Improved Competitive Algorithms for Online Scheduling with Partial Job Values

Francis Y. L. Chin * and Stanley P. Y. Fung

Department of Computer Science and Information Systems,
The University of Hong Kong, Hong Kong.
{chin, pyfung} @csis.hku.hk

Abstract. This paper considers an online scheduling problem arising from Quality-of-Service (QoS) applications. We are required to schedule a set of jobs, each with release time, deadline, processing time and weight. The objective is to maximize the total value obtained for scheduling the jobs. Unlike the traditional model of this scheduling problem, in our model unfinished jobs also get partial values proportional to their amounts processed.
We give a new non-timesharing algorithm GAP for this problem for bounded m, where m is the number of concurrent jobs or the number of weight classes. The competitive ratio is improved from 2 to 1.618 (golden ratio) which is optimal for $m = 2$, and when applied to cases with $m > 2$ it still gives a competitive ratio better than 2, e.g. 1.755 when $m = 3$. We also give a new study of the problem in the multiprocessor setting, giving an upper bound of 2 and a lower bound of 1.25 for the competitiveness. Finally, we consider resource augmentation and show that $O(\log \alpha)$ speedup or extra processors is sufficient to achieve optimality, where α is the importance ratio. We also give a tradeoff result between competitiveness and the amount of extra resources.

1 Introduction

We consider the following online scheduling problem. Given a set of jobs, each job is characterized by a 4-tuple (r, d, p, w) which are the release time, deadline, processing time and weight (value per unit time of processing) respectively. The *span* of a job is the time interval $[r, d]$. A job is *active* at time t if $t \in [r, d]$ and is not completely processed by time t. Preemption is allowed with no penalty, and the goal is to maximize the total value obtained by processing the jobs.

In the traditional model of this problem, only jobs that are completed receive their values, and partially processed jobs receive no value. Recently there is a new model in which jobs that are partially processed receive a partial value proportional to their amounts processed [3, 6, 4, 5]. This model is more relevant in some problem domains, and is first described as a Quality-of-Service (QoS) problem concerning the transmission of large images over a network of low bandwidth

* This work is supported by an RGC research grant.

[3]. This is also related to a problem called *imprecise computation* in real-time systems [11], and has applications in numerical computation, heuristic search, database query processing, etc.

Jobs arrive online, i.e., no details of a job is known before it is released, and the online scheduling algorithm has to made its decisions based only on the details of jobs already released. We assume all details of a job are known at the time it is released. We judge the performance of online algorithms by their competitive ratios [13]. An online algorithm is *c-competitive* if, for any instance of jobs, the value produced by the online algorithm is at least $1/c$ that of the offline optimal algorithm.

Tight bounds on the competitive ratio are known for the traditional model: both the upper and lower bounds are $(1+\sqrt{\alpha})^2$ [1,9], where α denotes the *importance ratio*, i.e., the ratio of maximum to minimum job weights. For previous results on the partial value model, Chang and Yap first gave 2-competitive algorithms and a lower bound of 1.17 on the competitive ratio [3]. The upper bound was then improved to $e/(e-1) \approx 1.58$ [5,6]. The lower bound was also improved to 1.236 [6] and most recently to 1.25 [5].

The $e/(e-1)$-competitive algorithm makes use of *timesharing*, i.e., it allows more than one job running on the processor simultaneously, each at reduced speeds so that the sum of processing speeds at any time does not exceed the processor speed. Although timesharing can be simulated in non-timesharing systems by alternating jobs at a very high frequency, this may not be desirable since it incurs a high cost. In fact we proved that timesharing does help: non-timesharing algorithms cannot be better than 1.618-competitive [5]. No non-timesharing algorithms are known to have competitive ratio $2-\epsilon$ for constant ϵ in general. However in practice there may be additional contraints, e.g. the job weights may not vary too much, or fall into fixed weight classes; or the system would not be too overloaded, i.e., too many jobs released at a short period of time. We can use these information to devise better algorithms. In [4] we give an algorithm which is $(2-1/(\lg\alpha+2))$-competitive[1] . In Sect. 3 we consider the case when there are bounded number of concurrent active jobs, or bounded number of weight classes. Let m be the bound on either one of these. A new online algorithm GAP is proposed which is 1.618-competitive for $m=2$ and is optimal. This new algorithm, although not optimal for $m>2$, gives a competitive ratio better than 2.

