

# Performance Guarantee for EDF under Overload\*

Tak-Wah Lam<sup>†‡</sup>   Tsuen-Wan Johnny Ngan<sup>§‡</sup>   Kar-Keung To<sup>†</sup>

October 17, 2003

**Abstract.** Earliest deadline first (EDF) is a widely used algorithm for online deadline scheduling. It has been known for long that EDF is optimal for scheduling an underloaded, single-processor system; recent results on the extra-resource analysis of EDF further revealed that EDF when using moderately faster processors can achieve optimal performance in the underloaded, multi-processor setting. This paper initiates the extra-resource analysis of EDF for overloaded systems, showing that EDF supplemented with a simple form of admission control can provide a similar performance guarantee in both the single and multi-processor settings.

Key words: online algorithms, extra-resource analysis, firm deadline scheduling, earliest deadline first.

## 1 Introduction

This paper is concerned with online algorithms for scheduling jobs with deadlines. A typical example is the earliest deadline first (EDF) algorithm, which has been widely used in real-time systems (see [15] for a survey). It is well-known that EDF is optimal for a single processor system that is underloaded, i.e., whenever there exists an offline schedule meeting the deadlines of all jobs released, EDF can always do so [7]. However, when the system is overloaded or involves more than one processor, EDF has no performance guarantee in the sense that its performance cannot match or even be competitive against the optimal offline algorithm. Indeed, in most settings, no online algorithm has this sort of performance guarantee [2, 8]. In recent years, a plausible approach to studying performance guarantee for online scheduling without restricting the inputs is to allow the online scheduler to use faster processors [1, 3, 5, 9, 10, 13, 14]. Intuitively, we want to

---

\*Results in this paper have appeared in a preliminary form in the Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, 2001 and Proceedings of the 15th International Parallel and Distributed Processing Symposium, 2001

<sup>†</sup>Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong (`{twlam, kkto}@cs.hku.hk`).

<sup>‡</sup>This research was supported in part by Hong Kong RGC Grant HKU-7024/01E.

<sup>§</sup>Department of Computer Science, Rice University, Houston, TX 77005, USA (`twngan@cs.rice.edu`).

study how effectively can faster processors compensate the online scheduler for the lack of future information. Phillips et al. [14] were able to extend the optimality of EDF to the underloaded, multiprocessor setting by allowing the online scheduler to use double-speed processors.

In this paper we study deadline scheduling for overloaded systems, in which even the offline adversary may not be able to meet the deadlines of all jobs and job deadlines are firm in the sense that it is useless to complete a job after its deadline. Our aim is to attain optimality with the meaning of matching the offline adversary on the total work of jobs that are completed by their deadlines. Scheduling under overload is more difficult as the online algorithm has to be smart in selecting the right jobs to schedule. Even for the single processor setting, no online algorithm using a normal speed or faster processor is known to be optimal. In fact, this paper presents a lower bound result that any online algorithm using a processor with speed factor less than the golden ratio ( $\approx 1.618$ ) cannot be optimal. This result should be contrasted with the fact that in the underloaded setting, EDF using a normal speed processor is already optimal [7]. If we do not insist on optimal scheduling, it is possible to have an online algorithm that can approximate the total work achieved by the adversary within a constant ratio when using moderately faster processors [10].

In this paper, we resolve in the affirmative that in the overloaded setting, EDF with moderately faster processors can achieve optimality for both single-processor and multiprocessor systems. The speed requirement is double and triple, respectively. Furthermore, we study the general case when jobs carry arbitrary values (or weights) that are independent of their processing times. The aim of a scheduler is to maximize the total value of the jobs meeting their deadlines. This scheduling problem is obviously more general (it is referred to as the firm deadline scheduling problem in the literature). Our main result is that EDF when given faster processors can still guarantee optimality in this more general setting.

**Problem and definitions:** We are given a pool of  $m \geq 1$  processors and a stream of jobs which are released at arbitrary times, with varying work (processing time) requirements and deadlines. Every job is sequential in nature and can be processed by at most one processor at a time. Preemption is allowed. For a hard deadline (or equivalently, an underloaded) system, the offline adversary can schedule all jobs to meet the deadlines, and we also aim at an online algorithm that does so. In general, the system may be overloaded, i.e., even the offline adversary may not be able to meet all the deadlines. In this case, our aim is to maximize the total work of jobs completed before the deadlines. There is no credit awarded to completing a job after its deadline.

In this paper, we consider overloaded systems and on-line algorithms equipped with processors that are faster than those available to the offline adversary. For any real number  $s \geq 1$ , a processor is said to be speed- $s$  if it can process one unit of work in time  $1/s$ . Furthermore, an online algorithm is said to be speed- $s$  *optimal* if the algorithm when using  $m$  speed- $s$  processors can match the total work of jobs completed by any offline

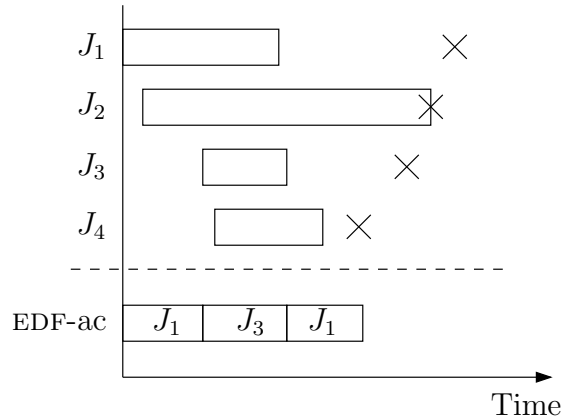


Figure 1: An example of EDF-ac scheduling. Note that  $J_2$  and  $J_4$  are not admitted for scheduling, and  $J_3$  preempts  $J_1$  due to its earlier deadline.

algorithm using  $m$  speed-1 processors. Note that an online algorithm that is optimal for overloaded systems is also optimal for underloaded systems.

**EDF with admission control:** The way EDF schedules jobs on  $m \geq 1$  processors is as follows: Whenever a job is released or completed, EDF examines the remaining jobs to be completed. If there are at most  $m$  such jobs, each job is scheduled to run in one processor; otherwise, EDF chooses  $m$  jobs with earliest deadlines for execution. When the system is overloaded EDF may be too aggressive in preempting jobs; thus, EDF is often supplemented with some kind of admission control. We consider the following simple form of admission control. Upon release, every job has to go through a feasibility test in order to get admitted for EDF scheduling. The test simply checks whether the new job together with the previously admitted jobs can all be completed before their deadlines using an EDF schedule. See Figure 1 for an example. In this paper, EDF when supplemented with the above form of admission control is referred to as EDF-ac.

