Title	Competitive deadline scheduling via additional or faster processors
Author(s)	Koo, CY; Lam, TW; Ngan, TW; To, KK
Citation	Journal Of Scheduling, 2003, v. 6 n. 2, p. 213-223
Issued Date	2003
URL	http://hdl.handle.net/10722/48428
Rights	Creative Commons: Attribution 3.0 Hong Kong License

Competitive Deadline Scheduling via Additional or Faster Processors

Chiu-Yuen Koo Tak-Wah Lam Tsuen-Wan Ngan Kar-Keung To
Department of Computer Science, University of Hong Kong, Hong Kong
Email: {cykoo, twlam, twngan, kkto}@cs.hku.hk

November 11, 2000

Abstract. We consider the online problem of scheduling jobs with deadlines in a single-processor system that allows preemption. The aim is to maximize the total value of jobs completed by their deadlines. It is known that the competitive ratio for this problem is $\Theta(k)$, where k is the ratio of the maximum possible value density to the smallest possible one. Yet, if the online scheduler is given a processor faster (say, two times faster) than the adversary, there exists an algorithm called SLACKER that can achieve an $O(1)$ competitive ratio. In this paper, we show that using only additional unit speed processors is a possible but not a cost effective way to achieve constant competitiveness. Specifically, we find that $\Theta(\log k)$ unit speed processors are required. On the other hand, we give a better analysis of the competitiveness of SLACKER; this new analysis enables us to show that SLACKER when extended to the multi-processor systems can still guarantee constant competitiveness.

1 Introduction

We study online algorithms for the following firm-deadline scheduling problem in a single-processor system. Jobs are released in an unpredictable fashion and the processing time and deadline of a job are known only when the job is released. Preemption is allowed at no cost. In general, a system may be *overloaded* in the sense that there is no schedule meeting the deadline of every job released. A scheduler aims to maximize the total *value* of jobs that can be completed by their deadlines, where the *value* of a job is another

[†]Some results in this paper appears as part of *Performance Guarantee for Online Deadline Scheduling in the Presence of Overload*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, 2001.

parameter given upon the release of the job, which reflects the importance of the job. To ease our discussion, let us define the *value density* of a job to be its value divided by the processing time. The *importance ratio* of a system is the ratio of the largest possible value density to the smallest possible one. See [18] for more discussion on deadline scheduling.

We analyze the performance of online algorithms with respect to their competitiveness (see, e.g., [5] and [17] for general background). In this paper, we say that an online algorithm $\mathcal{A}$ is 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. For the firm-deadline scheduling problem, the early work of Dertouzos [7] showed that the Earliest Deadline First (EDF) strategy is 1-competitive for *underloaded* systems. That is, whenever there exists a schedule meeting the deadline of every job, EDF can always do so. However, without the underloaded assumption, no algorithm can be 1-competitive; indeed, Baruah et al. [2,3] gave a lower bound of $(1+\sqrt{k})^2$ on the competitive ratio, where k is the importance ratio. That means, even if all jobs have uniform job density (i.e., $k = 1$), the best algorithm we can expect is 4-competitive. Afterward, Koren and Shasha [13] showed that this lower bound is tight by giving a $(1+\sqrt{k})^2$-competitive algorithm. Notice that when k is large, such performance guarantee is not satisfactory.

In recent years, a plausible approach to obtaining better performance guarantee without making assumption on future inputs is to allow the online scheduler to have more resources than the adversary (e.g., [4, 6, 8, 10, 14, 16]). Specifically, we would like to compare the online scheduler using a faster processor or more than one (unit speed) processors against an adversary using a unit speed processor. Intuitively, the additional resources are needed to compensate the online scheduler for the lack of future information. The key question is whether a moderate amount of additional resources can provide satisfactory competitiveness. For the firm-deadline scheduling problem, the pioneer work of Kalyanasundaram and Pruhs [10] showed that the competitive ratio can be improved to a constant independent of the importance ratio k when the online scheduler is given a moderately faster processor; precisely, they gave an algorithm called SLACKER, which, if given a speed-$(1+2\delta)$ processor for any $\delta > 0$, is $(1+\delta^{-1})(1+\delta^{-1/2})(1+\delta^{-1/2}+\delta^{-1})$-competitive. (A speed-$s$ processor means a processor s times faster than a unit speed processor.) For example, with a speed-2 processor, the competitive ratio of SLACKER is 32. Recently, Lam and To [15] showed that 1-competitiveness can be achieved if a processor of $4\lceil \log k \rceil$ times faster is used.