All the above results are for the single processor setting, and no previous results for this problem are known in the multiprocessor setting. In Sect. 4 we give a 2-competitive algorithm and a lower bound of 1.25 for the competitiveness in this case.

Using *resource augmentation* as a means of analyzing online algorithms first appeared in [12] and [7]. The idea is to give the online algorithm more resources to compensate for its lack of future information, and analyze the tradeoff between the amount of additional resources and improvement in performance. Since then, many problems are analyzed using this approach. We give a new study of

[1] In this paper lg denotes log to base 2.

applying the resource augmentation analysis to this problem, by using either a faster processor or more processors. The only known result is a lower bound of $\Omega(\log\log\alpha)$ speedup to achieve optimality (1-competitiveness) [6], which applies to both the traditional and partial value models. A $4\lceil\lg\alpha\rceil$ upper bound for the traditional model is known [10, 8]. In Sect. 5 we give the first upper bound results for the partial value model. We also give a tradeoff result between the amount of extra resources and the improvement to competitive ratio. Such tradeoff results also exist for the traditional model [7, 10].

2 Preliminaries

Let $r(q), d(q)$ and $w(q)$ denote the release time, deadline and weight of a job q, respectively. For an algorithm A, let $A(t)$ denote the job running by A at time t, and $done_A(q,t)$ be the amount of work done of job q by A by time t. Without confusion, 'algorithm' and 'schedule' are used interchangeably. If no job is running on A at time t we call $A(t)$ a *null job*. Let OPT denote the offline optimal algorithm. Let $||S||$ denote the value of a schedule S.

A schedule S is *canonical* if for any two times t_1 and t_2 ($t_1 < t_2$), the following is satisfied: if $q_1 = S(t_1)$, and $q_2 = S(t_2)$ is not null, then either (i) $r(q_2) > t_1$, or (ii) q_1 is not null and $d(q_1) \leq d(q_2)$. Intuitively, it means that among the active jobs at any time, S will either schedule the one with the earliest deadline, or discard it forever. We assume ties on deadlines are always broken consistently, for the offline optimal algorithm and the online algorithm, so that we may assume no two deadlines are equal. It can be shown that OPT is canonical [6].

We bound the competitive ratio of online algorithms by employing a *charging scheme* similar to that in [6]. Let A denote an online algorithm. We charge the values of infinitesimally small time periods (i.e. the 'value rates' or weights) from OPT to those in A. Let $F : \Re \to \Re$ be a function mapping each time in OPT to a time in A. For any time t, suppose q is the job currently running in OPT. If $done_{OPT}(q,t) > done_A(q,t)$, $F(t) = t$. Otherwise, find the time $u < t$ when $done_{OPT}(q,t) = done_A(q,u)$, and $F(t) = u$. In both cases the value rate charged is $w(q)$. It can be seen that all job values in OPT are charged under mapping F.

At any time t, there are at most two charges to A (i.e. two times mapped to t by F), one from time t and another from a time later than t. See Fig. 1. Define the *charging ratio* at any time t to be the sum of values of the charges made to t over the value A is getting at time t. If we can bound the charging ratio at time t for all t, this gives a bound on the competitive ratio of A.

3 An Improved Non-timesharing Algorithm

No non-timesharing algorithms are known to be better than 2-competitive for the general case. In this section we give a non-timesharing algorithm that achieves an improved competitive ratio when the number of concurrent jobs is bounded, or the number of weight classes is bounded.

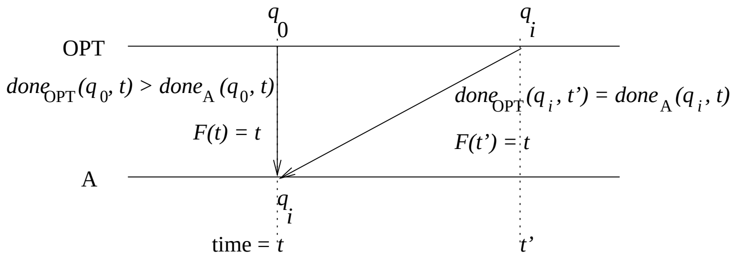

Fig. 1. Charging scheme.