Our first result on scheduling under overload is that for a single processor system, EDF-ac is speed-2 optimal. We also show that no algorithm can be speed- $s$  optimal if  $s < (1 + \sqrt{5})/2 \approx 1.618$ . For scheduling with  $m \geq 2$  processors, we find that speed-3 processors suffice to guarantee EDF-ac to be optimal, no matter how big  $m$  is. This is the first result attaining optimality for scheduling in overloaded multiprocessor systems.

**Scheduling jobs with values:** In general, the credit awarded to completing a job may not be related to its work. Jobs are given arbitrary values and the objective becomes to maximize the total value of the jobs completed by their deadlines. Define the value density of a job to be the ratio of its value to its required work. The importance ratio of a system refers to the ratio of the largest possible value density to the smallest possible one. In the special case when the importance ratio is one (i.e., all jobs have uniform value density), maximizing the total value is equivalent to maximizing the total work.

For maximizing total value, there are several online algorithms in the literature, whose performance has been rigorously analyzed. An online algorithm  $A$  is said to be  $c$ -competitive for some  $c > 0$  if, for any job sequence,  $A$  can attain at least a fraction  $1/c$  of the total value obtained by any offline algorithm [4]. Furthermore,  $A$  is said to be speed- $s$   $c$ -competitive if  $A$  is allowed to use processors  $s$  times faster than the offline adversary. In the single processor setting, Koren and Shasha [12] have given a  $(1 + \sqrt{k})^2$ -competitive algorithm, where  $k$  is the importance ratio. Kalyanasundaram and Pruhs [10] later showed that a speed- $O(1)$  processor is sufficient to improve the competitive ratio to a constant. In particular, their algorithm is speed-2 32-competitive. In the multiprocessor setting, Koren et al. [11] have given a  $(1 + m(k^{1/\psi} - 1))$ -competitive algorithm<sup>1</sup>, where  $\psi = m \ln k / 2(\ln k + 1)$ ; no algorithm has been known to exploit faster processors to achieve  $O(1)$ -competitiveness.

Note that none of the above algorithms achieves optimality, i.e., matching the total value achieved by the offline adversary. In this paper, we show that the complication due to varying value densities can again be offset by extra speed; precisely, EDF-ac is speed- $(k + 1)$  optimal in the single processor setting and speed- $(k + 2)$  optimal in the multiprocessor setting. When  $k$  is large, a simple adaptation of EDF-ac can reduce the speed factor to  $O(\log k)$ . The idea is to divide the processors into  $\lceil \log k \rceil$  groups such that each group handles jobs with a smaller range of densities.

Note that the lower bound of  $(1 + \sqrt{5})/2$  on the speed requirement for achieving optimal scheduling on a single processor is still valid in this general setting. Chrobak et al. [6] have recently given a better lower bound for jobs with varying value densities. In particular, they showed that the speed requirement is at least  $2 - 1/k$ , where  $k$  is the importance ratio. We conjecture that even for jobs with uniform value densities, the lower bound should approach 2.

**Organization of the paper:** The remainder of the paper is organized as follows. Section 2 serves as a warm-up, showing that in the single processor setting, EDF-ac is speed-2 optimal for scheduling jobs with uniform job density. Section 3 gives a lower bound on the speed factor to achieve optimality in this setting. Section 4 considers the general case of scheduling jobs with non-uniform value densities on multiprocessors. Note that the speed factor does not depend on the number of processors. We show that EDF-ac is speed- $(k + 2)$  optimal, where  $k$  is the importance ratio. In Section 5, we adapt EDF-ac so as to reduce the speed factor to  $O(\log k)$ .

**Notation:** In the rest of the paper, whenever we refer to a sequence of jobs, we assume that the first job is released at time 0. The release time, work requested, and deadline of a job  $J$  are denoted by  $r(J)$ ,  $p(J)$ , and  $d(J)$  respectively. We measure  $p(J)$  in terms of the time required to process  $J$  on a speed-1 processor. We only consider jobs satisfying  $p(J) \leq d(J) - r(J)$ . Without loss of generality, for a system with an importance ratio of  $k$ , we assume that jobs have value densities in the range  $[1, k]$ . Furthermore, we assume that jobs have distinct deadlines (ties are broken using the job IDs).

---

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

Suppose we schedule a sequence  $L$  of jobs on  $m \geq 1$  processors using EDF. The schedule in which jobs of  $L$  are processed is referred to as the EDF schedule of  $L$ . In this schedule, every job of  $L$  will get processed to completion, but the deadline may not be met. On the other hand, in the EDF-ac schedule of  $L$ , which is the schedule based on EDF-ac, not every job will receive processing time, but a job, once scheduled, will be completed by its deadline. Let  $\mathcal{E}(L) \subseteq L$  denote the set of jobs that EDF-ac can complete by their deadlines. It is worth-mentioning that the EDF-ac schedule of  $L$  is identical to the EDF schedule of  $\mathcal{E}(L)$ .

## 2 Optimal work for uni-processor scheduling

In this section we investigate online scheduling on a single processor. We show that increasing the processor speed by a factor of two suffices to guarantee that EDF-ac matches the total work achieved by any offline algorithm (using a speed-1 processor). In other words, when all jobs have the same value density, EDF-ac is speed-2 optimal.

For any sequence  $L$  of jobs, let  $\mathcal{E}(L) \subseteq L$  be the set of jobs that EDF-ac using a speed-2 processor can complete, and let  $\mathcal{O}(L) \subseteq L$  be the set of jobs that an optimal offline algorithm using a speed-1 processor can complete. Denote by  $\|L\|$  the total work (i.e. processing time) of the jobs in  $L$ .

**Theorem 1** *For any sequence  $L$  of jobs,  $\|\mathcal{E}(L)\| \geq \|\mathcal{O}(L)\|$ .*

We prove Theorem 1 by contradiction. Suppose that there exists a job sequence  $L$  such that  $\|\mathcal{O}(L)\| > \|\mathcal{E}(L)\|$ . Let us consider  $L$  to be such a job sequence containing the fewest jobs.