In reality, processor speed is bounded by technology and we may not be able to satisfy the demand of an arbitrarily fast processor. Using additional unit speed processors instead of a faster processor is a more feasible solution. Notice that a schedule using $p > 1$ unit speed processors always implies one using a speed-p processor; the converse is not true, though. For the firm-deadline scheduling problem, it is not known how to make use of additional unit speed processors to improve the competitive ratio from $(1+\sqrt{k})^2$ to $O(1)$. Nevertheless, there are two related results when value densities are not a concern. First, Baruah [1] considered jobs with uniform value density and showed that the competitive

δ	0.01	0.1	0.5	1	2	10
speed	1.02	1.2	2	3	5	21
$(1+\delta^{-1})(1+\delta^{-1/2})(1+\delta^{-1/2}+\delta^{-1})$	123321	648.4	32.0	12	5.65	2.05
$1+2\delta^{-1}+4\delta^{-2}$	40201	421	21	7	3	1.24

Table 1: The first row illustrates the competitive ratio of SLACKER given in [10]; the second row shows our new result.

ratio can be improved from 4 (due to the result in [13]) to 2 using two unit speed processors, and in general to $p/(p-1)$ with p processors. Second, if the concern is to maximize the total number of job completions, Kalyanasundaram and Pruhs [9] gave an algorithm that is O(1)-competitive when given two unit speed processors.

In this paper, we revisit the problem of maximizing the total value with respect to jobs with non-uniform value densities. We observe that the SAFE-RISKY Algorithm given by Koren [11] is indeed k-competitive when given two processors. (This implies an improvement over Baruah's result [1]; specifically, for the uniform value density setting, two processors are sufficient to guarantee 1-competitiveness.) More interestingly, SAFE-RISKY can be easily adapted to become 2-competitive when given $2\lceil \log k \rceil$ unit speed processors. Furthermore, we find that the processor bound is asymptotically tight as we prove that no online algorithm using p processors is O(1)-competitive unless $p = \Omega(\log k)$. Intuitively, increasing the computational power of the online scheduler with additional unit speed processors is not an effective way to attain $O(1)$ competitiveness.

Another contribution of this paper is a new analysis of SLACKER, improving the competitive ratio from $(1+\delta^{-1})(1+\delta^{-1/2})(1+\delta^{-1/2}+\delta^{-1})$ to $1+2\delta^{-1}+4\delta^{-2}$. For example, given a speed-2 processor, our analysis reveals that SLACKER is actually 21-competitive instead of 32-competitive. See Table 1 for a comparison. The importance of this result lies on its application to the multi-processor setting. For firm-deadline scheduling on $m \geq 2$ processors, it is known that the algorithm MOCA [12] is $(1 + m(k^{1/\psi} - 1))$-competitive, where $\psi = m \ln k/2(\ln k + 1)$, but no result has been heard on using faster processors to attain constant competitive ratio. In fact, SLACKER has a natural extension to the multiprocessor setting, but it is non-trivial to generalize the analysis of SLACKER in [10]. Based on our new analytical tool, we show that the multiprocessor version of SLACKER, when given m speed-$(1+2\delta)$ processors for any $\delta > 0$, is still $(1+2\delta^{-1}+4\delta^{-2})$-competitive against an adversary using m unit speed processors.

The remainder of this paper is organized as follows. Section 2 shows a tight bound on the number of unit speed processors to attain constant competitiveness. Section 3 extends the algorithm SLACKER and gives a better analysis on the competitive ratio.

Throughout this paper, we denote the release time, processing time, deadline, value, and value density of a job J as $r(J), p(J), d(J), v(J)$, and $\rho(J)$, respectively. Without loss of generality, we assume that all jobs have value densities in the range $[1, k]$, where k is the

importance ratio. It is also worth-mentioning that an online scheduling algorithm operates by reacting to the release or completion of jobs, and possibly other events scheduled by the algorithm itself.

2 Competitiveness via additional processors

In this section, we investigate the competitiveness of online algorithms that are given additional unit speed processors to solve the single-processor firm-deadline scheduling problem. We first observe that Koren's SAFE-RISKY algorithm [11] is O(1)-competitive when given $O(\log k)$ unit speed processors, where k is the importance ratio. Next, we give a lower bound of $\Omega(\log k)$ on the number of unit speed processors to guarantee O(1)-competitiveness.

Throughout this section, our concern is on processors of unit speed only. Unless otherwise specified, a processor is meant to be a unit speed processor.

