Available online at www.sciencedirect.com
SCIENCE DIRECT
Theoretical Computer Science 325 (2004) 467–478
Theoretical Computer Science
www.elsevier.com/locate/tcs

Improved competitive algorithms for online scheduling with partial job values

Francis Y.L. Chin[1], Stanley P.Y. Fung*

Department of Computer Science and Information Systems, The University of Hong Kong, Hong Kong

Received 10 October 2003; accepted 20 February 2004

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.

No non-timesharing algorithm for this problem with competitive ratio better than 2 is known. We give a new non-timesharing algorithm GAP that improves this ratio for bounded values of m, where m can be 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, showing that in fact a small amount of extra resources is sufficient for achieving close-to-optimal competitiveness.

Keywords: Online algorithms; Scheduling; Partial job values; Resource augmentation

* Corresponding author.
E-mail addresses: chin@csis.hku.hk (F.Y.L. Chin), pyfung@csis.hku.hk (S.P.Y. Fung).

[1] This work is supported by RGC Grant HKU7142/03E.

0304-3975/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.02.046

1. Introduction

We consider the following online scheduling problem. We are given a set of jobs, each 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. Preemption is allowed with no penalty, and the goal is to maximize the total value obtained in 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 (but not completed) still 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 [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 make 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 [13,2]. An online algorithm is *c-competitive* if, for any instance of jobs, the value obtained 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 concurrently, each at reduced speeds so that the sum of processing speeds at any time does not exceed the processor speed. Timesharing can be simulated in non-timesharing systems by alternating jobs at a very high frequency, however, this may not be desirable since it incurs a high cost. We therefore require a *non-timesharing* algorithm to be one that cannot switch jobs at arbitrarily small time intervals [5]. In particular, when all time parameters are integers, a non-timesharing algorithm can only change its job at integral times. In fact we proved that timesharing algorithms are indeed more powerful: non-timesharing algorithms cannot be better than ϕ-competitive, where $\phi = (\sqrt{5}+1)/2 \approx 1.618$ is the golden ratio [5]. No non-timesharing algorithms are known to have competitive ratio $2-\varepsilon$ for constant ε in general.

We want to develop non-timesharing algorithms for this problem with better competitive ratios. The best non-timesharing algorithms so far are the FirstFit and EndFit algorithms given in [3] which are both 2-competitive. In practice, there may be additional constraints on the job instances, e.g. the job weights may not differ by too much, or fall into fixed weight classes; or the system would not be too overloaded, i.e., too many jobs released in a short period of time. We can use these information to devise better algorithms. In [4] we

give an algorithm which is $(2 - 1/(\lceil \lg \alpha \rceil + 2))$-competitive,[2] which gives a better ratio when the job weights are within a small range.

In Section 3 we consider the case when there are a 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 ϕ-competitive for $m = 2$ and is optimal. This new algorithm, although not optimal for $m > 2$, gives a competitive ratio better than 2.

The improvement of the GAP algorithm comes from two observations. First, observe that if there are two jobs with the same (or very close) weights, we should only consider scheduling the job with earlier deadline. Similarly, for two jobs with the same or very close deadlines, one should only consider scheduling the heavier job. We capture this information by introducing the concept of 'dominant jobs', to be defined in Section 2. This also allows us to unify the two cases mentioned above (bounded number of concurrent jobs, or bounded number of weight classes).

Second, notice that not only are the job weights important in scheduling decisions, but also how close the job weights are. For example, with two jobs (0, 1, 1, 1) and (0, 2, 1, 1.01), scheduling the weight-1.01 job in time [0, 1] gives up the weight-1 job, but the optimal algorithm can schedule both. The same applies when there are more than two jobs. Thus the small differences in weights ('gaps') should be considered, and one probably should avoid scheduling jobs with small 'gap'. Our algorithm gives a balance between scheduling heavy jobs and jobs with small gaps. This reflects into a better utilization and analysis of the 'charging scheme', to be described in the next section, which is the key analysis technique we used.

All the above results are for the single processor setting, and no previous results for this problem are known in the multiprocessor setting. In Section 4 we give the first such results: a 2-competitive algorithm and a lower bound of 1.25 for the competitiveness. They are generalizations of previous uniprocessor techniques and results.

Using *resource augmentation* as a means of analyzing online algorithms first appeared in [12,7]. The idea is to give the online algorithm more resources to compensate for its lack of future information, and analyze the trade-off 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 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 Section 5 we give the first upper bound results for the partial value model, showing that a $O(\log \alpha)$ factor of more resources (either faster processors or more processors) can achieve optimality. This is achieved by showing that a simple earliest-deadline-first algorithm can give optimality for small values of α. This does not scale up well for large values of α, but we use a grouping technique to improve the bound. 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

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

[7,10]. Our result shows that using a fairly small amount of extra resources is sufficient to achieve a close-to-optimal competitive ratio.

