

Extra Processors versus Future Information in Optimal Deadline Scheduling

Chiu-Yuen Koo^{*}

Tak-Wah Lam[†]

Tsuen-Wan Ngan[‡]

Kar-Keung To[†]

ABSTRACT

This paper is concerned with the extra-resource analysis of online scheduling algorithms. 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 which are primarily based of 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 help.

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 [18] for a survey). From a theoretical viewpoint, EDF is optimal for scheduling underloaded systems, i.e., whenever there exists a schedule meeting the deadlines of all jobs released, EDF can always do so [6]. 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 studying 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 [3,5,7,10,14,16]. Intuitively, we need 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 [10] are 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 (i.e., optimal) algorithm can be achieved [15].

An alternative to using a faster processor is exploiting multiple unit-speed processors. The scheduling problem however becomes more difficult as jobs may not be parallizable 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 help. In other words, an m -times faster processor can simulate m unit-speed processors, but not vice versa. In this paper we show in the affirmative that multiple unit-speed processors can be used to counteract the lack of future information. More interestingly, the number of processors required for guaranteeing a 1-competitive deadline scheduling algorithm is in the same order of magnitude of the extra speed requirement given in the previous work. Our new result holds no matter job migration among processors is allowed or not. Details are as follows.

Problem definition: In this paper we study the following scheduling problem on a processor, which is also 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. Note that 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. 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 [18].

We analyze the performance of online algorithms with respect to their competitiveness (see, e.g., [4,17]). 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.

^{*}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, kktto}@cs.hku.hk).

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

A 1-competitive algorithm is also said to be *optimal*.

Previous work: The early work of Dertouzos [6] showed that for underloaded systems, the Earliest Deadline First (EDF) strategy is optimal. But in general, no optimal or $O(1)$ -competitive firm-deadline scheduler can exist; indeed, the best competitive ratio has a matching upper bound and lower bound of $(1 + \sqrt{k})^2$ (Baruah et al. [2], Koren and Shasha [13]). To obtain better performance guarantee, we allow online schedulers to use a faster processor. Specifically, we compare 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 [10] 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 double speed); more recently, Lam and To [15] 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. As mentioned before, it is more difficult to exploit multiple unit-speed processors than a faster processor to derive an $O(1)$ - or 1-competitive algorithm. Nevertheless, two results on restricted cases have been known. Baruah [1] considered jobs with uniform value density (i.e., $k = 1$) and gave a 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 [8] 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. Assume that migration is allowed, we give a two-processor optimal algorithm for $k = 1$ (i.e., a job's value is proportional to its computational time), and a $4 \lceil \log k \rceil$ -processor optimal algorithm for general k . Note that the processor bound is asymptotically tight as it is relatively easy to show that for any algorithm that is m -processor optimal, $m \geq \lceil \log k \rceil$ (see the Appendix). Eliminating migration via more processors has been a challenging problem even in the offline setting (e.g., [9, 11]). In this paper, we show that when migration is not allowed, optimality 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: The remainder of this paper is organized as follows. Section 2 shows a new way to enhance EDF, obtaining a two-processor optimal algorithm called EDF-Plus for $k = 1$. Section 3 extends EDF-Plus to handle the cases for general k . For the sake of completeness, the $\lceil \log k \rceil$ lower bound is given in the appendix. Section 4 gives a non-migratory version of EDF-Plus, which is three-processor optimal for $k = 1$ and $6 \lceil \log k \rceil$ -processor optimal

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, we denote $p(\mathcal{S})$ as the total processing time of the jobs in \mathcal{S} . Without loss of generality, for a system with importance ratio k , we assume that all jobs have value densities in the range $[1, k]$. Furthermore, we assume that jobs have distinct deadlines (ties can be broken using identification numbers of jobs).

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 is tested 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 by their deadlines using an EDF schedule.