2.1 The SAFE-RISKY Algorithm

The SAFE-RISKY algorithm is given by Koren [11] for solving the firm-deadline scheduling problem involving jobs with uniform value density. SAFE-RISKY naturally uses two processors, and Koren [11] showed that SAFE-RISKY is 2-competitive for the two-processor firm-deadline scheduling problem. I.e., SAFE-RISKY, running on two processors, is 2-competitive against an adversary using two processors. In fact, Koren's work is readily to give a more general result as follows.

Lemma 1 [11] *Let c be any positive integer. For jobs with uniform value density,* SAFE-RISKY*, when using two processors, is c-competitive against an adversary using c processors.*

Notice that SAFE-RISKY can also be used to schedule jobs with non-uniform value densities. I.e., SAFE-RISKY simply ignores the varying value densities. In this case, the performance of SAFE-RISKY is as follows.

Corollary 2 *With respect to jobs with non-uniform value densities,* SAFE-RISKY*, using two processors, is k-competitive against an adversary using one processor, where k is the importance ratio.*

Proof. Let X be any sequence of jobs with value densities in the range $[1, k]$, and let X' be identical to X except that the value density of each job is set to one. By Lemma 1, with respect to X', the value obtained by SAFE-RISKY, using two processors, is no less than the value obtained by the optimal offline schedule using one processor. On the other

hand, the value obtained on X by an adversary using one processor is at most k times of the value obtained on X' by the optimal offline schedule using one processor. Thus, the corollary follows. □

Based on Corollary 2, we can prove the main result of this section. Let p be any positive integer. We generalize SAFE-RISKY to an algorithm using $2p$ processors as follows. Divide jobs into p categories. For $1 \leq i \leq p$, the i-th category comprises jobs with value densities between $k^{(i-1)/p}$ and $k^{i/p}$. The $2p$ processors are also divided into p pairs, each pair is running a copy of SAFE-RISKY and is responsible for jobs of a unique category. Notice that the importance ratio of the set of jobs that are handled by each copy of SAFE-RISKY is bounded by $k^{1/p}$. We call this algorithm p-SAFE-RISKY. Below we prove that p-SAFE-RISKY is $k^{1/p}$-competitive against an adversary using one processor.

Theorem 3 *With respect to jobs with non-uniform value densities, the algorithm p-SAFE-RISKY, which uses $2p$ processors, is $k^{1/p}$-competitive against an adversary using one processor, where k is the importance ratio.*

Proof. Consider any job sequence X. Let $X_i \subseteq X$ contain the jobs of the i-th category. By Corollary 2, with respect to each X_i, the value obtained by p-SAFE-RISKY is at least $1/k^{1/p}$ times of the value obtained by an optimal offline schedule for X_i using one processor. With respect to X, any offline schedule for X using one processor gives a value no more than the sum of the values obtained by an optimal single-processor offline schedule for each individual X_i. Thus, the total value obtained by p-SAFE-RISKY on X is at least $1/k^{1/p}$ times of the value obtained by an adversary. The lemma follows. □

Corollary 4 *For any constant $c > 1$, let $p = \lceil \log_c k \rceil$. The algorithm p-SAFE-RISKY, which uses $2\lceil \log_c k \rceil$ processors, is c-competitive against an adversary using one processor.*

2.2 Lower bound

In this section, we derive a lower bound on the number of unit speed processors required to achieve constant competitiveness for the firm-deadline scheduling problem. Recall that k denotes the importance ratio. Precisely, we show that for any firm-deadline scheduling algorithm $\mathcal{A}$, if $\mathcal{A}$, using p processors, is O(1)-competitive against an adversary using one processor, then $p = \Omega(\log k)$. Below, unless otherwise specified, when we say that an algorithm is competitive, it is meant to be competitive against an adversary using one processor.

The lower bound stems from the following theorem.

Theorem 5 *Any firm-deadline scheduling algorithm using $p < \log k$ processors has a competitive ratio at least $\frac{1}{2}k^{1/p}$.*

Let us first look at the consequence of the above theorem. Suppose there is an online algorithm that uses p processors and is c-competitive for some constant c. If $p < \log k$, then by Theorem 5, $c \geq \frac{1}{2}k^{\frac{1}{p}}$, or equivalently, $p \geq \log_{2c} k$. Therefore, $p = \Omega(\log k)$.