If jobs in  $\mathcal{E}(L)$  are scheduled to run in two or more continuous periods, we can immediately derive a contradiction since one of such periods contains a smaller job sequence violating Theorem 1. Below, we assume that all jobs in  $\mathcal{E}(L)$  are scheduled in one continuous period. Denote by  $\ell$  the length of this period. A job in  $L$  is said to be a *late-dead* job if its deadline is after  $2\ell$ ; otherwise, it is called an *early-dead* job. We will show a contradiction that the early-dead jobs form a proper subset of  $L$  violating Theorem 1. The contradiction stems from an interesting property of  $\mathcal{E}(L)$ : any *late-dead* job (with deadline after  $2\ell$ ), if present in  $L$ , must be admitted into  $\mathcal{E}(L)$ , and would not affect the admittance of any *early-dead* jobs (with deadline at or before  $2\ell$ ) released subsequently. The latter means that even if we remove all late-dead jobs from  $L$ , we cannot schedule more early-dead jobs.

**Lemma 2** *Let  $L_e$  and  $L_t$  be the sets of early-dead and late-dead jobs in  $L$ , respectively. Then  $L_t \subseteq \mathcal{E}(L)$  and  $\mathcal{E}(L_e) = \mathcal{E}(L) - L_t$ .*

With Lemma 2, we can prove Theorem 1 by contradiction as follows: Recall that jobs in  $\mathcal{E}(L)$  are scheduled in one continuous period of length  $\ell$ . Since  $\|\mathcal{O}(L)\| > \|\mathcal{E}(L)\|$ , any offline algorithm using a speed-1 processor takes more than  $2\ell$  time to complete  $\mathcal{O}(L)$ , and some jobs in  $\mathcal{O}(L)$  have deadlines after  $2\ell$ . In other words,  $L$  contains some late-dead jobs. By Lemma 2,  $\|\mathcal{E}(L_e)\| = \|\mathcal{E}(L)\| - \|L_t\| < \|\mathcal{O}(L)\| - \|L_t\|$ . Note that  $\|\mathcal{O}(L_e)\| \geq \|\mathcal{O}(L) - L_t\| \geq \|\mathcal{O}(L)\| - \|L_t\|$ . In summary,  $\|\mathcal{E}(L_e)\| < \|\mathcal{O}(L_e)\|$ , yet  $L_e$  contains fewer jobs than  $L$ . This contradicts the definition of  $L$  and Theorem 1 follows.

*Proof of Lemma 2.* To see  $L_t \subseteq \mathcal{E}(L)$ , it suffices to observe that it is always feasible to schedule a late-dead job  $J$  within the period  $[\ell, d(J)]$ . This is because  $\ell \leq d(J)/2$  and the time needed to process  $J$  on a speed-2 processor is  $p(J)/2 \leq d(J)/2$ . Thus,  $J$  always passes the feasibility test of EDF-ac.

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

### 3 Lower bound for optimal work scheduling

This section shows that no online algorithm using a speed- $s$  processor, where  $s < (1 + \sqrt{5})/2$ , can match the total work achieved by the optimal offline algorithm using a speed-1 processor.

Let  $\phi$  denote  $(1 + \sqrt{5})/2$ . Note that  $1 + \frac{1}{\phi} = \phi$  and  $1 + \frac{1}{s} > s$  for any  $s < \phi$  (because  $1 + \frac{1}{s} > 1 + \frac{1}{\phi} = \phi > s$ ). The latter implies that a speed- $s$  processor, where  $s < \phi$ , cannot complete  $s + 1$  units of work in  $s$  units of time.

**Theorem 3** *For deadline scheduling on a single processor, no online algorithm is speed- $s$  optimal if  $s < \phi$ .*

To show that an online algorithm  $\mathcal{A}$  is not speed- $s$  optimal, where  $1 \leq s < \phi$ , we consider a sequence of four jobs. Initially,  $\mathcal{A}$  is given two jobs such that  $\mathcal{A}$  cannot complete both by their deadlines. To be optimal,  $\mathcal{A}$  must attempt to complete the job with bigger

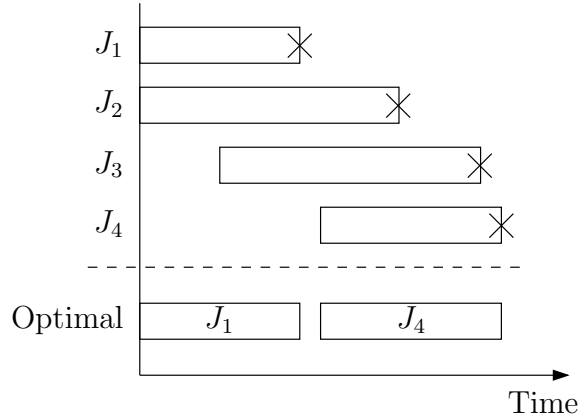


Figure 2: The four jobs.

work. When the job with smaller work becomes infeasible to complete, the third job with an even bigger work is released. Again,  $\mathcal{A}$  cannot complete both current jobs and is forced to switch to the third one without completing the first two jobs. Finally, the fourth job is released late enough so that the optimal offline algorithm can complete both the first and the fourth job, yet  $\mathcal{A}$  can only finish either the third or the fourth job, failing to be optimal. Figure 2 depicts the relationship of these jobs. Details are as follows.

**Definition.** A job  $J$  is said to be *feasible* with respect to a speed- $s$  processor at time  $t$  if the remaining work of  $J$  at time  $t$  is at most  $s(d(J) - t)$ ; otherwise,  $J$  is said to be *infeasible*.

Denote  $(J_1, J_2, J_3, J_4)$  as a sequence of four jobs. All these jobs have a tight deadline (i.e.,  $d(J_i) = r(J_i) + p(J_i)$ ). For  $J_1$  and  $J_2$ , they are both released at time 0, and their work requirements are 1 and  $s$  respectively. The release times of  $J_3$  and  $J_4$  depend on how  $\mathcal{A}$  schedules  $J_1$  and  $J_2$ .