2. THE EDF-Plus ALGORITHM

In this section we discuss a new algorithm called EDF-Plus which is two-processor optimal for scheduling jobs with value density equal to one. It is known that EDF (and EDF-AC) is one-processor optimal for underloaded systems [6]. Yet this is not true for overloaded systems. Intuitively, it is too difficult for an online algorithm to select the right jobs so as to maximize the overall processing time. For example, EDF-AC would make a mistake in rejecting a long job due to an earlier admission of a shorter job with close 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 in another processor temporarily; after a while, the remaining processing time will get smaller and hopefully, the job can get admitted by EDF-AC again. Thus, the enhanced EDF-AC will be more productive.

The above observation leads us to use two processors, denoted M_e and M_p , in the algorithm EDF-Plus. 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 scheduling a rejected job temporarily. 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. 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.

- **maximum processing time:** M_p only works on the job with the longest processing time as soon as it is rejected by M_e . All other rejected jobs from M_e are given up immediately.
- **zero slack time:** M_p works on a rejected job only when it has zero slack time (i.e., deadline = current time + processing time); in case there are more than one such

jobs, preference will be given to the one with the latest deadline.

Alg. 1 gives the details of EDF-Plus based on the first strategy. 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.

THEOREM 1. *For scheduling jobs with uniform value density, EDF-Plus is two-processor optimal.*

In the remainder of this section, we prove Theorem 1 by contradiction. Assume that EDF-Plus is not optimal for some job sequence. Let \mathcal{I} be the one containing the fewest jobs. Without loss of generality, suppose the first job is released at time 0. We first establish that for such \mathcal{I} , EDF-Plus keeps M_e busy over one continuous period (see Lemma 3). Then we show in Lemma 4 an interesting property of the job J_ℓ in \mathcal{I} that has the latest deadline. Using these lemmas, we can show that the total processing time of jobs completed by M_e is more than $d(J_\ell)$ (see Lemma 5). 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 $d(J_\ell)$. Therefore, this contradicts that EDF-Plus is not optimal for \mathcal{I} and we complete the proof of Theorem 1.

FACT 2. *At any time, if M_e is idle, then AC-Q (the queue storing all admitted jobs to be completed by M_e) is empty and M_p is idle.*

LEMMA 3. *In the course of scheduling \mathcal{I} , 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. Partition \mathcal{I} into two parts, one for jobs with release time before t_ℓ and one for the rest. Since EDF-Plus is not optimal for input \mathcal{I} , at least one of the two parts gives a job sequence that EDF-Plus is not optimal. This contradicts that \mathcal{I} contains the fewest jobs. \square

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

DEFINITION 1. *Consider any time t when M_e rejects a job J (see line 4 and 14 in Alg. 1). That is, if M_e uses EDF to schedule J together with the jobs admitted before t , some jobs J_o miss their deadlines. Any such jobs J_o is said to repudiate J at t . Note that a job can only repudiate itself or jobs with earlier deadlines.*

LEMMA 4. *In the course of scheduling \mathcal{I} , there is at least one time when J_ℓ repudiates a job.*

PROOF. Suppose on the contrary that J_ℓ never repudiates any job. J_ℓ and any other jobs do not repudiate J_ℓ when it is released; thus, J_ℓ must be admitted for 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 only 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 optimal. This contradicts that \mathcal{I} contains the fewest jobs. \square

LEMMA 5. *The value obtained by EDF-Plus in scheduling \mathcal{I} is more than $d(J_\ell)$.*

PROOF. By Lemma 4, J_ℓ repudiates some job J at some time t . Note that $r(J_\ell) \leq t \leq d(J_\ell)$ and M_e must be busy at time t . Furthermore, by Lemma 3, M_e is busy throughout the period $[0, t]$. By definition, at time t , using EDF to schedule the jobs currently found in AC-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 rejected job 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 processor 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 for scheduling \mathcal{I} is more than $d(J_\ell) - p(J) + p(J) = d(J_\ell)$. \square

3. OPTIMAL SCHEDULING FOR NON-UNIFORM VALUE DENSITIES

In this section we first present an algorithm called EDF-MSp which is 4-processor optimal 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 optimal algorithm. EDF-MSp uses four processors, 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 7 or line 9 in Alg. 1), the job is passed to Band 2.