The rest of this section is devoted to proving Theorem 5. Let $\mathcal{A}$ be an scheduling algorithm using $p < \log k$ processors. To avoid triviality, we assume that $k \geq 2$ and $p \geq 1$. Below we construct a set of inputs that make $\mathcal{A}$ perform poorly. For any real numbers $d, \varepsilon > 0$, define $T(d, \varepsilon)$ to be a set of $p + 1$ jobs, each belongs to a distinct category defined below.

job category	value density	processing time	value
0	1	1	1
1	$2d$	ε	$2d\varepsilon$
2	$(2d)^2$	ε^2	$(2d\varepsilon)^2$
$\cdots$	$\cdots$	$\cdots$	$\cdots$
p	$(2d)^p$	ε^p	$(2d\varepsilon)^p$

Furthermore, each job J in $T(d, \varepsilon)$ has zero slack time, i.e., $d(J) = r(J) + p(J)$.

To prove $\mathcal{A}$ is not $O(1)$-competitive, we set $d = \frac{1}{2}k^{1/p}$ and $\varepsilon = \frac{1}{pdk}$. Then $d > 1$, $\varepsilon < 1$, and $2d\varepsilon = \frac{2}{pk} \leq 1$. Notice that the job value is decreasing from Category 0 to Category p.

Consider a game played between an adversary using one processor and the online algorithm $\mathcal{A}$ (which uses p processors). The game consists of disjoint stages. At the beginning of each stage, the adversary releases a new job set $T(d, \varepsilon)$, i.e., $p + 1$ jobs. Based on the way $\mathcal{A}$ schedules the jobs in a stage, the adversary determines when the next stage begins.

The first stage begins at time 0. Since $p + 1$ jobs are released and $\mathcal{A}$ uses only p processors, there is a job not processed by $\mathcal{A}$ at time 0. Denote this job as J_0. The adversary schedules J_0 using its only processor. After the adversary completes J_0, the second stage begins and another set of $T(d, \varepsilon)$ is released.

In general, at the beginning of the i-th stage, say, at time t_0, the adversary releases another set of $T(d, \varepsilon)$ and chooses a job J_i to execute as follows. Note that at t_0, the jobs chosen by $\mathcal{A}$ may not have distinct value densities because $\mathcal{A}$ may continue to process some jobs released in previous stages. Let ℓ be the smallest category such that $\mathcal{A}$ processes only ℓ jobs in Categories 0 to ℓ. That is, $\mathcal{A}$ processes ℓ jobs in Categories 0 to $\ell - 1$, but not any in Category ℓ. Let J_i be the the job in Category ℓ just released. Denote $\alpha_i = \ell$ as the category of J_i. The last stage lasts for a time period of 1; this ensures that the deadline of every job released so far is on or before the end of the last stage.

Before we analyze the performance of $\mathcal{A}$, it is useful to note that jobs are released only at the beginning of each stage and they all have zero slack time. Thus, if a job is not scheduled to run on a processor upon release, it will definitely miss its deadline. In other words, in the middle of a stage, it makes no sense to schedule a processor to switch

to another job (which must be released before), and we can assume that within a stage, a processor works on at most one job.

Fact. In any i-th stage, the adversary completes the job J_i, obtaining a value of $v(J_i) = (2d\varepsilon)^{\alpha_i}$.

Lemma 6 *In any i-th stage except the last stage, the value that can be obtained by $\mathcal{A}$ is at most $\frac{1}{d}(2d\varepsilon)^{\alpha_i}$, i.e., at most $v(J_i)/d$.*

Proof. By the definition of α_i, $\mathcal{A}$ at the beginning of the i-th stage works on α_i jobs in Categories 0 to $\alpha_i - 1$ and at most $p - \alpha_i$ jobs in Categories $\alpha_i + 1$ to p. Once the i-th stage has begun, no other jobs will be processed.

Let us first consider jobs in Categories $\alpha_i + 1$ to p. We show that the total value due to such jobs is at most $\frac{1}{d}\varepsilon^{\alpha_i}$. The duration of the i-th stage is ε^{α_i}, which is long enough to complete any job in Categories $\alpha_i + 1$ to p. Each processor scheduled to a job in Category $\ell \geq \alpha_i + 1$ gives a value of $(2d\varepsilon)^{\ell} \leq (2d\varepsilon)^{\alpha_i+1}$. Thus, the total value obtained by such processors is bounded by

$$\begin{aligned} p(2d\varepsilon)^{\alpha_i+1} &\leq p(2d)^p\varepsilon^{\alpha_i+1} \\ &= p(k^{1/p})^p \frac{1}{pdk}\,\varepsilon^{\alpha_i} \\ &= \frac{1}{d}\,\varepsilon^{\alpha_i}. \end{aligned}$$