2. Preliminaries

Let $r(q), d(q)$ and $w(q)$ denote the release time, deadline and weight of a job q, respectively. The *span* of a job is the time interval $[r(q), d(q)]$. A job is *active* at time t if $t \in [r(q), d(q)]$ and is not completely processed by time t. 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 scheduled on A at time t we call $A(t)$ a *null job*. Let OPT denote the offline optimal algorithm, and 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 $d(q_1) \leqslant d(q_2)$. 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 $done_{\mathrm{OPT}}(q, t) > done_A(q, t)$, $F(t) = t$. Otherwise, find the time $u < t$ when $done_{\mathrm{OPT}}(q, t) = done_A(q, u)$, 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.

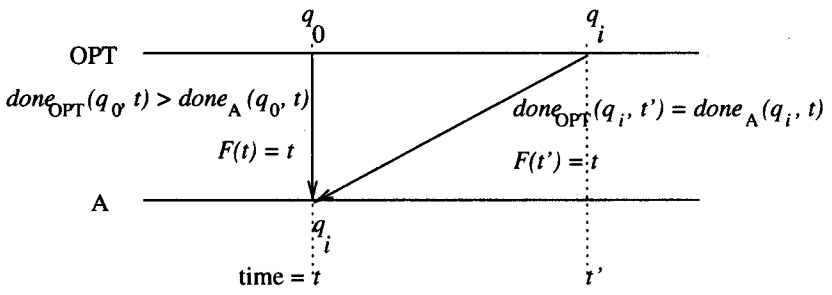

Fig. 1. Charging scheme.

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.

An active job x *dominates* another active job y if $w(x) \geqslant w(y)$ 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 ith 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).

The algorithm is designed to exploit the full power of charging schemes. 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 $r = 1 + r^{1/(1-m)}$. 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 $\geqslant (1/r)w_1$ (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 $w(q)/w(q') \geqslant r^{1/(m-1)}$. (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* $r = 1 + r^{1/(1-m)}$.

Proof. We first show that there must be a job that satisfies the above criteria to be scheduled. Let $w_1, w_2, \ldots, w_p$ be the weights of the active dominant jobs in S. If $w_i/w_{i+1} \geqslant r^{1/(m-1)}$ for some i in $1, 2, \ldots, p-1$, 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,

$p+1 \leqslant m$. We have $w_p > r^{-(p-1)/(m-1)} w_1 \geqslant r^{-(m-2)/(m-1)} w_1$ 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) $w(y) \geqslant w_1/r$

(2) no other active dominant job x satisfies $r^{1/(1-m)} w(y) < w(x) \leqslant w(y)$.

We use the charging scheme in Section 2. Suppose at time t, x and y are the jobs running in OPT and 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 $= w(x)/w(y) \leqslant w(x)/(w_1/r) \leqslant r$.

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 the 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, $w(x)/w(y) \leqslant r^{1/(1-m)}$.

(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, $w(z)/w(y) \leqslant r^{1/(1-m)}$, and we must have $w(x) \leqslant w(z)$ (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 $w(x)/w(y) \leqslant r^{1/(1-m)}$.

In both cases, the charging ratio $= (w(x)+w(y))/w(y) = 1+w(x)/w(y) \leqslant 1+r^{1/(1-m)}$.

Therefore, in any case charging ratio $\leqslant \max(r, 1+r^{1/(1-m)})$. This is minimized by setting r to be the root of $r = 1+r^{1/(1-m)}$. 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 2. *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* $w_1, w_2, \ldots, w_m$ *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, 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 3. *GAP is an optimal non-timesharing algorithm with competitive ratio* ϕ *when* $m = 2$.

GAP as described above assumes the value of m is known a priori. In fact GAP can also be used even when m is not known in advance: just use the number of active dominant jobs at the time instance when GAP is invoked and use this as the value of m to compute the corresponding r. At different times in the course of running GAP, the value of r is therefore different. Since the proof only bounds the charging ratio at each individual time, the overall competitive ratio is bounded above by the maximum of all charging ratios, which is the one when m is largest.

Note that GAP gives a competitive ratio better than 2 when m is bounded, while the algorithm in [4] has competitive ratio better than 2 when α (the importance ratio) is bounded.

4. The multiprocessor case

In this section, we consider the partial job value scheduling problem in a multiprocessor setting. We compare the performance of an online algorithm having M processors with an offline optimal algorithm also having M processors. We assume jobs are migratory, i.e., jobs on a processor can be switched to other processors to continue processing, but the same job cannot be run on more than one processor at any time.