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. Recall that EDF-Plus guarantees that $p(\mathcal{A}_1) \geq p(\mathcal{O})$. Though jobs in \mathcal{O} may have higher value, the importance ratio is at most two and $\|\mathcal{O}\|$, the total value of \mathcal{O} , is at most $2p(\mathcal{O})$. Optimality can be achieved if the Band 2 processors can complete a subset \mathcal{A}_2 of jobs discarded by EDF-Plus with sufficient processing time, say, $p(\mathcal{A}_2) \geq p(\mathcal{O})$. Then we can conclude that $\|\mathcal{A}_1\| + \|\mathcal{A}_2\| \geq \|\mathcal{O}\|$. Yet providing

```

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

```

Algorithm 1: The EDF-Plus algorithm

such a guarantee on $p(\mathcal{A}_2)$ seems to be very difficult. (This is mainly due to the fact that some jobs passed to Band 2 might have been released for a long time.) In fact, our algorithm takes advantage of a less demanding requirement, namely, $p(\mathcal{A}_2) \geq p(\mathcal{O} - \mathcal{A}_1 - \mathcal{A}_2)$.

LEMMA 6. *Let $\mathcal{O}' = \mathcal{O} - \mathcal{A}_1 - \mathcal{A}_2$. Suppose $p(\mathcal{A}_2) \geq p(\mathcal{O}')$. Then $\|\mathcal{A}_1\| + \|\mathcal{A}_2\| \geq \|\mathcal{O}\|$.*

PROOF. Denote $\mathcal{S}_1 = \mathcal{A}_1 \cap \mathcal{O}$ 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, $\|\mathcal{A}_1\| + \|\mathcal{A}_2\| \geq \|\mathcal{S}_1\| + \|\mathcal{S}_2\| + p(\mathcal{A}'_1) + p(\mathcal{A}'_2) \geq \|\mathcal{O}\|$. \square

Below we derive an algorithm called MSp for Band 2 so that $p(\mathcal{A}_2) \geq p(\mathcal{O}')$, where $\mathcal{O}' = \mathcal{O} - \mathcal{A}_1 - \mathcal{A}_2$. First of all, we note that job J passed to MSp is discarded by EDF-Plus either at $r(J)$ or strictly after $r(J)$. For the latter case, we observe the following property.

FACT 7. *If a job J is discarded by EDF-Plus at time $t > r(J)$, it has been running on M_p during $[r(J), t]$.*

We denote the two processors of MSp as M_r and M_d . Details of MSp are shown in Alg. 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 early 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. The

ensures M_r to be 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. Similar to the two processors of EDF-Plus, M_r and M_d have the following relationship.

FACT 8. *At any time, if M_r is idle, then SLACK_Q is empty and M_d is idle.*

The crux of the analysis of Band 2 is captured by the following theorem. Recall that with respect to a given job sequence \mathcal{I} , \mathcal{A}_1 and \mathcal{A}_2 denote the set of jobs completed by Band 1 and Band 2, respectively, and \mathcal{I}' denotes the set of jobs passed to but not completed by Band 2 (i.e., $\mathcal{I}' = \mathcal{I} - \mathcal{A}_1 - \mathcal{A}_2$). Furthermore, we need the following definition.

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} .*

THEOREM 9. $p(\mathcal{A}_2) \geq sp(\mathcal{I}')$.

Before proving Theorem 9, we note that Theorem 9 guarantees that EDF-MSp is a four-processor optimal algorithm for scheduling jobs with value densities in the range $[1, 2]$.