Next, we consider the jobs in Categories 0 to $\alpha_i - 1$. Recall that these α_i jobs may not have distinct value densities. Nevertheless, by the definition of α_i, the sum of their value densities is at most $1 + 2d + (2d)^2 + ... + (2d)^{\alpha_i-1}$. Assuming all these jobs are processed throughout the i-th stage, the value obtained by $\mathcal{A}$ is at most

$$\begin{aligned} [1 + 2d + (2d)^2 + ... + (2d)^{\alpha_i-1}]\varepsilon^{\alpha_i} &= \frac{(2d)^{\alpha_i} - 1}{2d - 1}\,\varepsilon^{\alpha_i} \\ &\leq \frac{(2d)^{\alpha_i} - 1}{d}\,\varepsilon^{\alpha_i}. \end{aligned}$$

Summing the above two parts together, we conclude that the value obtained by $\mathcal{A}$ during the i-stage is at most $\frac{1}{d}\varepsilon^{\alpha_i} + \frac{(2d)^{\alpha_i}-1}{d}\varepsilon^{\alpha_i}$, which is equal to $\frac{1}{d}(2d\varepsilon)^{\alpha_i}$. This completes the proof of Lemma 6. □

In the last stage, the best $\mathcal{A}$ can do is to complete the first p jobs just released, obtaining a total value of at most p. The adversary on the other hand completes the job with value 1.

Consider an instance of the above game that consists of $h + 1$ stages. Let $\mathcal{O}$ be the value obtained by the adversary in the first h stages. By Lemma 6, the competitive ratio

of $\mathcal{A}$ is at least $(\mathcal{O}+1)/(\mathcal{O}/d+p)$. Notice that p is a constant, and we can choose a sufficient large h to make $\mathcal{O}$ arbitrarily large and $(\mathcal{O}+1)/(\mathcal{O}/d+p)$ arbitrarily close to d. Therefore, the competitive ratio of $\mathcal{A}$ has a lower bound of d, which is defined as $\frac{1}{2}k^{1/p}$. This completes the proof of Theorem 5.

3 Competitiveness via a faster processor

For firm-deadline scheduling on a single processor, the algorithm SLACKER given by Kalyanasundaram and Pruhs [10] is $(1+\delta^{-1})(1+\delta^{-1/2})(1+\delta^{-1/2}+\delta^{-1})$-competitive when using a speed-$(1+2\delta)$ processor for any $\delta > 0$. For example, with a speed-2 processor, the competitive ratio is 32. There is however no similar result known for firm-deadline scheduling on two or more processors. In this section, we give a straightforward extension of SLACKER, denoted MSLACKER below, for scheduling jobs on any $m \geq 1$ processors, and we show that MSLACKER, when given m speed-$(1+2\delta)$ processors for any $\delta > 0$, is $(1+2\delta^{-1}+4\delta^{-2})$-competitive against any offline adversary using m unit speed processors. The competitive ratio of MSLACKER also implies an improvement for SLACKER since SLACKER is a special case of MSLACKER. This improvement is mainly resulted from a new upper bound on the total value of jobs that can be completed by the adversary.

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

Below we analyze the performance of MSLACKER, showing that if c is chosen as $1+2\delta^{-1}$, then MSLACKER when given speed-$(1+2\delta)$ processors is $(1+2\delta^{-1}+4\delta^{-2})$-competitive.

Definition. Consider any input job sequence. Let $\mathcal{O}$ be the set of jobs completed by the optimal offline algorithm, and let $\mathcal{S}$ and $\mathcal{R}$ denote the sets of jobs completed by MSLACKER and ever added into M, respectively. Let $\|\mathcal{S}\|$ be the total value of jobs in $\mathcal{S}$, and similar for $\|\mathcal{O}\|$ and $\|\mathcal{R}\|$. By definition, $\|\mathcal{S}\| \leq \|\mathcal{R}\|$.

Definition. A released job J is said to be *fresh* at any time before $d(J) - (1+\delta)\frac{p(J)}{s}$.