An active job x *dominates* another active job y if $w(x) \geq w(y)$ and $d(x) < d(y)$. An active job is *dominant* if no other active job dominates it. Let w_i denote the weight of the i-th heaviest active dominant job at any time. Note that no two active dominant jobs have equal weight.

3.1 Algorithm GAP

Suppose there are at most m active dominant jobs at any moment, where m is known in advance. Later we will see that this assumption generalizes the two conditions mentioned before (bounded number of concurrent jobs or bounded number of weight classes). Intuitively, our algorithm tries to find a sufficiently heavy job which, at the same time, has a weight far away from other lighter jobs. This helps to give a good charging ratio by avoiding jobs with similar weights to charge to one point in time. Formally, algorithm GAP uses a parameter $r > 1$, which depends on m, and is the unique positive real root of the equation $r = 1 + r^{-1/(m-1)}$. The following table shows some values of r and m.

m	2	3	4	5	10	20	∞
r	1.618	1.755	1.819	1.857	1.930	1.965	2

When GAP is invoked, it first finds all active dominant jobs with weights $\geq (1/r)w_1$ (w_1 is the weight of the currently heaviest job). Call this set S. Among jobs in S, find a job q such that, for any other active dominant job q' with $w(q') < w(q)$, we have $w(q)/w(q') \geq r^{1/(m-1)}$. (Note that q' may not be in S). Schedule q. (We will show that such a q always exists. If there are more than one such q then schedule any one of them.) GAP is invoked again when some job is finished, reached its deadline, or a new job arrives.

3.2 Analysis

Theorem 1. *For a system with at most m active dominant jobs at any time, GAP is r-competitive, where r the unique positive real root of $r = 1 + r^{-1/(m-1)}$.*

Proof. We first show that there must be a job that satisfies the above criteria to be scheduled. Let $w_1, w_2, ..., w_p$ be the weights of the active dominant jobs in S. If $w_i/w_{i+1} \geq r^{1/(m-1)}$ for some i in 1, 2, ..., $p-1$, we are done. Hence suppose $w_1/w_p < r^{(p-1)/(m-1)}$. If there is no other active dominant job outside S, then the job with weight w_p can be scheduled. Otherwise, let w_{p+1} be the weight of the heaviest active dominant job outside S, $p+1 \leq m$. We have $w_p > r^{-(p-1)/(m-1)} w_1 \geq r^{-(m-2)/(m-1)} w_1$ and thus $w_p/w_{p+1} > r^{-(m-2)/(m-1)}/(1/r) = r^{1/(m-1)}$. Therefore the job with weight w_p can be scheduled.

By the definition of S and the way the algorithm works, we have the following properties of GAP: for any job y picked by GAP,
(1) $w(y) \geq w_1/r$
(2) no other active dominant job x satisfies $r^{-1/(m-1)} w(y) < w(x) \leq w(y)$.

We use the charging scheme in Sect. 2. Suppose at time t, x, y are the jobs running in OPT, GAP respectively. We consider the charges made to job y. y may receive charges from x and/or charges from y from a later time in OPT. There are three cases:

- Case 1. x does not charge to y. In this case charging ratio $= w(y)/w(y) = 1$.
- Case 2. Only x charges to y. Since x must be active in GAP, and GAP always choose jobs within $1/r$ of the maximum weight (Property (1)), we have charging ratio $= w(x)/w(y) \leq w(x)/(w_1/r) \leq r$.
- Case 3. Both x and y charge to y. By definition of the mapping F, both x and y should be active in GAP at time t. By canonical property of OPT, $d(x) < d(y)$. Therefore $w(x) < w(y)$, or else x dominates y and y would not be scheduled in GAP.
  (i) if x is dominant in GAP, then by Property (2) of GAP, $w(x)/w(y) \leq r^{-1/(m-1)}$.
  (ii) if x is not dominant in GAP, then suppose $z \neq x$ is the 'next' (smaller-weight) active dominant job after y. By Property (2) of GAP, $w(z)/w(y) \leq r^{-1/(m-1)}$, and we must have $w(x) \leq w(z)$ (because y and z are consecutive dominant jobs, there cannot be an active job with weight $> w(z)$ and deadline $< d(y)$), so again we have $w(x)/w(y) \leq r^{-1/(m-1)}$.
  In both cases, the charging ratio $= (w(x) + w(y))/w(y) = 1 + w(x)/w(y) \leq 1 + r^{-1/(m-1)}$.