The time required by a speed- $s$  processor to complete both  $J_1$  and  $J_2$  is  $\frac{1+s}{s} > s$ . Since  $d(J_1) = 1$  and  $d(J_2) = s$ ,  $\mathcal{A}$  cannot meet the deadlines of both  $J_1$  and  $J_2$ .  $J_3$  will be released after  $J_1$  becomes infeasible. If  $J_1$  is feasible during the entire period  $[0, d(J_1)] = [0, 1]$ ,  $\mathcal{A}$  must have met the deadline of  $J_1$  by  $d(J_1)$  and cannot complete  $J_2$  on time. In this case, we do not release any more jobs and  $\mathcal{A}$  is already not optimal. It remains to consider the case where  $J_1$  becomes infeasible before  $d(J_1)$ . Let  $t$  be the last instant such that  $J_1$  is still feasible, and let  $\ell$  be the total processing time  $J_1$  received up to time  $t$ . By the definition of  $t$ ,  $s(1 - t) + s\ell = p(J_1) = 1$ , or equivalently,  $t = 1 + \ell - 1/s$ .

We let  $r(J_3) = 1 + \ell - 1/\phi > t$ . That is,  $J_3$  is released after  $J_1$  becomes infeasible. Furthermore,  $J_3$  has a higher work requirement than  $J_2$ ; we let  $p(J_3) = \phi$  and  $d(J_3) = r(J_3) + p(J_3)$ .

**Claim 4** *At time  $r(J_3)$ , the remaining work of  $J_2$  is at least  $s/\phi$ .*

*Proof.* At time  $r(J_3)$ , the work done on  $J_2$  is at most  $s(r(J_3) - \ell) = s(1 + \ell - 1/\phi - \ell) = s - s/\phi$ . The remaining work is at least  $p(J_2) - (s - s/\phi) = s - (s - s/\phi) = s/\phi$ .  $\square$

Next, we show that  $\mathcal{A}$  cannot meet the deadlines of both  $J_2$  and  $J_3$ . At time  $r(J_3)$ , the total remaining work of  $J_2$  and  $J_3$  is at least  $s/\phi + \phi$ . The time required by a speed- $s$  processor to complete both jobs is at least  $(s/\phi + \phi)/s > 1/\phi + 1 = \phi$ . Note that  $\max(d(J_2), d(J_3)) = r(J_3) + \phi$ . Thus,  $\mathcal{A}$  cannot complete both  $J_2$  and  $J_3$  by their deadlines.

Note that  $p(J_3) > p(J_2)$ . Using the same argument for  $J_1$ , we only need to consider the case where  $J_2$  is abandoned by  $\mathcal{A}$  and becomes infeasible before  $d(J_2)$ .  $J_4$  is released after  $J_2$  becomes infeasible. We require that  $J_1$  and  $J_4$  can both be completed by an offline algorithm, yet  $\mathcal{A}$  will complete only one of  $J_3$  and  $J_4$ . Furthermore, we make  $p(J_1) + p(J_4) > p(J_3)$ . Then it follows that  $\mathcal{A}$  is not optimal.

Let  $t'$  be the last instant that  $J_2$  is still feasible. Let  $\ell'$  be the processing time received by  $J_2$  during  $[r(J_3), t']$ . By the definition of  $t'$ , we have  $s(d(J_2) - t') + s\ell'$  equal to the remaining work of  $J_2$  at  $r(J_3)$ , which is at least  $s/\phi$ . In other words,  $t' \leq s + \ell' - 1/\phi$ .  $J_4$  is released after  $J_2$  becomes infeasible. Let  $r(J_4) = \phi + \ell' - 1/\phi$ , which is strictly beyond  $t'$ . Note that  $r(J_4) \geq \phi - 1/\phi = 1$ . Thus,  $r(J_4) \geq d(J_1) = 1$ , and it is possible to complete both  $J_1$  and  $J_4$  using a speed-1 processor.

**Claim 5** *At time  $r(J_4)$ , the remaining work of  $J_3$  is at least  $\phi + s(1 + \ell - \phi)$ .*

*Proof.* Recall that  $r(J_4) > t'$ . During  $[r(J_3), r(J_4)]$ , the work done on  $J_2$  is at least  $s\ell'$ , and the work done on  $J_3$  is at most  $s(r(J_4) - r(J_3)) - s\ell' = s[(\phi + \ell' - 1/\phi) - (1 + \ell - 1/\phi)] - s\ell' = s(\phi - 1 - \ell)$ . The remaining work of  $J_3$  at time  $r(J_4)$  is at least  $p(J_3) - s(\phi - 1 - \ell) = \phi + s(1 + \ell - \phi)$ .  $\square$

To complete the proof of Theorem 3, it remains to show that we can choose the value of  $p(J_4)$  such that an offline algorithm can complete more work than  $\mathcal{A}$ .

We set  $p(J_4) = 1 + \ell$ . Then  $d(J_4) = r(J_4) + p(J_4) = (\phi + \ell' - 1/\phi) + (1 + \ell)$ . Note that  $d(J_3) = r(J_3) + p(J_3) = (1 + \ell - 1/\phi) + \phi$ . Thus,  $d(J_4) = d(J_3) + \ell'$ . Starting from  $r(J_4)$ , the total remaining work of  $J_3$  and  $J_4$  is at least  $\phi + s(1 + \ell - \phi) + (1 + \ell)$  and the time needed by a speed- $s$  processor to complete them is at least  $[\phi + s(1 + \ell - \phi) + (1 + \ell)]/s \geq 1 + \ell - \phi + (\phi + 1)/s = 1 + \ell - \phi + \phi^2/s > 1 + \ell = p(J_4)$ . That is,  $\mathcal{A}$  cannot complete both  $J_3$  and  $J_4$  at the time  $r(J_4) + p(J_4)$ , which is equal to  $\max(d(J_3), d(J_4))$ . The maximum possible work achieved by  $\mathcal{A}$  is  $\max(p(J_3), p(J_4)) = \max(\phi, 1 + \ell)$ .

On the other hand, an offline algorithm using a speed-1 processor can meet the deadlines of  $J_1$  and  $J_4$ , attaining a total work of  $1 + 1 + \ell > \max(\phi, 1 + \ell)$ . Therefore,  $\mathcal{A}$  is not optimal. Theorem 3 follows.

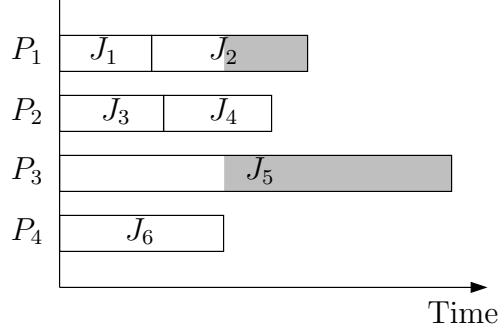


Figure 3: An example of EDF-ac schedule with  $m = 4$  processors. Assume that  $\mathcal{O}$  contains  $J_4$  but not  $J_2$  and  $J_5$ . The shaded regions count toward  $\delta(L, \mathcal{O})$ .