To bound the competitive ratio of MSLACKER, it is equivalent to derive a general

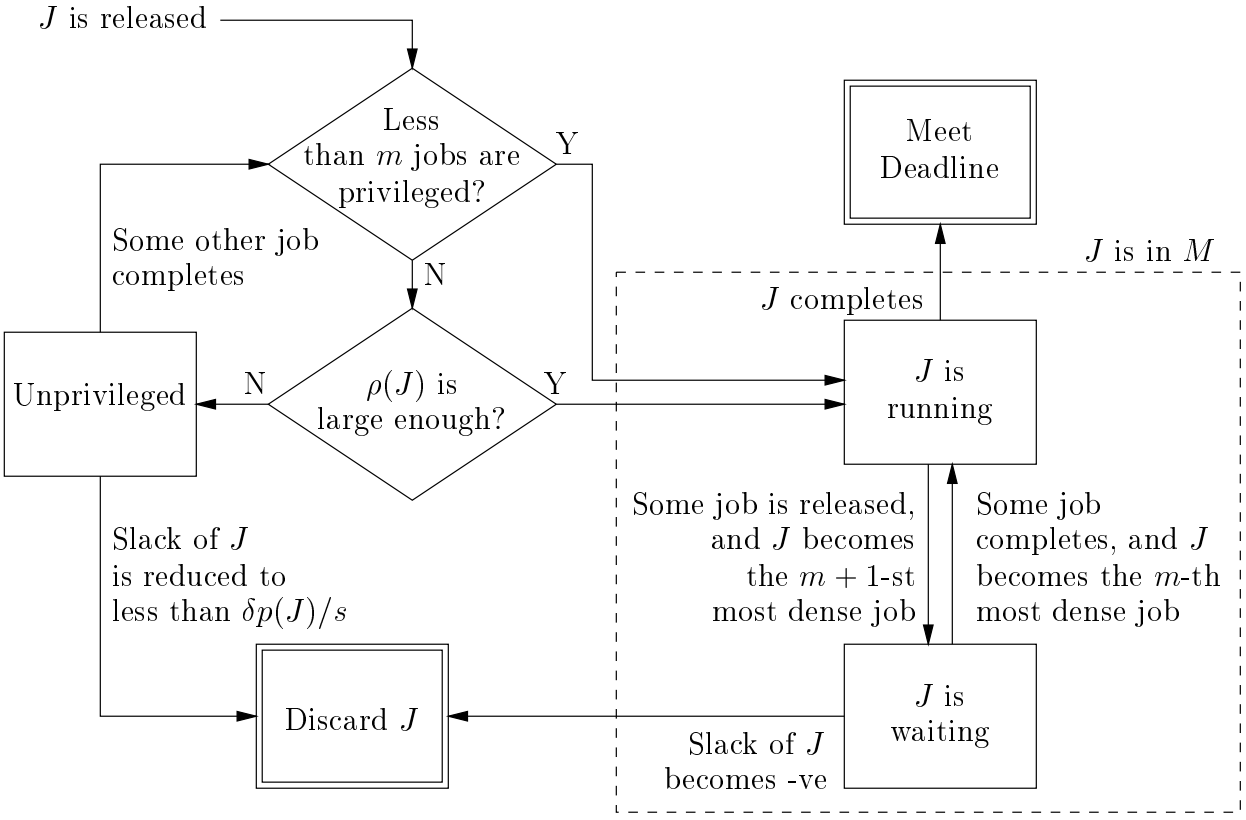

Figure 1: The Life-Cycle of a Job as Scheduled by MSLACKER

upper bound of $\|\mathcal{O}\|/\|\mathcal{S}\|$. That is, we want to determine a lower bound of $\|\mathcal{S}\|$ and an upper bound of $\|\mathcal{O}\|$. Intuitively, MSLACKER is very conservative and can complete most of jobs added into M; specifically, we show that $\|\mathcal{S}\|$ is at least a significant fraction of $\|\mathcal{R}\|$ (see Lemma 10). On the other hand, the adversary may be able to pick some more valuable jobs to execute, but the way MSLACKER selects the jobs guarantees that $\|\mathcal{O}\|$ cannot exceed $\|\mathcal{R}\|$ too much (see Lemma 8).

We first show the upper bound of $\|\mathcal{O}\|$. We consider the upper bound of $\|\mathcal{O}\|$ by analyzing separately the jobs that MSLACKER also meet the deadlines and jobs that MSLACKER miss the deadlines. This is unlike the proof in [10], which consider both types of jobs at the same time. Our approach leads to a simpler and tighter analysis, and makes it applicable to the multi-processor setting. First of all, let us observe a simple property of MSLACKER.

Lemma 7 *At any time t when a job J has not yet completed by* MSLACKER *and J is still fresh, if J is not processed by* MSLACKER*, then* MSLACKER *is processing m other*

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

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

We are now ready to show the upper bound of $\|\mathcal{O}\|$.

Lemma 8 $\|\mathcal{O}\| \leq \|\mathcal{S}\| + \frac{c}{\delta}\|\mathcal{R}\|$.

