, 30(2):511–527, 2000.

[13] G. Koren and D. Shasha. MOCA: A multiprocessor on-line competitive algorithm real-time system scheduling. *Theoretical Computer Science*, 128:75–97, 1994.

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

[15] T. W. Lam and K. K. To. Trade-offs between speed and processor in hard-deadline scheduling. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 1999.

[16] T. W. Lam and K. K. To. Performance guarantee for online deadline scheduling in the presence of overload. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 755–764, 2001.

[17] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

[18] J. Sgall. On-line scheduling — a survey. In A. Fiat and G. Woeginger, editors, *On-line Algorithms: The State of the Art*, pages 196–231. Lecture Notes in Computer Science, Springer Verlag, 1998.

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

Appendix: Lower bound

In this section we show a lower bound on the number of extra processors required by an online algorithm so as to be 1-competitive against any offline algorithm using one processor. Some techniques used in the proof are inspired by Koren and Shasha's lower bound result on the competitive ratio of multi-processor scheduling algorithms using no additional resources [13].

Theorem 6. *For scheduling jobs with importance ratio k, there is no m-processor 1-competitive algorithm (against one-processor offline schedulers) for all $m \leq \lfloor \log k \rfloor$.*

Let $\mathcal{A}$ be an algorithm using m processors, where $1 < m \leq \lfloor \log k \rfloor$. To ease our discussion, let $\varepsilon = 1/2mk$. Below we describe an adversary which constructs an input sequence to make $\mathcal{A}$ perform poorly. The adversary divides the time into a number of stages. At the beginning of each stage, a fixed set of jobs is released. To ensure that the offline algorithm outperforms $\mathcal{A}$, the adversary controls the number of stages and the time to start each stage.

In each stage, the following $m+1$ jobs are released. All jobs have zero slack time, i.e., their deadlines are exactly the start time of the stage plus their processing time.

Job category	Value density	Processing time	Value
0	1	1	1
1	2	ε	2ε
2	2^2	ε^2	$(2\varepsilon)^2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
m	2^m	ε^m	$(2\varepsilon)^m$

Note that a job of higher category has higher value density, but much smaller processing time, leading to a much smaller value. The importance ratio of the job set is $2^m \leq k$. Since all jobs are tight, if a job is not chosen to be scheduled immediately upon release, it will miss its deadline. Intuitively, it is desirable to schedule jobs of higher value density, so as to get a larger value in a short time. However, doing so risks idling for most of the time if the next stage does not start early enough. With only m processors, $\mathcal{A}$ is forced to abandon at least one of the $m+1$ jobs immediately after it is released. $\mathcal{A}$ suffers if the next stage starts exactly at the deadline of that job, since the jobs of lower categories are of much less value density, while the jobs of higher categories are too short to contribute significant value. We show that in such a case, the sum of values obtained by all the m processors is still less than the value that can be obtained by an offline algorithm. The adversary can thus stop after a sufficiently large number of stages, when it is known that $\mathcal{A}$ cannot match the optimal offline algorithm with the value obtained after the last stage.

We begin by defining when each stage starts. Stage 1 starts at time 0. Immediately after a stage starts, say at time t, the adversary examines the jobs that $\mathcal{A}$ chooses for scheduling. Note that there may be more than one job of each category, since a job may need multiple stages to complete. For $i = \{0, 1, \ldots, m\}$, let n_i be the number of jobs of category i or below scheduled by $\mathcal{A}$. For convenience, we define $n_{-1} = 0$. The adversary finds the minimum number $\alpha \in \{0, 1, \ldots, m\}$ such that $n_\alpha \leq \alpha$. This number is well defined, since $n_m \leq m$. Note that $n_{\alpha-1} > \alpha - 1$, so the number of scheduled jobs of category α is $n_\alpha - n_{\alpha-1} < 1$, i.e., zero. The adversary declares the stage ends at $t + \varepsilon^\alpha$, i.e., the deadline of the category α job released in the stage. The next stage starts immediately, unless the adversary decides to stop.

Fact 5. *During a stage, an offline algorithm can obtain a value of $(2\varepsilon)^\alpha$, by scheduling the category α job in its only processor.*

Lemma 17. *During a stage, $\mathcal{A}$ obtains at most a value of $(2\varepsilon)^\alpha - \varepsilon^\alpha/2$.*

Proof. We separately analyze the value $\mathcal{A}$ obtains for jobs in categories above and below α (recall that no job of category α is scheduled). Let us first consider jobs of categories $\alpha+1$ and above. Since these jobs have deadlines earlier than the category α job, their value is obtained completely during the stage. As we have discussed, jobs of higher category are of less value, so the value of each job is at most $(2\varepsilon)^{\alpha+1}$. With m processors, $\mathcal{A}$ schedules no more than m such jobs, resulting in a total value of at most $m(2\varepsilon)^{\alpha+1} \leq m2^m\varepsilon^{\alpha+1} \leq mk(\varepsilon^\alpha/2mk) = \varepsilon^\alpha/2$.

Next, we consider jobs of categories $\alpha - 1$ and below. These jobs may not necessarily be completed within the stage, and we count only the value associated with the work

done in the stage. By the definition of α, the number of jobs of category i to $\alpha - 1$ is $n_\alpha - n_{i-1} < \alpha - (i-1)$, i.e., at most $\alpha - i$. Thus the total value density is no more than the case when there is one job of each category, with the total value density $1 + 2 + \cdots + 2^{\alpha-1} = 2^\alpha - 1$. The value obtained by $\mathcal{A}$ is thus no more than $(2^\alpha - 1)\varepsilon^\alpha$.

Lemma 17 results immediately by summing the above two parts. □

Note that in each stage, $\mathcal{A}$ lags behind the offline algorithm for an additional amount $\varepsilon^\alpha/2 \geq \varepsilon^m/2$ of value. So if there are $m\lfloor 2/\varepsilon^m + 1 \rfloor$ stages, $\mathcal{A}$ lags behind the offline algorithm by more than m in value. After the last stage, $\mathcal{A}$ can complete no more than m jobs, each job has a value of no more than 1. The offline algorithm thus obtains more value than $\mathcal{A}$, completing the proof of Theorem 6.