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

Theorem 4. *In the multiprocessor setting, FirstFit is 2-competitive, and this is tight.*

Proof. We use the charging scheme in Section 2. Suppose at time t, $j_1, \ldots, j_M$ are the M jobs running on the processors of FirstFit, $w(j_1) \geqslant w(j_2) \geqslant \ldots \geqslant w(j_M)$, and suppose $q_1, \ldots, q_M$ 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 $i \in I \subset \{1, 2, \ldots, M\}$ (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 $w(q_i) \leqslant w(j_M)$ 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

$$c \leqslant \frac{\sum_{i\in I} w(q_i) + (M - |I|)w(j_M) + \sum_{i\notin I} w(j_i)}{\sum_{i=1}^{M} w(j_i)}$$

$$= \frac{\sum_{i=1}^{M} w(j_i) + (M - |I|)w(j_M)}{\sum_{i=1}^{M} w(j_i)} \leqslant \frac{\sum_{i=1}^{M} w(j_i) + Mw(j_M)}{\sum_{i=1}^{M} w(j_i)} \leqslant 2.$$

Consider the following instance of jobs: M copies of (0, 2, 1, 1+ε), and M copies of (0, 1, 1, 1), where $\varepsilon > 0$ is very small. (Recall that (r, d, p, w) are the release time, deadline, processing time and weight respectively.) FirstFit schedules all weight-(1+ε) jobs and misses

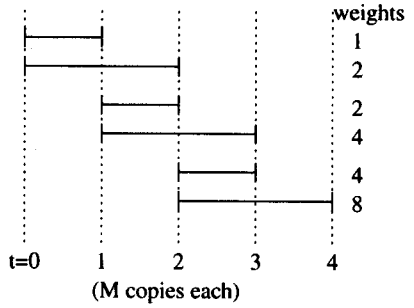

Fig. 2. Lower bound construction, showing J_3 as an example.

all weight-1 jobs, while OPT can schedule all of them. Thus the competitive ratio $\geqslant (M + M(1+\varepsilon))/(M(1+\varepsilon)) \approx 2$. □

Next we consider the lower bound. In [5] we proved a randomized lower bound of $\frac{5}{4}$ for the uniprocessor case. Here we extend the result to the multiprocessor case with a similar proof.

Theorem 5. *No randomized (and hence deterministic) algorithms can be better than $\frac{5}{4}$-competitive for any M (for both timesharing and non-timesharing algorithms).*

Proof. 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 (see Fig. 2):

$$\begin{aligned} J_1 &= M \text{ copies of } \{(0,1,1,1),(0,2,1,2)\},\\ J_i &= J_{i-1} \cup M \text{ copies of } \{(i-1,i,1,2^{i-1}),(i-1,i+1,1,2^i)\},\\ &\quad \text{for } i = 2,\ldots,n\\ J_{n+1} &= J_n \cup M \text{ copies of } \{(n,n+1,1,2^n)\}. \end{aligned}$$

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 $i = 1,2,\ldots,n$ and $p_{n+1} = 1/2^n$. Clearly $\sum p_i = 1$. Consider the offline optimal value. Here $\mathrm{OPT}(J_i)$ means the optimal schedule of instance J_i. It is easy to see that, for $i = 1,2,\ldots,n$,

$$\begin{aligned} \|\mathrm{OPT}(J_i)\| &= (2+2^2+\cdots+2^i)M + 2^{i-1}M\\ &= (2(2^i-1)+2^{i-1})M = (5(2^{i-1})-2)M,\\ \|\mathrm{OPT}(J_{n+1})\| &= (2+2^2+\cdots+2^n)M + 2^n M\\ &= (2(2^n-1)+2^n)M = (3(2^n)-2)M. \end{aligned}$$

Thus

$$E[\|\mathrm{OPT}\|] = \sum_{i=1}^{n} \frac{5(2^{i-1})-2}{2^i} M + \frac{3(2^n)-2}{2^n} M$$
$$= \sum_{i=1}^{n} \left(\frac{5}{2} - \frac{2}{2^i}\right) M + \left(3 - \frac{2}{2^n}\right) M$$
$$= \left(\frac{5n}{2} + 1\right) M.$$

