A Tighter Extra-Resource Analysis of Online Deadline Scheduling*

Tak-Wah Lam†
Department of Computer Science
University of Hong Kong
twlam@cs.hku.hk

Tusen-Wan Johnny Ngan
Department of Computer Science
Rice University, Houston
twngan@cs.rice.edu

Kar-Keung To
Department of Computer Science
University of Hong Kong
kkto@cs.hku.hk

Abstract

This paper is concerned with online algorithms for scheduling jobs with deadlines on a single processor. It has been known for long that unless the system is underloaded, no online scheduling algorithm can be 1-competitive, i.e., matching the performance of the optimal offline algorithm. Nevertheless, recent work has revealed that some online algorithms using a moderately faster processor (or extra processors) can guarantee very competitive performance [11] or even be 1-competitive [15, 18]. This paper takes a further step to investigate online scheduling algorithms with an even higher performance guarantee (i.e., better than 1-competitive algorithms) and in particular, presents an extra-resource analysis of the earliest-deadline-first strategy (EDF) with respect to such a higher performance guarantee.

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

1 Introduction

This paper is concerned with online algorithms for scheduling jobs with deadlines on a processor that allows preemption (see, e.g., [2,7,17,21]). A classical example is the earliest deadline

*A preliminary version of this paper has been accepted by The Australian Theory Symposium on Computing, 2004.

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

first (EDF) algorithm, which has been widely used in real-time systems (see [22] for a survey of EDF). A typical concern of the performance of such online algorithms is on the total work of jobs that can be completed by their deadlines (later we will also address the measure of the values of jobs). An online algorithm is said to be $\frac{1}{c}$-competitive for some fraction $\frac{1}{c} \leq 1$ if, for any job sequence, it guarantees to achieve at least $\frac{1}{c}$ of the total work obtained by any offline algorithm (see [4] for more details). Notice that a 1-competitive algorithm matches the performance of the optimal offline algorithm. It has been known for long that when the system is underloaded, EDF is actually 1-competitive [8]; however, in general, EDF as well as any online algorithm is not 1-competitive [9]; indeed, the best possible algorithm can match at most $\frac{1}{4}$ of the total work obtained by any offline algorithm [17].

In recent years, there have been a lot of studies on how to obtain better performance guarantee of online schedulers using a faster processor or other extra resources; such studies are often referred to as the resource augmentation or extra-resource analysis (e.g., [3, 5–7, 10–12, 15, 18, 20]). Intuitively, allowing the online scheduler to use a processor faster than the offline scheduler provides a way to compensate the online scheduler for the lack of future information. For deadline scheduling, Kalyanasundaram and Pruhs [11] were the first to show that a moderately faster processor can guarantee a $\frac{1}{c}$-competitive algorithm, where c is a constant that can be made arbitrarily close to one. Later Lam et al. [18] further showed that EDF supplemented with admission control (denoted as EDF-AC) is indeed 1-competitive when using a speed-2 processor[1], and more recently Koo et al. [15] showed an even stronger result that 1-competitiveness can also be achieved if the online scheduler is given two unit-speed processors.

Knowing that a faster processor can allow an online scheduler to match the optimal offline algorithm using a speed-1 processor, one would naturally demand a better performance guarantee. That is, it is desirable to have a k-competitive algorithm for some $k > 1$. For example, for scheduling jobs (without deadlines) to minimize the total flow time, the recent work of McCullough and Torng [19] gives such a performance guarantee.[2] Yet, for online deadine scheduling, demanding a a k-competitive algorithm for some $k > 1$ is not possible from a technical viewpoint because there always exist job sequences for which no online algorithm can outperform the optimal offline algorithm; for example, a job sequence may be underloaded (i.e., it can be completed entirely by an offline algorithm) or has only a short overloaded period. The above observation motivates us to investigate a less demanding notion called k-aggressive algorithms for deadline scheduling, which can be perceived as follows. Ideally, we want an online algorithm to complete all jobs whenever possible; when

[1] A speed-x processor can process x units of work in one unit of time.