## 4 Optimality of edf-ac for multiprocessor scheduling

We extend the result of Section 2 to the setting with multiprocessors and non-uniform value densities. Specifically, we show that for scheduling on  $m \geq 1$  processors, EDF-ac is speed- $(2 + k)$  optimal, where  $k$  is the importance ratio.

Consider any sequence  $L$  of jobs with value densities in the range  $[1, k]$ . Denote by  $\|L\|$  the total value of jobs in  $L$ , and by  $\mathcal{E}(L)$  the subset of jobs in  $L$  that EDF-ac using  $m$  speed- $(2 + k)$  processors can complete by their deadlines. Note that jobs in  $L - \mathcal{E}(L)$  are never scheduled by EDF-ac. With respect to the EDF-ac schedule of  $L$ , a period is said to be *busy* if all the  $m$  processors are working throughout this period.

Unlike the uni-processor setting, the case when EDF-ac schedules  $L$  over two or more busy periods is no longer straightforward. The problem is that EDF-ac may cause a job to span more than one busy period, making it difficult to derive a contradiction by splitting  $L$ . To deal with multiple busy periods, we strengthen Theorem 1 and prove that the value attained by EDF-ac not only matches that of any offline algorithm but is in excess by a significant amount. Consider any subset  $\mathcal{O}$  of  $L$  that can be completed by the deadlines by some offline algorithm using  $m$  speed-1 processors. For any job  $J$  in  $L$ , let  $\sigma(L, J)$  be the amount of work of  $J$  scheduled by EDF-ac after the last busy period in the EDF-ac schedule of  $L$ . Define  $\delta(L, \mathcal{O})$  to be the sum of  $\sigma(L, J)$  over all jobs  $J$  in  $\mathcal{E}(L) - \mathcal{O}$  (see Figure 3 for an example). We show that  $\|\mathcal{E}(L)\|$  is in excess of  $\|\mathcal{O}\|$  by at least  $\delta(L, \mathcal{O})$ . Then, when there is more than one busy period, we can split  $L$  into two parts both containing those jobs spanning two busy periods, and the overlap is compensated by the excess of  $\delta(L, \mathcal{O})$  after the split. Details are as follows.

**Theorem 6** *Let  $L$  be any job sequence. Then  $\|\mathcal{E}(L)\| - \delta(L, \mathcal{O}) \geq \|\mathcal{O}\|$  for any  $\mathcal{O} \subseteq L$  that can be completed by the deadlines by some offline algorithm using  $m$  speed-1 processors.*

We prove Theorem 6 by contradiction. Suppose there exists a job sequence  $L$  such that, for some  $\mathcal{O} \subseteq L$ ,  $\|\mathcal{E}(L)\| - \delta(L, \mathcal{O}) < \|\mathcal{O}\|$ . Without loss of generality, we consider  $L$

to be the job sequence containing the fewest jobs. The rest of this section is divided into parts, considering the cases when the EDF-ac schedule of  $L$  admits more than one busy period and exactly one busy period, respectively. In each case, we derive a contradiction.

Before showing the details of the contradiction, we first observe that  $L$  can be assumed to satisfy an additional assumption: With respect to the EDF-ac schedule of  $L$ , no job is released outside a busy period. This assumption helps us simplify the contradiction argument.

**Lemma 7** *Suppose that there is a job sequence violating Theorem 6. Then among all such job sequences, there exists one, denoted  $L$ , such that  $L$  contains the smallest number of jobs and no job is released outside a busy period in the EDF-ac schedule of  $L$ .*

*Proof.* Let  $L_0$  be any job sequence with the fewest jobs that violates Theorem 6. Let  $\mathcal{O}_0$  be the subset of  $L_0$  such that  $\|\mathcal{E}(L_0)\| - \delta(L_0, \mathcal{O}_0) < \|\mathcal{O}_0\|$ . With respect to the EDF-ac schedule of  $L_0$ , it is possible that some job  $J$  is released outside a busy period. Yet it is impossible that  $J$ 's deadline is before the next busy period starts. Otherwise, EDF-ac can meet the deadline of  $J$  and deleting  $J$  from  $L_0$  would give a smaller job sequence violating Theorem 6.

Below we show how to construct another sequence  $L$  from  $L_0$  to satisfy the assumption on release time. For every job  $J$  released outside a busy period,  $J$  must be admitted by EDF-ac. Let  $w$  denote the time when the next busy period starts in the EDF-ac schedule of  $L_0$ . We delay the release time of  $J$  to  $w$  and reduce the required work by the amount of work  $J$  has been processed before time  $w$  in the EDF-ac schedule of  $L_0$ . The new value of  $J$  is in proportion to the remaining work. This results in another sequence of jobs  $L$ , which, when scheduled by EDF-ac, have exactly the same busy periods and the same jobs completed. Note that  $\|\mathcal{E}(L)\|$  is less than  $\|\mathcal{E}(L_0)\|$  by exactly the total value reduced in modifying  $L_0$  to  $L$ . With respect to the subset  $\mathcal{O}_0$  of  $L_0$ , we denote the collection of the corresponding jobs in  $L$  as  $\mathcal{O}$ . Then  $\|\mathcal{O}_0\| - \|\mathcal{O}\| \leq \|L_0\| - \|L\|$ . On the other hand,  $\delta(L, \mathcal{O})$  is the same as  $\delta(L_0, \mathcal{O}_0)$ . Recall that  $\|\mathcal{E}(L_0)\| - \delta(L_0, \mathcal{O}_0) < \|\mathcal{O}_0\|$ . Thus,  $\|\mathcal{E}(L)\| - \delta(L, \mathcal{O}) < \|\mathcal{O}\|$ , and  $L$  also violates Theorem 6.  $\square$

## 4.1 More than one busy period

We are now ready to show that if the EDF-ac schedule of  $L$  contains two or more busy periods, we can obtain a contradiction that there is a job sequence with fewer jobs than  $L$ , which violates Theorem 6. Recall that  $\mathcal{O}$  denotes a subset of  $L$  such that  $\|\mathcal{E}(L)\| - \delta(L, \mathcal{O}) < \|\mathcal{O}\|$ .