Therefore, in any case charging ratio $\leq \max(r, 1 + r^{-1/(m-1)})$. This is minimized by setting r to be the root of $r = 1 + r^{-1/(m-1)}$. In this case, the competitive ratio is r. □

The above proof only uses the assumption that there are at most m active dominant jobs at any time. Note that whether jobs are active/dominant or not depends on how the algorithm schedules them, not just the instance itself. This is not desirable. However the theorem is still true for the following models, which are more realistic and generalized by the above:

Corollary 1. *GAP is r-competitive if*
(i) at any time t there are at most m jobs with t in their span; or
(ii) there are at most m weight classes, i.e., all jobs are of weight $w_1, w_2, ..., w_m$ for some fixed w_i's.

Proof. (i) automatically implies there are at most m active dominant jobs, while (ii) means there are at most m jobs having different weights at any time, thus at most m active dominant jobs. □

In [5] it is proved that no non-timesharing algorithms are better than ϕ-competitive, where $\phi \approx 1.618$ is the golden ratio, and the construction is for $m = 2$. When $m = 2$, GAP chooses r to be the root of $r = 1 + 1/r$, i.e. $r = \phi$. Thus we have

Corollary 2. *GAP is an optimal non-timesharing algorithm with competitive ratio 1.618 when $m = 2$.*

4 The Multiprocessor Case

In this section, we consider the partial job value scheduling problem in a multiprocessor setting. The online algorithm has M processors, and we compare its performance with an offline optimal algorithm also having M processors. Throughout the paper we assume jobs are migratory, i.e., jobs on a processor can be switched to other processors to continue processing.

First consider the upper bound. For the uniprocessor case, FirstFit (i.e., always schedule the heaviest job) is 2-competitive [3]. The same holds true for the multiprocessor case, in which FirstFit always schedules the M heaviest jobs (if there are less than M active jobs, then some processors will idle).

Theorem 2. *In the multiprocessor setting, FirstFit is 2-competitive, and this is tight.*

Proof. We use the charging scheme in Sect. 2. Suppose at time t, $j_1, ..., j_M$ are the M jobs running on the processors of FirstFit, $w(j_1) \geq w(j_2) \geq ... \geq w(j_M)$, and suppose $q_1, ..., q_M$ are jobs running on the processors of the offline optimal algorithm. Some of the q_i's may be the same as some of the j_i's. Without loss of generality assume $j_i = q_i$ for $i \in I \subset \{1, 2, ..., M\}$ (reordering indices of q_i's as necessary). Consider the charges to a certain time t. For those $i \in I$, j_i can charge to time t at most once. For those $i \notin I$, q_i either do not charge to time t, or if they do, then we must have $w(q_i) \leq w(j_M)$ since they are unfinished but not chosen by FirstFit. For these i's, j_i may also charge to t from a later time in OPT. Thus the charging ratio is given by

$$\begin{aligned} c &\leq \frac{\sum_{i \in I} w(q_i) + (M - |I|)w(j_M) + \sum_{i \notin I} w(j_i)}{\sum_{i=1}^{M} w(j_i)} \\ &= \frac{\sum_{i=1}^{M} w(j_i) + (M - |I|)w(j_M)}{\sum_{i=1}^{M} w(j_i)} \leq \frac{\sum_{i=1}^{M} w(j_i) + M w(j_M)}{\sum_{i=1}^{M} w(j_i)} \leq 2 \end{aligned}$$

Consider the following instance of jobs: M copies of (0, 2, 1, 1+ϵ), and M copies of (0, 1, 1, 1), where $\epsilon > 0$ is very small. FirstFit schedules all weight-(1+ϵ) jobs and misses all weight-1 jobs, while OPT can schedule all of them. Thus the competitive ratio $\geq (M + M(1+\epsilon))/(M(1+\epsilon)) \approx 2$. □

Next we consider the lower bound. In [5] we proved a randomized lower bound of 1.25 for the uniprocessor case. Here we extend the result to the multiprocessor case, with a proof similar to that in [5].