[2] For the problem of scheduling jobs (without deadlines) to minimize the total flow time on multiprocessors, Phillips et al. [20] were the first to show that the algorithm SRPT, when using speed-s processors where $s \geq 2$, is 1-competitive, i.e., matching the total flow tme incurred by the optimal offline algorithm using speed-1 processors; more recently, McCullough and Torng [19] improved the analysis to show that the flow time incurred by SRPT is indeed at least s times smaller than that of the optimal offline algorithm using speed-1 processors.

it cannot complete all, there must be a certain number of conflicting jobs and we require the algorithm to guarantee that its performance on such jobs is at least k times as good as that of the optimal algorithm. In other words, we avoid demanding a higher performance guarantee for those trivial cases where the system is underloaded.

Definition 1. *An online algorithm A is said to be k-aggressive if A can satisfy the following requirements. Given any job sequence I, A partitions the jobs into two categories, called overloaded jobs and underloaded jobs, such that*

- *with respect to overloaded jobs, the work attained by A is at least k times that of any offline algorithm (using a unit-speed processor); and*
- *for underloaded jobs, A completes all of them by their deadlines.*

Note that the above definition has no restriction on the way an online algorithm classifies the jobs. Different k-aggressive algorithms can perceive the distribution of overloaded and underloaded jobs differently. At first glance, one might think that labeling more jobs as underloaded jobs would give an advantage; yet this cannot be overdone as all underloaded jobs are required to be completed by their deadlines. When the system is reasonably overloaded, a k-aggressive algorithm must classify most jobs as overloaded and outperform the optimal offline algorithm on these jobs.

By definition, for any $k \leq 1$, an algorithm is k-competitive if and only if it is k-aggressive. In other words, aggressiveness can be regarded as an extension of competitiveness. As mentioned before, EDF-AC using a two times faster processor is 1-competitive [18], and thus 1-aggressive. It is however non-trivial whether a higher degree of aggressiveness can be achieved. In this paper we answer in the affirmative that EDF-AC can be k-aggressive for any $k \geq 1$ when using a speed-$(k+1)$ processor.

Our aggressive analysis of EDF-AC is indeed tight as we can show that using a speed-$(k+1)$ processor, EDF-AC is not $(k+\epsilon)$-aggressive for any $\epsilon > 0$. Furthermore, if the speed factor is reduced to be slightly less than $k+1$, then EDF-AC is not k-aggressive. Such lower bound results can be generalized to any online algorithms that decide at release time, i.e., the decision of committing to a job to meet its deadline is made immediately when the job is released.

Our aggressive analysis of EDF-AC can be extended to the case when the objective is to maximize the total value instead of the total work of jobs completed. In this case, we assume that each job is associated with a value which is independent of the work requirement. With extra resource, the best possible online algorithm is is $1/(\sqrt{\lambda}+1)^2$-competitive [17], where λ is the importance ratio, i.e., the ratio of the largest possible value per unit of work to the smallest possible value per unit of work. In this paper, we show that a simple extension of EDF-AC, suggested in [18], is indeed k-aggressive for any $k \geq 1$ when using a speed-$((2k+1)\lceil \log \lambda \rceil)$ processor.

The remainder of this paper is organized as follows: Section 2 presents the aggressive analysis of EDF-AC when the concern is the work, or equivalently, $\lambda = 1$. Section 3 extends

the analysis to the case of general λ (i.e., jobs with non-uniform value densities). Section 4 shows that our analysis is tight.

Before leaving this section, we give a precise definition of the deadline scheduling problem (which is often referred to as the firm deadline scheduling problem in the literature) and EDF-AC.

Problem definition: Consider a single processor system. Jobs are released in an unpredictable fashion, each requesting a certain amount of work (processing time). The work, deadline, and value of a job are revealed only when the job is released. Deadlines are firm in the sense that completing a job after its deadline gives zero value. Notice that a system may be overloaded, in which case there is no way to schedule every job released to meet the deadline. The aim of a scheduler is to maximize the total value (or work) of jobs meeting the deadlines. Preemption is allowed at no cost (i.e., a preempted job can be restarted from the point of preemption at any time). The design of a good scheduler is further complicated by the fact that jobs may have different value densities, i.e., different ratios of value to processing time. The importance ratio λ of a system is defined as the ratio of the largest possible value density to the smallest possible value density. When $\lambda = 1$, all jobs have the same value density. The problem of maximizing the total work is actually a special case of the problem of maximizing the total value where $\lambda = 1$.