Proof. Partition $\mathcal{O}$ into $\mathcal{O}^{\mathrm{c}} = \mathcal{O} \cap \mathcal{S}$ and $\mathcal{O}^{\mathrm{u}} = \mathcal{O} - \mathcal{S}$. Since $\mathcal{O}^{\mathrm{c}} \subseteq \mathcal{S}$, $\|\mathcal{O}\| \leq \|\mathcal{S}\| + \|\mathcal{O}^{\mathrm{u}}\|$. Intuitively, each job in $\mathcal{O}^{\mathrm{u}}$ must be fresh for a large proportion of the time when the optimal offline algorithm chooses the job for execution. On the other hand, Lemma 7 guarantees that at such time MSLACKER must have chosen jobs with large value density for execution. Details are as follows.

For any job J in $\mathcal{O}^{\mathrm{u}}$, define $a_1(J)$ (resp. $a_2(J)$) to be the total amount of time when the optimal offline algorithm executes J while J is fresh (resp. J is no longer fresh). Obviously, $a_1(J) + a_2(J) = p(J)$. For any job $J \in \mathcal{O}^{\mathrm{u}}$, $a_2(J) \leq (1+\delta)\frac{p(J)}{s}$, and $a_1(J) = p(J) - a_2(J) \geq \delta\,\frac{p(J)}{s}$ (recall that $s = 1 + 2\delta$). Thus, $\frac{\delta}{s}\,v(J) = \frac{\delta}{s}\,p(J)\rho(J) \leq a_1(J)\rho(J)$, and $\frac{\delta}{s}\|\mathcal{O}^{\mathrm{u}}\| \leq \sum_{J\in\mathcal{O}^{\mathrm{u}}} a_1(J)\rho(J)$.

To derive an upper bound of $\|\mathcal{O}\|$, we consider $a_1(J)$ for each job $J \in \mathcal{O}^{\mathrm{u}}$. By definition, every job $J \in \mathcal{O}^{\mathrm{u}}$ is not completed by MSLACKER. At any time when the adversary executes a job $J \in \mathcal{O}^{\mathrm{u}}$ while J is fresh, Lemma 7 tells us that MSLACKER either executes J, or executes m jobs each of value density at least $\rho(J)/c$. In general, at any time t, let X_t be the set of fresh jobs in $\mathcal{O}^{\mathrm{u}}$ currently executed by the adversary; then for each job $J \in X_t$, we can identify a distinct job currently executed by MSLACKER with job density at least $\rho(J)/c$. In other words, the total value density of jobs in X_t is at most c times the total value density of jobs currently executed by MSLACKER. To bound $\sum_{J\in\mathcal{O}^{\mathrm{u}}} a_1(J)\rho(J)$, it suffices to consider the sum over all time t of the total value density of jobs in X_t, which is at most c times of the sum over all time t of the total value density of jobs executed by MSLACKER at time t. Note that each job $J \in \mathcal{R}$ can contribute a quantity of at most $\rho(J)\frac{p(J)}{s}$ to the latter sum. Thus, the latter sum can be expressed as $\sum_{J\in\mathcal{R}} \rho(J)\frac{p(J)}{s}$, which is equal to $\frac{1}{s}\|\mathcal{R}\|$. In summary, $\sum_{J\in\mathcal{O}^{\mathrm{u}}} a_1(J)\rho(J) \leq \frac{c}{s}\|\mathcal{R}\|$. Therefore, $\delta\|\mathcal{O}^{\mathrm{u}}\| \leq c\|\mathcal{R}\|$, and $\|\mathcal{O}\| \leq \|\mathcal{S}\| + \|\mathcal{O}^{\mathrm{u}}\| \leq \|\mathcal{S}\| + \frac{c}{\delta}\|\mathcal{R}\|$. □

Next, we show the lower bound of $\|\mathcal{S}\|$. The proof is similar to the analysis given in [10]. This similarity roots at the following lemma about MSLACKER.

Lemma 9 *At any time, for any $1 \leq i \leq |M| - m$, the i-th most dense job in M is at least c times as dense as the $(i+m)$-th most dense job in M.*

Proof. We prove the lemma by focusing on an arbitrary job J in M, and show the property that when it is the i-th most dense job in M, the value density of the $(i+m)$-th most dense job in M is no more than $\rho(J)/c$.

It is clear by the rules of MSLACKER that when J is added to M as the i-th most dense job, it must be at least c times as dense as the new $(m+1)$-st most dense job, and thus must be at least c times as dense as the $(i+m)$-th most dense job. So we just need to show that the property is preserved when other jobs are added to and removed from M. Assume that J is the i'-th most dense job in M when such an operation is performed. Obviously, adding or removing a job more dense than J or less dense than the $(i'+m)$-th most dense job does not affect this property. If a job between the $(i'+1)$-st and the $(i'+m)$-th most dense job in M is removed from M, the property is preserved since a less dense job then becomes the $(i'+m)$-th most dense job. Finally, if a job with value density between that of J and the $(i'+m)$-th most dense job in M is added to M, the newly added job must be at least c times as dense as the $(m+1)$-st most dense job, and thus the $(i'+m)$-th most dense job. Since the newly added job has smaller value density than J, the property is again preserved. □

Lemma 10 $\|\mathcal{S}\| \geq \frac{\delta(c-1)-1}{\delta(c-1)}\|\mathcal{R}\|$.