We construct two smaller job sets  $L_a$  and  $L_b$  from  $L$  as follows. Let  $t$  be the time when the second busy period starts.  $L_a$  contains all jobs in  $L$  released earlier than  $t$ .  $L_b$  contains all other jobs plus some new jobs derived from  $L_a$  as follows. With respect to the EDF-ac schedule of  $L$ , we denote by  $N$  the set of jobs in  $\mathcal{E}(L) \cap L_a$  which have not

yet completed at time  $t$ , and let  $\epsilon > 0$  be a small enough constant such that during the interval  $[t - \epsilon, t]$ , every job in  $N$  is being processed. For each job  $J$  in  $N$ , we add a new job to  $L_b$  with release time  $t - \epsilon$ , deadline  $d(J)$ , required work  $x$ , and value  $x$ , where  $x$  denotes the amount of remaining work of  $J$  at time  $t - \epsilon$ . Denote by  $N'$  the set of new jobs added to  $L_b$ . All jobs in  $L_b$  except those in  $N'$  are released at or after  $t$ , and  $N'$  contains less than  $m$  jobs. When scheduling  $L_b$  using EDF-ac, all the jobs in  $N'$  will get admitted first. It is easy to see that  $\mathcal{E}(L_a) = \mathcal{E}(L) \cap L_a$ .  $\mathcal{E}(L_b)$  contains all jobs in  $N'$  as well as all jobs in  $\mathcal{E}(L) - L_a$ . Thus,  $\|\mathcal{E}(L_a)\| + \|\mathcal{E}(L_b)\| = \|\mathcal{E}(L)\| + \|N'\| < \|\mathcal{O}\| + \delta(L, \mathcal{O}) + \|N'\|$ .

It remains to show that  $L_a$  or  $L_b$  violates Theorem 6. Consider the following two sets:  $\mathcal{O}_a = L_a \cap \mathcal{O}$ , and  $\mathcal{O}_b$  contains all jobs in  $L_b$  that are either found in  $\mathcal{O}$  or derived from jobs in  $\mathcal{O} \cap N$ . Note that  $\|\mathcal{O}_a\| + \|\mathcal{O}_b\| = \|\mathcal{O}\| + \|N' \cap \mathcal{O}_b\|$ . Furthermore,  $\delta(L, \mathcal{O})$  is exactly  $\delta(L_b, \mathcal{O}_b)$ , and  $\delta(L_a, \mathcal{O}_a)$  is at least the total work in  $N' - (N' \cap \mathcal{O}_b)$ , which also equals  $\|N' - (N' \cap \mathcal{O}_b)\|$ . Consider the value of  $\|\mathcal{E}(L_a)\| + \|\mathcal{E}(L_b)\| - \delta(L_a, \mathcal{O}_a) - \delta(L_b, \mathcal{O}_b)$ . It is smaller than  $\|\mathcal{O}\| + \|N'\| - \delta(L_a, \mathcal{O}_a)$ , which is at most  $\|\mathcal{O}\| + \|N' \cap \mathcal{O}_b\|$ , or equivalently,  $\|\mathcal{O}_a\| + \|\mathcal{O}_b\|$ . We conclude that  $\|\mathcal{E}(L_a)\| + \|\mathcal{E}(L_b)\| - \delta(L_a, \mathcal{O}_a) - \delta(L_b, \mathcal{O}_b) < \|\mathcal{O}_a\| + \|\mathcal{O}_b\|$ . Thus, either  $\mathcal{O}_a$  causes  $L_a$  to violate Theorem 6, or  $\mathcal{O}_b$  causes  $L_b$  to violate Theorem 6. Both  $L_a$  and  $L_b$  contain less jobs than  $L$ . This contradicts the definition of  $L$ .

## 4.2 One busy period

To complete the proof of Theorem 6, it remains to derive a contradiction when the EDF-ac schedule of  $L$  contains only one busy period, say,  $[0, \ell]$ . Again, we assume that all jobs in  $L$  are released within this busy period. We say that a job in  $L$  is *early-dead* if its deadline is at or before  $\frac{2+k}{k}\ell$ , and *late-dead* otherwise. For example, when  $k = 1$ , a late dead job has deadline beyond  $3\ell$ . We will show that the early-dead and late-dead jobs in  $L$  satisfy the same properties as in the single-processor setting (see Lemma 9 below); the proof is more complicated due to the multiprocessor setting.

**Lemma 8**  *$L$  contains at least one late-dead job.*

*Proof.* By the definition of  $L$ , there exists  $\mathcal{O} \subseteq L$  such that  $\|\mathcal{O}\| > \|\mathcal{E}(L)\| - \delta(L, \mathcal{O})$  and  $\mathcal{O}$  can be completed by the deadlines using  $m$  speed-1 processors. Note that  $\|\mathcal{E}(L)\| - \delta(L, \mathcal{O})$  is bounded below by the total work scheduled during the busy period of the EDF-ac schedule of  $L$ , which is at least  $m(k+2)\ell$ . In other words,  $\|\mathcal{O}\| > m(k+2)\ell$ . To achieve a total value of  $\|\mathcal{O}\|$ , any offline algorithm using  $m$  speed-1 processors requires time more than  $[m(k+2)\ell]/mk = \frac{k+2}{k}\ell$  units. Thus,  $\mathcal{O}$  contains a job whose deadline is beyond  $\frac{k+2}{k}\ell$ , i.e., a late-dead job.  $\square$

**Lemma 9** *Let  $L_e$  and  $L_t$  be the sets of the early-dead jobs and late-dead jobs in  $L$ , respectively. Then  $L_t \subseteq \mathcal{E}(L)$  and  $\mathcal{E}(L_e) = \mathcal{E}(L) - L_t$ .*

By Lemma 8,  $L_t$  is nonempty and  $L_e$  contains fewer jobs than  $L$ . Before we prove Lemma 9, we show how this lemma can lead to the contradiction that  $L_e$  violates Theorem 6. Partition  $\mathcal{O}$  into two subsets:  $\mathcal{O}_e$  contains all early-dead jobs and  $\mathcal{O}_t$  contains all late-dead jobs. By definition,  $\mathcal{O}_e \subseteq L_e$  and  $\mathcal{O}_t \subseteq L_t$ . As  $\mathcal{O}_e \subseteq \mathcal{O}$ ,  $\mathcal{O}_e$  can also be completed by the deadlines using  $m$  speed-1 processors. The following lemma shows the key property that  $\|\mathcal{O}_e\| > \|\mathcal{E}(L_e)\| - \delta(L_e, \mathcal{O}_e)$ . That is,  $L_e$  violates Theorem 6.