To simplify our argument, we assume that jobs have distinct release times and deadlines since ties can be broken consistently, say, by job identification numbers. Furthermore, we assume that the first job is released at time 0.

EDF and EDF-AC: In this paper, EDF 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. In the following EDF-AC denotes EDF enhanced with the following simple form of admission control. Upon release, a job must pass a test to get admitted for EDF scheduling. The test simply checks whether the new job together with the previously admitted jobs can all be completed by their deadlines using EDF. Once a job is admitted, EDF-AC guarantees to complete it.

2 The Aggressiveness of EDF-AC

In this section we show that when jobs have uniform value density (or equivalently, when we want to maximize the total work), EDF-AC, when using a speed-$(k+1)$ processor, is k-aggressive, where k is any number at least one.

Theorem 1. *EDF-AC, when using a speed-$(k+1)$ processor, is k-aggressive for any $k \geq 1$.*

We prove Theorem 1 by contradiction. Assume that EDF-AC, using a speed-$(k+1)$ processor, is not k-aggressive for some job sequence. That is, there exists some job sequence that does not admit a categorization into overloaded and underloaded jobs such that EDF-AC

completes all underloaded jobs, and for overloaded jobs, EDF-AC achieves a total work at least k times that of the optimal offline algorithm. Let I be such a job sequence containing the fewest jobs. We will first show that I must satisfy certain properties (see Lemmas 2 and 3). Then we prove that EDF-AC is k-aggressive for any job sequence which possesses those properties. In other words, EDF-AC is k-aggressive for I, and we obtain a contradiction to complete the proof of Theorem 1.

Definition 2. *At any time t, we say that EDF-AC is idle if all admitted jobs have already been completed; otherwise, it is said to be busy.*

Lemma 2. *When scheduling the job set I, EDF-AC is busy over exactly one continuous period.*

Proof. Assume that EDF-AC is busy over two or more disjoint periods. Let t_ℓ be the start time of the last busy period. Divide I into two parts I_1 and I_2, one for the jobs released before t_ℓ and one for the rest. If we schedule I_1 and I_2 by EDF-AC separately, the schedules produced are disjoint and their union is identical to the schedule of I. Since I is the smallest counter-example showing that EDF-AC is not k-aggressive, jobs in I_1 (I_2, respectively) can be categorized into overloaded jobs called O_1 and underloaded jobs U_1 (O_2 and U_2, respectively) such that the k-aggressive property holds (see the definition in the introduction). Consider the following categorization of I. Jobs in $O = O_1 \cup O_2$ are considered overloaded, and jobs in $U = U_1 \cup U_2$ underloaded. EDF-AC, when scheduling I, completes all jobs in U in time, while the amount of work scheduled by EDF-AC for overloaded jobs would still exceed that of any offline algorithm by a factor of k. In other words, I should be k-aggressive, contradicting our assumption on I. □

The second property of I is related to a concept called repudiation, defined as follows.

Definition 3. *Consider the moment when EDF-AC fails to admit a newly released job J. That is, when J is released, it is found that using EDF to schedule J together with all jobs admitted previously would cause some job to miss the deadline. Any such job is said to* repudiate *J. Note that it is possible that a job repudiates itself.*

Lemma 3. *Let J_ℓ be the job in I with the latest deadline. In the course of scheduling I there is at least one time when J_ℓ repudiates a job.*