COROLLARY 10. *(i) $p(\mathcal{A}_2) \geq sp(\mathcal{O}')$; (ii) $\|\mathcal{A}_1\| + \|\mathcal{A}_2\| \geq \|\mathcal{O}\|$.*

PROOF. As $\mathcal{O}' \subseteq \mathcal{I}'$ and OPT schedules at most one job at a time, we have $p(\mathcal{O}') \leq sp(\mathcal{I}')$. By Theorem 9, $p(\mathcal{A}_2) \geq sp(\mathcal{I}') \geq p(\mathcal{O}')$. Then, by Lemma 6, we conclude that $\|\mathcal{A}_1\| + \|\mathcal{A}_2\| \geq \|\mathcal{O}\|$. \square

```

(1) Initialization: SLACK_Q  $\leftarrow \emptyset$  // A job in SLACK_Q waits until its slack time is zero
(2)
(3) When job  $J$  is passed to band 2:
(4)   if  $M_r$  is idle or  $r(J) < r(J_r)$  where  $J_r$  is the job running in  $M_r$ 
(5)     SLACK_Q  $\leftarrow$  SLACK_Q  $\cup \{J_r\}$ 
(6)      $M_r$  runs  $J$ 
(7)   else
(8)     SLACK_Q  $\leftarrow$  SLACK_Q  $\cup \{J\}$ 
(9)
(10) When  $M_r$  completes job  $J$ :
(11)   if SLACK_Q  $\neq \emptyset$ 
(12)     SLACK_Q  $\leftarrow$  SLACK_Q  $- \{J'\}$  where  $J'$  is chosen arbitrarily from SLACK_Q
(13)      $M_r$  runs  $J'$ 
(14)   else if  $M_d$  is working on a job  $J_d$ 
(15)      $M_r$  runs  $J_d$  //  $M_d$  becomes idle
(16)
(17) When job  $J \in$  SLACK_Q has zero slack time:
(18)   SLACK_Q  $\leftarrow$  SLACK_Q  $- \{J\}$ 
(19)   if  $M_d$  is idle or  $d(J) > d(J_d)$  where  $J_d$  is the job running in  $M_d$ 
(20)      $M_d$  runs  $J$ ;  $J_d$  is discarded
(21)   else
(22)      $J$  is discarded

```

Algorithm 2: MSp — the algorithm for Band 2.

We prove Theorem 9 via the following three lemmas.

LEMMA 11. *Let J be a job passed to Band 2 at time $t > r(J)$. Then any job J' with $r(J') \leq r(J)$, if passed to Band 2, must be passed on or before $r(J)$.*

PROOF. Assume on the contrary that there is a job J' such that $r(J') \leq r(J)$ and J' is passed to Band 2 at time $t' > r(J)$. Consider the nonempty interval $[r(J), \min(t, t')]$. By applying Fact 7 to J and J' , we obtain the contradiction that M_p has been running J and J' during this interval. \square

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 12. *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 EDF-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 7, J' is running on M_p during the period $[r(J'), t_2]$. By Lemma 11, 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 . \square

LEMMA 13. *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., $[r(J), d(J)]$.*

PROOF. By Lemma 12, it suffices to show that M_r is busy during the span of J . We divide the span into three periods (which may not all exist) 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 should have been completed or discarded by t ; otherwise M_r should 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 11, 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 put into SLACK_Q at least once. Consider the period from t_o to the last time t_ℓ when J is removed from SLACK_Q for consideration of M_d . At any time within this period, J is in SLACK_Q or is processed by M_d or M_r . In the first two cases, M_r cannot be idle because of Fact 8 (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 8, M_r must be busy with some other job. \square

We are now ready to prove Theorem 9, i.e., $p(\mathcal{A}_2) \geq sp(\mathcal{I}')$.