**Lemma 10**  $\|\mathcal{O}_e\| > \|\mathcal{E}(L_e)\| - \delta(L_e, \mathcal{O}_e)$ .

*Proof.* Recall that  $\|\mathcal{O}\| > \|\mathcal{E}(L)\| - \delta(L, \mathcal{O})$ . By Lemma 9,  $\mathcal{E}(L) = \mathcal{E}(L_e) \cup L_t$  and  $\|\mathcal{E}(L)\| = \|\mathcal{E}(L_e)\| + \|L_t\|$ . Thus,  $\|\mathcal{O}\| > \|\mathcal{E}(L_e)\| + \|L_t\| - \delta(L, \mathcal{O})$ . Furthermore,  $\|\mathcal{O}_e\| = \|\mathcal{O}\| - \|\mathcal{O}_t\| > \|\mathcal{E}(L_e)\| - \delta(L, \mathcal{O}) + \|L_t\| - \|\mathcal{O}_t\|$ .

Next, we derive an upper bound of  $\delta(L, \mathcal{O})$ . Recall that  $\delta(L, \mathcal{O})$  denotes the amount of work that the EDF-ac schedule of  $L$  has performed on jobs in  $\mathcal{E}(L) - \mathcal{O}$  after the last busy period. We consider how each job in  $\mathcal{E}(L) - \mathcal{O}$  contributes to  $\delta(L, \mathcal{O})$ . The late-dead jobs in  $\mathcal{E}(L) - \mathcal{O}$  are captured by the set  $L_t - \mathcal{O}_t$ , which contributes an amount of at most  $\|L_t\| - \|\mathcal{O}_t\|$  to  $\delta(L, \mathcal{O})$ . The set of early-dead jobs in  $\mathcal{E}(L) - \mathcal{O}$  is exactly  $(\mathcal{E}(L) - L_t) - \mathcal{O}_e = \mathcal{E}(L_e) - \mathcal{O}_e$ . Since the EDF schedules of  $\mathcal{E}(L_e)$  and  $\mathcal{E}(L)$  give an identical schedule with respect to the early-dead jobs, the last busy period of the former cannot exceed that of the latter, i.e., it must end no later than  $\ell$ . The amount contributed by the early jobs in  $\mathcal{E}(L) - \mathcal{O}$  to  $\delta(L, \mathcal{O})$  is at most  $\delta(L_e, \mathcal{O}_e)$ .

In conclusion,  $\delta(L, \mathcal{O}) \leq (\|L_t\| - \|\mathcal{O}_t\|) + \delta(L_e, \mathcal{O}_e)$ . And  $\|\mathcal{O}_e\| > \|\mathcal{E}(L_e)\| - \delta(L, \mathcal{O}) + (\|L_t\| - \|\mathcal{O}_t\|) \geq \|\mathcal{E}(L_e)\| - \delta(L_e, \mathcal{O}_e)$ .  $\square$

To complete the contradiction argument, it remains to prove Lemma 9.

**Proof of  $L_t \subseteq \mathcal{E}(L)$ .** Let  $J$  be any late-dead job in  $L$ . I.e.,  $d(J) > \frac{2+k}{k}\ell$ . Let  $X$  be the set of jobs admitted by EDF-ac just before  $J$  is released. Note that  $X \subset \mathcal{E}(L)$  and the EDF schedule of  $X$  contains only one busy period up to time  $h$ , where  $h \leq \ell$ . To see why  $J$  must be admitted, we show that the EDF schedule of  $X \cup \{J\}$  always meets the deadline of all jobs in  $X \cup \{J\}$ .

A basic property of EDF is that at any time, the EDF schedule of  $X \cup \{J\}$  uses at least as many processors as the EDF schedule of  $X$ . Thus, the EDF schedule of  $X \cup \{J\}$  contains a busy period ending at  $\hat{h} \geq h$  and outperforms the EDF schedule of  $X$  by at least  $(k+2)(\hat{h} - h)$  units of work. Note that these two schedules differ by exactly  $p(J)$  with regard to the total amount of work. Thus,  $(k+2)(\hat{h} - h) \leq p(J)$ , or equivalently,  $\hat{h} \leq h + p(J)/(k+2) \leq h + d(J)/(k+2)$ .

By the definition of EDF, adding  $J$  into  $X$  does not cause any jobs with deadline earlier than  $d(J)$  to miss the deadline. It remains to consider those jobs  $J'$  in  $X \cup \{J\}$  with  $d(J') \geq d(J) > \frac{k+2}{k}\ell$ . In the EDF schedule of  $X \cup \{J\}$ , the time  $J'$  is completed is bounded above by  $\hat{h} + \frac{p(J')}{k+2}$ , which is at most  $h + \frac{d(J)}{k+2} + \frac{d(J')}{k+2} \leq \ell + \frac{d(J')}{k+2} + \frac{d(J')}{k+2} \leq d(J')$ . In other words, the deadline of  $J'$  is met. In summary, the EDF schedule of  $X \cup \{J\}$  meets the deadline of every job. Thus,  $J$  should be included in  $\mathcal{E}(L)$ .

**Proof of  $\mathcal{E}(L_e) = \mathcal{E}(L) - L_t$ .** To show this equality, it suffices to consider all early-dead jobs  $J$  in the order of their release times and prove inductively that  $J \in \mathcal{E}(L_e)$  if and only if  $J \in \mathcal{E}(L)$ . Let  $J$  be an early-dead job in  $L$ . Assume that before  $J$  is released, the EDF-ac schedules of  $L$  and  $L_e$  have admitted the same set of early-dead jobs, which is denoted  $X$  below. With respect to the EDF-ac schedule of  $L$ , we further define  $Y$  to be the set of late-dead jobs in  $\mathcal{E}(L)$  before  $J$  is released. As  $X \cup Y$  is a subset of  $\mathcal{E}(L)$ , the EDF schedule of  $X \cup Y$  contains only one single busy period, the length, denoted by  $h$ , is at most  $\ell$ .

If  $J \in \mathcal{E}(L)$ , then the EDF schedule of  $X \cup Y \cup \{J\}$  can meet the deadline of all jobs. Removing all late-dead jobs does not change the schedule of the early-dead jobs. The EDF schedule of  $X \cup \{J\}$  can meet the deadline of all jobs. Thus,  $J \in \mathcal{E}(L_e)$ .