Proof. Suppose to the contrary that J_ℓ never repudiates any job. Note that a job can only repudiate another job if the latter has an earlier deadline. Since J_ℓ does not repudiate itself, no job can repudiate it. Hence J_ℓ is admitted by EDF-AC at its release time. Consider any moment after J_ℓ is admitted. Any newly released job, if rejected by EDF-AC, must be repudiated by a job other than J_ℓ. Thus if we remove J_ℓ from I, EDF-AC will not admit more jobs. Since J_ℓ is the latest deadline job, removing it does not affect the EDF-AC schedule of any other job in I. Notice that $I - \{J_\ell\}$ contains one less job than I, so EDF-AC is k-aggressive for $I - \{J_\ell\}$. In other words, jobs in $I - \{J_\ell\}$ can be categorized into overloaded jobs

O and underloaded jobs U in such a way that the k-aggressive property holds. Consider the following categorization of I: jobs in O are overloaded, and jobs in $U \cup \{J_\ell\}$ are underloaded. When EDF-AC schedules I, the underloaded jobs are all scheduled, while the amount of work scheduled by EDF-AC for overloaded jobs would still exceed that of any offline algorithm by a factor of k. In other words, I should be k-aggressive, contradicting the definition of I. □

With the above two properties of I, we are ready to prove that EDF-AC is k-aggressive for I.

Proof of Theorem 1. By Lemmas 2 and 3, we can assume that I contains only one busy period, and J_ℓ, the job with the latest deadline in I, repudiates another job J at some time t. Let $p(J)$ and $d(J)$ denote the processing time and deadline of job J, respectively. Below we show that I can satisfy the requirement for k-aggressiveness, in particular, by assigning all jobs of I to be overloaded. Since all jobs have deadline no later than $d(J_\ell)$, the offline algorithm can only complete work amounting to $d(J_\ell)$. We want to show that EDF-AC can complete at least $k\ d(J_\ell)$ units of work.

Recall that J is repudiated by J_ℓ at time t. By definition, at time t, if we attempt to use EDF to schedule J together with all the previous jobs admitted by EDF-AC, J_ℓ can complete only after its deadline (i.e., $d(J_\ell)$). Note that a speed-$(k+1)$ processor takes $p(J)/(k+1)$ units of time to process J. That means, at time t, all the previous jobs admitted before must be able to keep the processor busy up to $d(J_\ell) - p(J)/(k+1)$. Thus, EDF-AC is busy during the whole period from 0 to $d(J_\ell) - p(J)/(k+1)$. The length of this interval is at least $d(J_\ell) - d(J)/(k+1) > d(J_\ell) - d(J_\ell)/(k+1) = k\ d(J_\ell)/(k+1)$. Using a speed-$(k+1)$ processor, EDF-AC can complete at least $k\ d(J_\ell)$ units of work, thus satisfying the k-aggressive requirement. This contradicts the fact that EDF-AC is not k-aggressive for I. □

Notice the proof above implicitly shows how to categorize jobs into overloaded and underloaded jobs so that EDF-AC satisfies the requirement of k-aggressiveness. Details are as follows. Consider the schedule produced by EDF-AC. It may contain multiple busy periods, in which case we will consider each busy period independently, each time categorizing those jobs released during one busy period as follows. We will find the latest deadline job J among the jobs released during one busy period. If the job J repudiates another job during its scheduling, all jobs in the period are categorized as overloaded, and the categorization completes. Otherwise, J is categorized as underloaded, and is removed from consideration. Once we remove J, the scheduling of the remaining jobs may again contain multiple busy periods. They are categorized recursively using the same strategy. The proof above shows that the k-aggressive property holds for this categorization.

3 General value density

It is more difficult to guarantee aggressiveness when jobs have vastly different value densities. Recall that λ denotes the importance ratio of the jobs. The result of Section 2 can be easily extended to show that EDF-AC is k-aggressive when using a speed-$(k\lambda + 1)$ processor.