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 $i = 1, 2, \ldots, n$,

$$\|A(J_i)\| = [\beta_1 + 2(M-\beta_1)] + \cdots + [2^{i-1}\beta_i + 2^i(M-\beta_i)] + 2^i\beta_i,$$
$$\|A(J_{n+1})\| = [\beta_1 + 2(M-\beta_1)] + \cdots + [2^{n-1}\beta_n + 2^n(M-\beta_n)] + 2^n M,$$
$$E[\|A\|] = \frac{1}{2}\|A(J_1)\| + \frac{1}{4}\|A(J_2)\| + \cdots + \frac{1}{2^n}\|A(J_n)\| + \frac{1}{2^n}\|A(J_{n+1})\|.$$

Consider $E[\|A\|]$ by its constant terms, coefficients of β_i's, etc. For $1 \leqslant i \leqslant n-1$, coefficient of $\beta_i = \frac{1}{2^i}(2^{i-1}) + \frac{1}{2^{i+1}}(2^{i-1} - 2^i) + \cdots + \frac{1}{2^n}(2^{i-1} - 2^i) + \frac{1}{2^n}(2^{i-1} - 2^i) = 0$, and coefficient of $\beta_n = \frac{1}{2^n}(2^{n-1}) + \frac{1}{2^n}(2^{n-1} - 2^n) = 0$. Thus $E[\|A\|]$ only depends on the constant terms, and

$$E[\|A\|] = \sum_{i=1}^{n} \frac{1}{2^i}(2 + \cdots + 2^i)M + \frac{1}{2^n}(2 + \cdots + 2^n + 2^n)M$$
$$= \sum_{i=1}^{n} \frac{2(2^i-1)}{2^i} M + \frac{2(2^n-1)+2^n}{2^n} M$$
$$= \sum_{i=1}^{n} \left(2 - \frac{2}{2^i}\right) M + \left(3 - \frac{2}{2^n}\right) M$$
$$= (2n+1)M.$$

Hence

$$\frac{E[\|\mathrm{OPT}\|]}{E[\|A\|]} = \frac{5n/2+1}{2n+1} \to \frac{5}{4}$$

as n is very large. Thus no randomized algorithms have competitive ratio better than $\frac{5}{4}$. □

5. Resource augmentation

How much extra resources is required to get 1-competitive algorithms? In this section we give an algorithm that requires roughly a factor of $1.88 \lg \alpha$ 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 $4\lceil \lg \alpha \rceil$ upper bound for the traditional model [10,8].

5.1. Earliest deadline first

We first consider the earliest deadline first (EDF) algorithm, which always schedules the job with the earliest deadline, with a speed-s processor (one which has speed s times that of a normal processor).

Lemma 6. *EDF with a speed-s processor is α/s-competitive.*

Proof. We use a charging scheme almost identical to that stated in Section 2. The only difference is that, when $done_{\mathrm{OPT}}(q,t) < done_{\mathrm{EDF}}(q,t)$ and $done_{\mathrm{OPT}}(q,t) = done_{\mathrm{EDF}}(q,u)$ for $u < t$ (i.e., charge from t to u), the charges made to time u is $s \cdot w(q)$, i.e., s times 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 $s \cdot w(q_1)$ every unit time since it is running at speed-s. Consider the charges to time t, it consists of a $w(q_0)$ charge from time t and/or a $s \cdot w(q_1)$ charge from a later time.

Case 1: q_0 does not charge to t. Charging ratio $c \leqslant (s \cdot w(q_1))/(s \cdot w(q_1)) = 1$.

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 $[1,\alpha]$, then $w(q_0) \leqslant \alpha$, $w(q_1) \geqslant 1$. If there are no other charges (from later times in OPT), then $c \leqslant w(q_0)/(s \cdot w(q_1)) \leqslant \alpha/s$. Suppose q_1 charges from a later time $u > t$ in OPT. Since OPT is canonical, $d(q_0) \leqslant d(q_1)$, 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 $c \leqslant (s \cdot w(q_1))/(s \cdot w(q_1)) = 1$ since q_0 cannot charge to t at two different times. □

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 s processors P1, ..., Ps schedule the first, ..., sth earliest-deadline jobs, respectively. (If there are less than s active jobs then some processors idle.)

Lemma 7. *EDF with s speed-1 processors is α/s-competitive.*

Proof. For any time t, let q_0 be the job running in OPT, $q_1, \ldots, q_s$ be the jobs running in P1, ..., Ps, respectively. We again use the same charging scheme in Section 2. Consider the charges to time t, which consists of $w(q_0)$ from t, and/or $w(q_1), \ldots, w(q_s)$ from later times.

Case 1: q_0 does not charge to t. Then $c \leqslant (w(q_1) + \cdots + w(q_s))/(w(q_1) + \cdots + w(q_s)) = 1$.