PROOF OF THEOREM 9. First of all, $p(\mathcal{A}_2)$ is at least the total time during which EDF-MSp is productive on \mathcal{A}_2 . Lemma 13 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, $p(\mathcal{A}_2) \geq sp(\mathcal{I}')$. \square

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

THEOREM 14. *For scheduling jobs with value densities in the range $[1, k]$ where $k > 1$, there exists a $4 \lceil \log k \rceil$ -processor optimal algorithm.*

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 4 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 $[1, k]$. \square

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 optimal for scheduling jobs with uniform value density (i.e., $k = 1$). Then we show that N-EDF-Plus(2) is 6-processor optimal for scheduling jobs with importance ratio at most 2. Using the technique in Theorem 14, we can make use of N-EDF-Plus(2) to construct a $6 \lceil \log k \rceil$ -processor optimal 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 give J a second chance by scheduling it on any available Mu_j temporarily. I.e., each Mu_j is using the zero slack time strategy and a job running on an Mu_j will try to migrate to an Md_i whenever an Md_i completes a job. 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 the deadlines).

Note 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 eliminated by switching the role of the two processors. Thus, N-EDF-Plus(η) is non-migratory in nature.

Below 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 Me of EDF-Plus so as to avoid any job migration to Me .

By construction, 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 optimality of N-EDF-Plus(1) is based on the following theorem.

THEOREM 15. *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 15 is analogous to the proof of EDF-Plus being optimal. Assume on the contrary that Theorem 15 fails for some job sequences. Let \mathcal{I} be the one 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. Using the techniques in Lemmas 3 and 4, we can prove the following two claims.

CLAIM 16. *In the course of scheduling \mathcal{I} , N-EDF-Plus(1) is busy over exactly one continuous period.*

CLAIM 17. *There exists a moment when J_ℓ , the latest deadline job in \mathcal{I} , repudiates some job.*

Based on the above two claims, we can prove the following lemma, which shows that N-EDF-Plus(1) can complete jobs with a total time at least $d(J_\ell)$. Jobs in \mathcal{I} can be processed with the period $[0, d(J_\ell)]$ and any offline algorithm, using one processor, can obtain a value at most $d(J_\ell)$. This contradicts N-EDF-Plus(1) is not optimal for \mathcal{I} , and we complete the proof of Theorem 15.

LEMMA 18. *The safe processing time is at least $d(J_\ell)$.*

PROOF. By Claim 17, J_ℓ repudiates some job J at $r(J)$. Me must be busy at $r(J)$. By Claim 16, 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 later 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 $d(J_\ell)$. \square

Next, we show that N-EDF-Plus(2) is a 6-processor optimal 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 OPT, respectively. Let $\mathcal{O}_o = \mathcal{O} - \mathcal{A}$. Furthermore, we partition \mathcal{A} into five groups according to the processors where jobs are completed.

- \mathcal{U} : the set of jobs completed using only Mu_1 or Mu_2 .
- \mathcal{A}_1 : the set of jobs completed in either Me_1 or Md_1 but not completed by OPT.
- \mathcal{B}_1 : the set of jobs completed in either Me_1 or Md_1 , as well as completed by OPT.
- \mathcal{A}_2 : the set of jobs completed in either Me_2 or Md_2 , but not completed by OPT.
- \mathcal{B}_2 : the set of jobs completed in either Me_2 or Md_2 , as well as completed by OPT.

Note that $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{B}_1 \cup \mathcal{A}_2 \cup \mathcal{B}_2 \cup \mathcal{U}$, while $\mathcal{O} \subseteq \mathcal{O}_o \cup \mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{U}$. To prove the optimality of N-EDF-Plus(2), it suffices to show the following.

LEMMA 19. (i) $p(\mathcal{A}_1) \geq p(\mathcal{O}_o)$; and (ii) $p(\mathcal{A}_2) \geq p(\mathcal{O}_o)$.

By Lemma 19, $p(\mathcal{A}_1) + p(\mathcal{A}_2) \geq p(\mathcal{O}_o) + p(\mathcal{O}_o) \geq \|\mathcal{O}_o\|$, and

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

Therefore, we can conclude the following theorem, showing that N-EDF-Plus(2) is optimal.