Lemma 4. *With respect to jobs with an importance ratio λ, EDF-AC, when using a speed-$(k\lambda + 1)$ processor, is k-aggressive for any $k \geq 1$.*

Proof. (Sketch) Again, we suppose on the contrary that I is a job sequence containing the fewest jobs that violating Lemma 4. Let J_ℓ be the job with the latest deadline in I. Note that Lemmas 2 and 3 are not related to the job values and remain valid. Thus, using the argument in the proof of Theorem 1, we can show that EDF-AC using a speed-$(k\lambda + 1)$-processor can complete at least $k\lambda d(J_\ell)$ units of work, attaining a total value of at least $k\lambda d(J_\ell)$. On the other hand, an offline algorithm can only complete jobs with a total value at most $\lambda d(J_\ell)$. This contradicts that EDF-AC is not k-aggressive for I, and Lemma 4 follows. □

Notice that the aggressiveness guarantee provided by Lemma 4 deteriorates quickly as the importance ratio increases. Nonetheless, we can use EDF-AC as a building block to devise an improved algorithm, called λ-EDF-AC, that requires only a speed-$((2k+1)\lceil \log \lambda \rceil)$ processor.

We assume the value densities to be between 1 and λ. The λ-EDF-AC algorithm splits its processor into $\lceil \log \lambda \rceil$ virtual processors, each being speed-$(2k + 1)$. Each of them runs EDF-AC for a disjoint subset of jobs. In particular, the i-th virtual processor schedules jobs with value density between $[2^{i-1}, 2^i)$, and jobs of value density λ is also scheduled by the $\lceil \log \lambda \rceil$-th processor even if λ is a power of 2.

Theorem 5. *With respect to jobs with an importance ratio λ, λ-EDF-AC is k-aggressive when each virtual processor is speed-$(2k+1)$, i.e., when λ-EDF-AC is using a speed-$((2k+1)\lceil \log \lambda \rceil)$ processor.*

Proof. Notice that each virtual processor is running a k-aggressive algorithm. In other words, the jobs S_i given to the i-th processor can be categorized as overloaded jobs O_i and underloaded jobs U_i, such that the i-th virtual processor completes all jobs in U_i, and attains at least k times the work attained by any offline algorithm for O_i. Define $U = \bigcup U_i$ and $O = \bigcup O_i$. Such a categorization of jobs can demonstrate that λ-EDF-AC is k-aggressive: All jobs in U are completed by λ-EDF-AC; for jobs in O, λ-EDF-AC attains a value at least k times the sum of values that an offline algorithm can attain for each O_i, where the sum is at least the maximum value obtained by an offline algorithm for the job set O. □

4 Lower Bound for Aggressiveness

An algorithm is said to *decide at release time* if the decision of committing to a job to meet its deadline is made immediately when the job is released. Note that EDF-AC is an algorithm

that decides at release time. For algorithms of this type, we can show that the above speed and aggressiveness are tight.

Theorem 6. *Let $\mathcal{A}$ be an online algorithm using a speed-$(k+1)$ processor for some positive integer k. If $\mathcal{A}$ decides at release time, then $\mathcal{A}$ is not $(k+\varepsilon)$-aggressive for any $\varepsilon > 0$.*

Proof. Consider the following sequence S of $k+2$ jobs. The first job J_0 has release time 0, processing time $\varepsilon/2$, and deadline $\varepsilon/2$. All the remaining $k+1$ jobs have release time, processing time, and deadline be δ, 1, and $1+\delta$, respectively, where $\delta < \varepsilon/(2k+2)$.

$\mathcal{A}$ must commit to the first job positively, as it is the only available job at that time. Then it can commit to at most k late jobs positively, and has to reject at least one late job J. (This is because $(k+1+\varepsilon/2)/(k+1) = 1+\varepsilon/(2k+2) > 1+\delta$.) Since an aggressive algorithm must complete all underloaded jobs, J must be categorized as overloaded. An optimal offline algorithm can choose to work only on J. Being a $(k+\varepsilon)$-aggressive algorithm, $\mathcal{A}$ must thus complete a set of overloaded jobs with total work being at least $k+\varepsilon$. But as we have seen, $\mathcal{A}$ must reject at least one job with processing time 1, and thus completes only work amounting to $k+\varepsilon/2 < k+\varepsilon$. □

Theorem 7. *Let $\mathcal{A}$ be an online algorithm using a speed-$(k+1-\varepsilon)$ processor for some positive integer k and some $\varepsilon > 0$. If $\mathcal{A}$ decides at release time, then $\mathcal{A}$ is not k-aggressive.*