Theorem 3. *No randomized (and hence deterministic) algorithms can be better than 1.25-competitive for any M .*

Proof. (Sketch) We make use of Yao's principle [14]. Basically, it enables us to find a lower bound of randomized algorithms by finding a probability distribution of instances, such that we can bound the ratio of the expected offline optimal value to the expected online value of the best *deterministic* algorithm. This ratio will then be a lower bound of randomized algorithms (see [2]).

Consider a set of $n+1$ instances:

$J_1 = M$ copies of $\{(0,1,1,1),(0,2,1,2)\}$

$J_i = J_{i-1} \cup M$ copies of $\{(i-1,i,1,2^{i-1}),(i-1,i+1,1,2^i)\}$, for $i = 2, ..., n$

$J_{n+1} = J_n \cup M$ copies of $\{(n,n+1,1,2^n)\}$

We form a probability distribution of J_i's with p_i being the probability of picking J_i: $p_i = 1/2^i$ for $i = 1,2,...,n$ and $p_{n+1} = 1/2^n$. Clearly $\sum p_i = 1$.

First consider the offline optimal value. It is easy to see that $||OPT(J_i)|| = (2+2^2...+2^i)M+2^{i-1}M = [2(2^i-1)+2^{i-1}]M$ for $i = 1,2,...,n$, and $||OPT(J_{n+1})||$ $= (2+2^2...+2^n)M + 2^nM = [2(2^n-1)+2^n]M$. Thus

$$E[||OPT||] = \sum_{i=1}^{n} \frac{2(2^i-1)+2^{i-1}}{2^i}M + \frac{2(2^n-1)+2^n}{2^n}M = (\frac{5n}{2}+1)M$$

Fix a deterministic online algorithm A. At any time interval $[i-1,i]$ where i is an integer, A is faced with M heavier jobs and M or more lighter jobs. Suppose it spends β_i processor-time (total amount of time available on all processors) on lighter jobs in this time interval (and hence $M-\beta_i$ on heavier jobs). The β_i's completely determine the value obtained by this algorithm (on these instances). We can show that, for $i = 1,2,...,n$,

$$||A(J_i)|| = [\beta_1 + 2(M-\beta_1)] + ... + [2^{i-1}\beta_i + 2^i(M-\beta_i)] + 2^i\beta_i$$

$$||A(J_{n+1})|| = [\beta_1 + 2(M-\beta_1)] + ... + [2^{n-1}\beta_n + 2^n(M-\beta_n)] + 2^nM$$

$$E[||A||] = \frac{1}{2}||A(J_1)|| + \frac{1}{4}||A(J_2)|| + ... + \frac{1}{2^n}||A(J_n)|| + \frac{1}{2^n}||A(J_{n+1})||$$

We can show that all coefficients of β_i's in $E[||A||]$ vanish. Thus $E[||A||]$ only depends on the constant terms, and

$$E[||A||] = \sum_{i=1}^{n} \frac{1}{2^i}(2+...+2^i)M + \frac{1}{2^n}(2+...+2^n+2^n)M = (2n+1)M$$

Hence $\dfrac{E[||OPT||]}{E[||A||]} = \dfrac{5n/2+1}{2n+1} \to \dfrac{5}{4}$ as n is very large. Thus no randomized algorithms have competitive ratio better than 1.25. □

5 Resource Augmentation

How much extra resources is required to get 1-competitive algorithms? In this section we give an algorithm that requires $2\lceil \lg \alpha \rceil$ extra resources to achieve 1-competitiveness, in contrast with the $\Omega(\log \log \alpha)$ lower bound [6]. It is in parallel to, but smaller than, the $4\lceil \lg \alpha \rceil$ bound for the traditional model [10, 8].

5.1 Importance Ratio = 2

We first consider the case $\alpha = 2$. We use EDF (Earliest Deadline First, i.e., always schedule the job with the earliest deadline) with a speed-2 processor (a speed-s processor is one which has speed s times that of a normal processor).