Proof. We use the following amortization scheme. Each job J in $\mathcal{R}$ is associated with an account $\Psi(J)$. If $J \in \mathcal{S}$, $\Psi(J)$ has an initial balance of $v(J)$ credits. Otherwise, $\Psi(J)$ has an initial balance of 0 credit. We show how to transfer credits among the accounts of jobs in $\mathcal{R}$, so that each account has a final balance of at least $v(J) - \frac{v(J)}{\delta(c-1)}$. The total initial balance is $\|\mathcal{S}\|$, while the total final balance is at least $\sum_{J\in\mathcal{R}}(v(J) - \frac{v(J)}{\delta(c-1)}) = (1 - \frac{1}{\delta(c-1)})\|\mathcal{R}\|$. Thus $\|\mathcal{S}\| \geq (1 - \frac{1}{\delta(c-1)})\|\mathcal{R}\|$.

The transfer takes place as follows. Whenever a privileged job has to wait, credits is transferred to the account of the job at a rate of s/δ times its value density, from the account of a running job. More precisely, consider any time when a job J is the i-th most dense job in M, where $i \leq m$. For each job J' being the $(i+m)$-th, $(i+2m)$-th, ..., $(i + \lfloor(|M|-i)/m\rfloor m)$-th most dense job, credits are transfered from $\Psi(J)$ to $\Psi(J')$ at a rate of $s\rho(J')/\delta$.

By definition, the account of each job $J \in \mathcal{S}$ receives $v(J)$ credit initially. On the other hand, the account of each job $J \in \mathcal{R} - \mathcal{S}$, which misses deadline, also receives at least $v(J)$ credit subsequently. This is because when J is added to M, it is fresh and thus has a slack of at least $\delta\, p(J)/s$. For J to miss deadline, it must be idling for at least that amount of time when it is in M. It thus receives at least $(\delta\, p(J)/s)s\rho(J)/\delta = v(J)$ credits.

Finally, by Lemma 9, at a time when J is the i-th most dense job where $i \leq m$, the jobs receiving credits from $\Psi(J)$ have value densities at most $\rho(J)/c$, $\rho(J)/c^2$, ..., respectively. Thus the total rate of transfer from J is less than $(1/c + 1/c^2 + \cdots)s\rho(J)/\delta = \frac{s\rho(J)}{\delta(c-1)}$. Since

J runs for at most $p(J)/s$ time, the amount transfered from $\Psi(J)$ is at most $\frac{v(J)}{\delta\,(c-1)}$. The final balance of each account is thus at least $v(J) - \frac{v(J)}{\delta\,(c-1)}$, as required. □

Based on the above upper bound and lower bound results, we can conclude the competitiveness of MSLACKER.

Theorem 11 MSLACKER, *when given speed-*$(1+2\delta)$ *processors and when* c *is chosen as* $1+2\delta^{-1}$, *is* $(1+2\delta^{-1}+4\delta^{-2})$*-competitive.*

Proof. It follows from Lemmas 8 and 10. □

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, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. *Journal of Real-Time Systems*, 4:124–144, 1992.

[3] 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 IEEE 32nd Annual Symposium on Foundations of Computer Science*, pages 101–110, San Juan, Puerto Rico, 1991.

[4] 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, 8–10 July 1998.

[5] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, The Pitt Building, Trumpington Street, Cambridge, United Kingdom, 1998.

[6] 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, 9–11 January 2000.

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

[8] Jeff Edmonds. Scheduling in the dark. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 179–188, 1999.

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

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

[11] Gilad Koren. Competitive on-line scheduling for overloaded real-time systems. 1993. PhD Thesis, Department of Computer Science, New York University.

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

[13] 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.

[14] 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, 17–19 January 1999.

[15] T.W. Lam and K.K. To. Performance guarantee for online deadline scheduling in the presence of overload. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001. To appear.

[16] 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, 4–6 May 1997.

[17] Jiří Sgall. On-line scheduling — a survery. 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] John Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio Buttazzo. *Deadline scheduling for real-time systems: EDF and related algorithms.* Kluwer Academic Publishers, Boston, Mass., 1998.