Proof. Consider the following sequence S of $k+2$ jobs. The first $k+1$ jobs have release time 0, processing time $1-\varepsilon/3k$, and deadline 1. The last job has release time $\varepsilon/4k$, processing time $1-\varepsilon/4k$ and deadline 1. $\mathcal{A}$ cannot commit to all the first $k+1$ jobs, since it can only complete $k+1-\varepsilon$ work in 1 unit of time, while the total work of the $k+1$ jobs is $k+1-(k+1)\varepsilon/3k$ which is larger. Thus $\mathcal{A}$ rejects at least one of these $k+1$ jobs, which must thus be considered as overloaded job. Since an offline algorithm may work on that job, to be k-aggressive $\mathcal{A}$ must commit to all the remaining k jobs. As a result, the last job (of slightly larger work) must be rejected, since to complete it means the algorithm completes work amounting to $k-\varepsilon/2+1-\varepsilon/4k$, which is more than the maximum we have mentioned earlier. So the last job must be categorized as overloaded. The offline algorithm may opt to complete the job, so to be k-aggressive the algorithm must complete work amounting to $k-\varepsilon/4$, which is not the case ($\mathcal{A}$ only completes $k-\varepsilon/3$ work). □

5 Concluding remarks and future work

This paper introduces the notion of k-aggressive algorithms for deadline scheduling, aiming at studying online algorithms with a performance guarantee better than 1-competitiveness. While we show that EDF-AC using a speed-$(k+1)$ processor is k-aggressive, such a virtue is definitely not limited to EDF-AC. It can be shown that the EDF-Plus algorithm, introduced in [15], also has this property.

There are several open problems related to aggressiveness. It is non-trivial to us whether there exists an online algorithm that is k-aggressive when given extra unit-speed processors. The special case when $k = 1$ has been resolved in [15]. Furthermore, this paper focuses on single-processor scheduling. it is interesting to extend the study of aggressiveness to the multiprocessor setting [16]. We believe that similar results can be obtained.

Aggressive analysis can also be applied to other deadline scheduling problems such as non-preemptive scheduling, scheduling with partial credits [7], and on-demand broadcasting [13, 14]. The idea is again to study how extra resources can provide a performance guarantee better than 1-competitiveness.

References

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

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

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

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

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

[6] H.L. Chan, T.W. Lam, and K.K. To. Non-migratory Online Deadline Scheduling on Multiprocessors. In *Proc. SODA*, pages 970–979, 2004.

[7] M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichý, and N. Vakhania. Preemptive scheduling in overloaded systems. In *Proc. ICALP*, pages 800–811, 2002.

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

[9] M. L. Dertouzos and A. K. L. Mok. Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks *IEEE Transactions on Software Engineering*, 15(12): 1497–1506, 1989.

[10] J. Edmonds. Scheduling in the dark. In *Proc. STOC*, pages 179–188, 1999.

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

[12] B. Kalyanasundaram and K. R. Pruhs. Maximizing job completions online. In *Proc. ESA*, pages 235–246, 1998.

[13] B. Kalyanasundaram and K. R. Pruhs. On-demand Broadcasting under Deadline. In *Proc. ESA*, pages 313–324, 2003.

[14] J.H. Kim and K.Y. Chwa. Scheduling Broadcasts with Deadlines. In *Proc. COCOON*, pages 415-424, 2003.

[15] C. Y. Koo, T. W. Lam, T. W. Ngan, and K. K. To. Extra Processors versus Future Information in Optimal Deadline Scheduling. In *Proc. SPAA*, pages 133–142, 2002.

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

[17] 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.

[18] T. W. Lam and K. K. To. Performance Guarantee for Online Deadline Scheduling in the Presence of Overload. In *Proc. SODA*, pages 755–764, 2001.

[19] J. McCullough and E. Torng. SRPT Optimally Utilizes Faster Machines to Minimize Flow Time. In *Proc. SODA*, pages 350–358, 2004

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

[21] 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.

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