THEOREM 20. $\|\mathcal{A}\| \geq \|\mathcal{O}\|$.

It remains to prove Lemma 19. As the two parts are symmetric, we only show the proof of $p(\mathcal{A}_1) \geq p(\mathcal{O}_o)$. Intuitively, we observe that the scheduling of N-EDF-Plus(2) for \mathcal{I} resembles the schedule produced by N-EDF-Plus(1) for the jobs in $\mathcal{A}_1 \cup \mathcal{B}_1 \cup \mathcal{O}_o$, and hence we can make use of the result in the last section to show that $p(\mathcal{A}_1) \geq p(\mathcal{O}_o)$. Define $\mathcal{I}' = \mathcal{A}_1 \cup \mathcal{B}_1 \cup \mathcal{O}_o$. Precisely, we consider how N-EDF-Plus(1) schedules \mathcal{I}' , and compare it to the scheduling of these jobs when N-EDF-Plus(2) schedules \mathcal{I} . Lemma 21 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. Since $p(\mathcal{A}_1 \cup \mathcal{B}_1)$ is no less than the safe processing time of N-EDF-Plus(1), Theorem 15 guarantees that $p(\mathcal{A}_1 \cup \mathcal{B}_1)$ is at least the longest processing time an optimal offline algorithm can get from \mathcal{I}' , which is at least $p(\mathcal{B}_1 \cup \mathcal{O}_o)$. Part (i) of Lemma 19 follows.

LEMMA 21. *At any time, the jobs scheduled by Me (resp. Md) when N-EDF-Plus(1) schedules \mathcal{I}' is exactly the same as the jobs scheduled by Me_1 (resp. Md_1) when N-EDF-Plus(2) schedules \mathcal{I} .*

PROOF. See Appendix II. \square

5. 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. 32th FOCS*, 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, Cambridge, United Kingdom, 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] J. Edmonds. Scheduling in the dark. In *Proc. 31st ACM STOC*, pages 179–188, 1999.
- [8] B. Kalyanasundaram and K. R. Pruhs. Maximizing job completions online. In *Proc. 6th ESA*, pages 235–246, 1998.
- [9] B. Kalyanasundaram and K. R. Pruhs. Eliminating migration in multi-processor scheduling. In *Proc. 10th ACM-SIAM SODA*, pages 499–506, 1999.

- [10] B. Kalyanasundaram and K. R. Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.
- [11] G. Koren, E. Dar, and A. Amir. The power of migration in multiprocessor scheduling of real-time systems. *SIAM J. Comput.*, 30(2):511–527, 2000.
- [12] G. Koren and D. Shasha. MOCA: A multiprocessor on-line competitive algorithm real-time system scheduling. *Theoretical Computer Science*, 128:75–97, 1994.
- [13] 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.
- [14] 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.
- [15] T. W. Lam and K. K. To. Performance guarantee for online deadline scheduling in the presence of overload. In *Proc. 12th ACM-SIAM SODA*, pages 755–764, 2001.
- [16] 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.
- [17] 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.
- [18] J. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo. *Deadline scheduling for real-time systems: EDF and related algorithms*. Kluwer Academic Publishers, Boston, Mass., 1998.

Appendix I — 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 optimal 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 [12].

THEOREM 22. *For scheduling with importance ratio k , there is no m -processor optimal algorithm with $m < \lceil \log k \rceil$.*

Let \mathcal{A} be an algorithm using m processors, where $1 < m < \lceil \log k \rceil$. 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 $i = \{0, 1, \dots, m\}$, 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 $\alpha \in \{0, 1, \dots, m\}$ 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 23. *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 24. *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 can only be completed after the end of 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 $1 + 2 + \dots + 2^{\alpha-1} = 2^\alpha - 1$. The value obtained by \mathcal{A} is thus no more than $(2^\alpha - 1)\varepsilon^\alpha$.