Theorem 4. *For $\alpha = 2$, EDF with a speed-2 processor is necessary and sufficient to achieve 1-competitiveness.*

Proof. We use a charging scheme almost identical to that stated in Sect. 2. The only difference is that, when $done_{OPT}(q,t) < done_{EDF}(q,t)$ and $done_{OPT}(q,t) = done_{EDF}(q,u)$ for $u < t$ (i.e., charge from t to u), the charges made to time u is $2w(q)$, i.e., twice the weight, to account for the difference in speeds between the offline and online algorithms. Suppose at time t, job q_0 is running in OPT, q_1 is running in EDF. Note that EDF is getting a value of $2w(q_1)$ every unit time since it is running at speed-2. Consider the charges to time t, it consists of a $w(q_0)$ charge from time t and/or a $2w(q_1)$ charge from a later time.

- Case 1. q_0 does not charge to t. Charging ratio $c \leq (2w(q_1))/(2w(q_1)) = 1$.
- Case 2. q_0 charges to t. Thus q_0 is unfinished at time t in EDF, therefore q_1 cannot be null. Suppose all job weights are normalized to be in the range [1,2], then $w(q_0) \leq 2, w(q_1) \geq 1$. If there are no other charges (from later times in OPT), then $c \leq w(q_0)/(2w(q_1)) \leq 1$. Suppose q_1 charges from a later time $u > t$ in OPT. Since OPT is canonical, $d(q_0) \leq d(q_1)$, thus EDF should schedule q_0 instead of q_1. The only possibility is then $q_0 = q_1$, but in this case we still have $c \leq (2w(q_1))/(2w(q_1)) = 1$ since q_0 cannot charge to t at two different times.

Since at any time $c \leq 1$, we showed that EDF is 1-competitive.

Consider using a speed-s processor with $s < 2$. Let $s = 2 - \delta, 0 < \delta < 1$ and $0 < \epsilon < \delta/(1-\delta)$. Consider the instance consisting of a job $(0, 2+\epsilon, 2+\epsilon, 2)$ and 4 copies of $(0, 2, 1, 1)$. OPT gets a value of $4 + 2\epsilon$ by executing the heaviest job. Speed-s EDF gets a value of $2s + 2s\epsilon$. Then $2s + 2s\epsilon = 4 - 2\delta + 4\epsilon - 2\delta\epsilon = (4 + 2\epsilon) + 2\epsilon - 2\delta(1+\epsilon)$. This is smaller than $4 + 2\epsilon$ if $2\epsilon < 2\delta(1+\epsilon)$, i.e. $\epsilon < \delta/(1-\delta)$. □

Due to its sequential nature, a job cannot be running on two processors simultaneously. Thus a speed-2 processor is more powerful than two speed-1 processors, since it can simulate two speed-1 processors by timesharing but not vice versa. However, we still have the following stronger result, using extra processors instead of higher-speed processor to achieve 1-competitiveness. We again use EDF, i.e., the two processors will schedule the earliest-deadline and second-earliest-deadline job respectively. Due to space limitation we omit the proof.

Theorem 5. *EDF with two speed-1 processors is 1-competitive for* $\alpha = 2$.

5.2 General Importance Ratio

In fact, the proof of Theorem 4 can easily be generalized to show that EDF with speed-s processor is α/s-competitive. Thus 1-competitiveness can be achieved for general values of α by using a speed-α processor. However, we can do better: the following algorithm *Grouped-EDF* partitions the weight ranges $[1..\alpha]$ into $\lceil \lg \alpha \rceil$ classes, each having weights in the interval $[1, 2), [2, 4)$, ..., $[2^{\lfloor \lg \alpha \rfloor - 1}, 2^{\lfloor \lg \alpha \rfloor})$, $[2^{\lfloor \lg \alpha \rfloor}, \alpha]$. Jobs in each class have importance ratio at most 2. The algorithm assigns two speed-1 processors to process jobs in each class using EDF.