If  $J \in \mathcal{E}(L_e)$ , then the EDF schedule of  $X \cup \{J\}$  can meet the deadline of all jobs. Below we investigate the EDF schedule of  $X \cup Y \cup \{J\}$  and argue that the deadline of every job is met. Since the presence of late-dead jobs does not affect the schedule of early-dead jobs, all early-dead jobs in  $X \cup Y \cup \{J\}$  (i.e.,  $X \cup \{J\}$ ) can be completed by their deadlines. For any late-dead job  $J'$  in  $Y$ , the time  $J'$  is completed is bounded by  $\hat{h} + \frac{p(J')}{k+2}$ , where  $\hat{h}$  is the length of the new busy period of the EDF schedule of  $X \cup Y \cup \{J\}$  and  $\hat{h}$  is at most  $\ell + \frac{p(J)}{k+2} < \frac{kd(J')}{k+2} + \frac{d(J)}{k+2}$ . In other words,  $J'$  is completed before  $d(J')$ . In summary, the EDF schedule of  $X \cup Y \cup \{J\}$  meets the deadline of every job, and  $J$  should be included in  $\mathcal{E}(L)$ .

Remarks: To analyze the performance of EDF-ac for scheduling jobs with varying value densities on  $m = 1$  processor, we can also generalize the proof in Section 2, showing that EDF-ac is indeed speed- $(1+k)$  optimal. The details are omitted as the only thing we need to change is the definition of a late-dead job (which becomes a job with a deadline after  $\frac{1+k}{k}\ell$ ) and the argument remains the same.

## 5 Improving edf-ac with value density groups

As shown in the previous section, EDF-ac is speed- $(2+k)$  optimal, where  $k$  is the importance ratio. When  $k$  is large, EDF-ac demands very fast processors in order to be optimal. In this section, we use EDF-ac as a black box to derive a more speed-efficient algorithm. To ease our discussion, we first define an algorithm called  $\lambda$ -EDF-ac, where  $\lambda$  is any integer greater than 1.  $\lambda$ -EDF-ac uses  $\lambda m$  speed- $(2+k^{1/\lambda})$  processors and it can match the total value obtained by any offline algorithm using  $m$  speed-1 processors (see Theorem 11). In particular, we are interested in  $\lceil \log_2 k \rceil$ -EDF-ac, i.e., when the algorithm is using  $\lceil \log_2 k \rceil m$  speed-4 processors. Using time sharing, we can simulate  $\lceil \log_2 k \rceil$ -EDF-ac using  $m$  speed- $(4\lceil \log_2 k \rceil)$  processors and thus obtain an algorithm that is speed- $(4\lceil \log_2 k \rceil)$  optimal.

The algorithm is defined as follows:

**$\lambda$ -edf-ac:** Divide  $\lambda m$  processors into  $\lambda$  clusters, each with  $m$  processors run-

ning EDF-ac independently. For each  $1 \leq i \leq \lambda$ , the  $i$ -th cluster is responsible for jobs with value density between  $k^{(i-1)/\lambda}$  and  $k^{i/\lambda}$ .

**Theorem 11**  $\lambda$ -EDF-ac, using  $\lambda m$  speed- $(2 + k^{1/\lambda})$  processors, can match the value attained by any offline algorithm using  $m$  speed-1 processors.

*Proof.* Consider an input job sequence  $L$ . Let  $L_i$  be the set of jobs that are allowed to use the  $i$ -th cluster. For a set of jobs  $\mathcal{X} \subseteq L$ , let  $\mathcal{O}(\mathcal{X})$  be the set of jobs that meet their deadlines when the optimal off-line algorithm schedules  $\mathcal{X}$  using  $m$  speed-1 processors. Note that  $\|\mathcal{O}(L_1)\| + \dots + \|\mathcal{O}(L_\lambda)\| \geq \|\mathcal{O}(L)\|$ , since the optimal off-line algorithm can always choose to run jobs in  $\mathcal{O}(L) \cap L_1, \dots, \mathcal{O}(L) \cap L_\lambda$  when scheduling  $L_1, \dots, L_\lambda$ , respectively.

The value obtained by  $\lambda$ -EDF-ac is the sum of values obtained by each cluster, i.e.  $\|\mathcal{E}(L_1)\| + \dots + \|\mathcal{E}(L_\lambda)\|$ . Jobs in the same cluster have value densities which differ by at most a factor of  $k^{1/\lambda}$ . For scheduling  $L_i$ , EDF-ac uses  $m$  speed- $(2 + k^{1/\lambda})$  processors. Applying Theorem 6, we have  $\|\mathcal{E}(L_i)\| \geq \|\mathcal{O}(L_i)\|$ . This results in that the value obtained by  $\lambda$ -EDF-ac is at least  $\|\mathcal{O}(L_1)\| + \dots + \|\mathcal{O}(L_\lambda)\| \geq \|\mathcal{O}(L)\|$ .  $\square$

## References

- [1] S. Baruah. Overload tolerance for single-processor workloads. In *IEEE Symposium on Real time technology and applications*, 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 1991 IEEE Real-Time Systems Symposium*, pages 101–110, San Juan, Puerto Rico, 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, Stockholm, Sweden, July 1998.
- [4] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, The Pitt Building, Trumpington Street, Cambridge, United Kingdom, 1998.
- [5] Mark Brehob, Eric Torng, and Patchrawat Uthaisombut. Applying extra-resource analysis to load balancing. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 560–561, San Francisco, California, January 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] Michael L. Dertouzos. Control robotics: the procedural control of physical processes. In *Proceedings of IFIP Congress*, pages 807–813, 1974.
- [8] Michael L. Dertouzos and Aloysius Ka Lau Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, December 1989.
- [9] Jeff Edmonds. Scheduling in the dark. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 178–188, 1999.
- [10] B. Kalyanasundaram and K. R. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.
- [11] G. Koren, D. Shasha, and S. C. Huang. MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. In *Proceedings of the Fourteenth Real-Time Systems Symposium*, pages 172–181, 1993.
- [12] Gilad Koren and Dennis Shasha.  $D^{over}$ : An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal on Computing*, 24(2):318–339, April 1995.
- [13] 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, Baltimore, Maryland, January 1999.
- [14] Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 140–149, El Paso, Texas, May 1997.
- [15] John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo. *Deadline scheduling for real-time systems: EDF and related algorithms*. Kluwer Academic Publishers, Boston, Mass., 1998.