Lemma 24 results immediately by summing the above two parts. \square

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 22.

Appendix II — Proof of Lemma 21

In Lemma 25, we show that at any time, the EDF queues of Me and Me_1 are the same. It follows that if Lemma 21 is false, the jobs scheduled by Md and Md_1 must be different at some time. Assume such time exists and let t be the earliest time. We are going to explore a property of the schedules at t in Lemma 26. By making use this property, we would be able to disprove the existence of t and completes the proof of Lemma 21.

LEMMA 25. *At any time, the EDF queues of Me and Me_1 are exactly the same.*

PROOF. Suppose otherwise. Let t' be the earliest time such that the EDF queues are different. Then the EDF queues of Me and Me_1 just before t are the same, say both being \mathcal{Q} . The difference must due to the release of a job J at t' such that J is accepted by one to EDF queue but not both of Me and Me_1 . For the case $J \in \mathcal{I}'$, if J is not accepted by Me_1 , scheduling $\mathcal{Q} \cup \{J\}$ by EDF causes some deadline to be missed, so it is not accepted by Me , and vice versa. For the case $J \notin \mathcal{I}'$, this is absurd, since by definition, $\mathcal{I}' = \mathcal{A}_1 \cup \mathcal{B}_1 \cup \mathcal{O}_o$, J is not scheduled by Me_1 . \square

LEMMA 26. *At time t , both Md and Md_1 are free to schedule a new job. In other words, Md and Md_1 are either both idle prior to t , or both have completed the same job at t .*

PROOF. By definition of t , the schedules of Md and Md_1 are the same prior to t . Thus, they are both idle or scheduling the same job J prior to t . If they both schedule J prior to t , one of them, say Md , must have completed J at t as Md and Md_1 never preempt jobs. If $t = d(J)$, both processors must have completed the job at t . Otherwise, Md must have completed it without the help of Mu , and thus Md_1 should also complete it at t . \square

PROOF OF LEMMA 21. From Lemma 25, jobs scheduled by Me and Me_1 are the same at any time. It follows that schedules of Md and Md_1 are different at t . By Lemma 26, at least one of Md and Md_1 schedules a new job at t such that it is not scheduled by the counterpart. Denote J as the job newly scheduled by Md or Md_1 at t (choose the one with a later deadline if more than one). Note that $t < d(J)$. In the rest of the proof, we show that there is a job with a later deadline than $d(J)$ in the pool of the counterpart schedule. This contradicts to the definition of J , since the counterpart Md or Md_1 should schedule a job with a later deadline.

First consider the case when J corresponds to the job scheduled by Md_1 at t . Since Md either idles or schedules a job with an earlier deadline, N-EDF-Plus(1) must have removed J from its pool at some time $t' < t$ due to an urgent job J' , where $d(J') > d(J)$. J' might be further removed due to other urgent jobs with later deadlines, yet we can guarantee that there is an urgent job of deadline later than $d(J)$ in the pool of N-EDF-Plus(1) during $[t', t]$. By definition, urgent jobs cannot be completed before their deadlines. Up to time t , that job has not been completed and must still be in the pool.

We then turn to the remaining case, when Md schedules J at t . Note that J is not in \mathcal{A}_2 or \mathcal{B}_2 , so it cannot be taken by Md_2 . The proof is similar to the previous case, except that there are two urgent jobs of deadlines later than $d(J)$ in the pool of N-EDF-Plus(2) at time t' . These jobs might further be removed due to other urgent jobs rejected by Me_1 and Me_2 . However, those new urgent jobs must have later deadlines than the ones being removed. Therefore, there must still be two urgent jobs rejected by Me_1 and Me_2 which have later deadline than $d(J)$ at any time during $[t', d(J)]$, in particular at time t . At most one of them can be migrated to Md_2 . Therefore, there is at least one urgent job with its deadline later than $d(J)$ in the pool of N-EDF-Plus(2). \square