Theorem 6. *Grouped-EDF is 1-competitive using* $2\lceil \lg \alpha \rceil$ *speed-1 processors.*

Proof. Let OPT_i and EDF_i be the 1-processor optimal and 2-processor EDF schedules for scheduling the sub-instance consisting of only the i-th class jobs, respectively. By Theorem 5, $||OPT_i|| \leq ||EDF_i||$ for all i. We also have $||OPT|| \leq \sum ||OPT_i||$ because the processors in each OPT_i can always schedule at least that much obtained by the subset of jobs in OPT restricted to that class. Thus $||OPT|| \leq \sum ||OPT_i|| \leq \sum ||EDF_i|| = ||EDF||$. □

By using higher-speed processors and timesharing to simulate multiple processors, and by Theorem 4, we have:

Theorem 7. *Grouped-EDF is 1-competitive with a speed-*$(2\lceil \lg \alpha \rceil)$ *processor, or with* $\lceil \lg \alpha \rceil$ *speed-2 processors.*

An $O(\log \alpha)$-speed processor may not be practical. If we allow timesharing, we can use a speed-s version of the algorithm MIX in [5] to give a tradeoff between speedup and competitive ratio. The proof is similar to the $e/(e-1)$-competitiveness upper bound proof in [5] and is therefore omitted.

Theorem 8. *The speed-s version of MIX is* $1/(1 - e^{-s})$*-competitive.*

A small amount of additional processing power can give very good competitiveness results, irrespective of α. For example, with $s = 2$ we have $c = 1.16$, and with $s = 5, c = 1.00678$, i.e. just 0.68% fewer than the optimal value.

6 Conclusion

In this paper we consider the online scheduling problem with partial job values, and give new results in the non-timesharing case, the multiprocessor case, and the resource augmentation analysis. Some questions remain open. Most importantly, we do not know whether there are non-timesharing algorithms with competitive ratio better than 2. Another problem is about the exact speedup required for achieving 1-competitiveness: both the traditional and partial value models have bounds $\Omega(\log\log\alpha)$ and $O(\log\alpha)$. Would their true bounds be different? (The partial value model seems 'easier': it has 2-competitive algorithms whereas there is a lower bound of 4 in the traditional model [1].)

References

1. 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, *Real-Time Systems* 4, 125–144, 1992.
2. A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, New York, 1998.
3. E. Chang and C. Yap, Competitive Online Scheduling with Level of Service, *Proceedings of 7th Annual International Computing and Combinatorics Conference*, 453–462, 2001.
4. F. Y. L. Chin and S. P. Y. Fung, Online Scheduling with Partial Job Values and Bounded Importance Ratio, *Proceedings of International Computer Symposium*, 787–794, 2002.
5. F. Y. L. Chin and S. P. Y. Fung, Online Scheduling with Partial Job Values: Does Timesharing or Randomization Help? to appear in *Algorithmica*.
6. M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichý and N. Vakhania, Preemptive Scheduling in Overloaded Systems, preliminary version appeared in *Proceedings of 29th International Colloqium on Automata, Languages and Programming*, 800–811, 2002.
7. B. Kalyanasunaram and K. Pruhs, Speed is as Powerful as Clairvoyance, *Journal of the ACM* 47(4), 617–643, 2000.
8. C.-Y. Koo, T.-W. Lam, T.-W. Ngan and K.-K. To, Extra Processors versus Future Information in Optimal Deadline Scheduling, *Proceedings of 15th ACM Symposium on Parallelism in Algorithms and Architectures*, 133–142, 2002.
9. G. Koren and D. Shasha, D^{over}: An Optimal On-line Scheduling Algorithm for Overloaded Uniprocessor Real-time Systems, *SIAM Journal on Computing* 24, 318–339, 1995.
10. T.-W. Lam and K.-K. To, Performance Guarantee for Online Deadline Scheduling in the Presence of Overload, *Proceedings of 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, 755–764, 2001.
11. J. W. S. Liu, K.-J. Lin, W.-K. Shih, A. C.-S. Yu, J.-Y. Chung and W. Zhao, Algorithms for Scheduling Imprecise Computations, *IEEE Computer* 24(5), 58–68, 1991.
12. C. A. Philips, C. Stein, E. Torng and J. Wein, Optimal Time-critical Scheduling via Resource Augmentation, *Proceedings of 29th Annual ACM Symposium on Theory of Computing*, 140–149, 1997.
13. D. Sleator and R. Tarjan, Amortized Efficiency of List Update and Paging Rules, *Communications of the ACM* 28(2), 202–208, 1985.
14. A. C.-C. Yao, Probabilistic Computations: Toward a Unified Measure of Complexity, *Proceedings of 18th IEEE Symposium on Foundations of Computer Science*, 222–227, 1977.