Case 2: q_0 charges to t, and all $q_1, \ldots, q_s$ are not null. Suppose job weights are normalized so that $w(q_0) \leqslant \alpha$, $w(q_1), \ldots, w(q_s) \geqslant 1$.

Case 2.1: $q_0 \neq$ any one of q_i's. If q_i charges to t for some i, then since $d(q_0) \leqslant d(q_i)$ (OPT is canonical), Pi should schedule q_0 instead. So none of q_i can charge to t. Hence $c \leqslant w(q_0)/(w(q_1) + \cdots + w(q_s)) \leqslant \alpha/s$.

Case 2.2: $q_0 = q_i$ for some i. q_i cannot charge to t from a later time in OPT, since $done_{OPT}(q_i, t) > done_{EDF}(q_i, t)$ and thus $done_{OPT}(q_i, u) > done_{EDF}(q_i, t)$ for all later times u. Thus the set of OPT charges is a subset of the set of EDF jobs, so $c \leqslant 1$.

Case 3: q_0 charges to t, $q_1..q_i$ are not null, but $q_{i+1}..q_s$ are null, for some $i \geqslant 1$. (q_1 cannot be null, since q_0 is unfinished.) If q_0 is not one of $q_1..q_i$, then q_{i+1} cannot be null since q_0 is unfinished. Thus q_0 is one of $q_1..q_i$, say q_j, in which case q_j cannot charge to t from a later time in OPT. So the set of OPT charges is a subset of the set of EDF jobs, and hence $c \leqslant 1$. □

5.2. *Grouped-EDF*

Lemmas 6 and 7 implies that 1-competitiveness can be achieved by using a speed-α processor or $\lceil \alpha \rceil$ speed-1 processor. In fact this is the best EDF can do. Consider using a speed-s processor with $s < \alpha$ (assuming α is an integer). Let $s = \alpha - \delta, \delta > 0$ and $0 < \varepsilon < \delta/(\alpha - 1 - \delta)$. Consider the instance consisting of a job $(0, \alpha + \varepsilon, \alpha + \varepsilon, \alpha)$ and α copies of $(0, \alpha, \alpha, 1)$. OPT gets a value of $\alpha(\alpha + \varepsilon)$ by executing the heaviest job. Speed-s EDF gets a value of $\alpha s + \alpha s \varepsilon$. It is easy to verify that $\alpha s + \alpha s \varepsilon < \alpha(\alpha + \varepsilon)$.

However, we can do better: the following algorithm *Grouped-EDF* partitions the weight ranges $[1..\alpha]$ into $\lceil \log_\lambda \alpha \rceil$ classes, each having weights in the interval $[1, \lambda), [\lambda, \lambda^2), \ldots, [\lambda^{\lfloor \log_\lambda \alpha \rfloor - 1}, \lambda^{\lfloor \log_\lambda \alpha \rfloor}), [\lambda^{\lfloor \log_\lambda \alpha \rfloor}, \alpha]$. Jobs in each class have importance ratio at most λ. The algorithm assigns λ speed-1 processors to process jobs in each class using EDF. The total number of processors used, $\lambda \lceil \log_\lambda \alpha \rceil$, is minimum when $\lambda = e$ ($\lambda = 3$ if λ is restricted to be an integer).

Theorem 8. *Grouped-EDF is* 1*-competitive using* $3\lceil \log_3 \alpha \rceil$ *speed-*1 *processors.*

Proof. We use Grouped-EDF with $\lambda = 3$. Let OPT_i and EDF_i be the 1-processor optimal and λ-processor EDF schedules for the sub-instance consisting of only the ith class jobs, respectively. By Lemma 7, $\|OPT_i\| \leqslant \|EDF_i\|$ for all i. We also have $\|OPT\| \leqslant \sum \|OPT_i\|$ 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 $\|OPT\| \leqslant \sum \|OPT_i\| \leqslant \sum \|EDF_i\| = \|\text{Grouped-EDF}\|$. □

Alternatively, Grouped-EDF can assign a speed-λ processor to each class. By Lemma 6 and timesharing to simulate multiple processors, we have:

Theorem 9. *Grouped-EDF is* 1*-competitive with a speed-*$(e\lceil \ln \alpha \rceil)$ *processor, or with* $\lceil \ln \alpha \rceil$ *speed-e 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 10. *The speed-s version of MIX is* $1/(1-e^{-s})$*-competitive against a speed-*1 *offline optimal algorithm.*

A small amount of additional processing power can give very good competitiveness results, irrespective of the value of α. For example, with $s = 2$ we have $c = 1.16$, with $s = 3, c = 1.05$, and with $s = 5, c = 1.00678$, i.e. just 0.68% fewer than the optimal value.

6. Conclusion

In this paper we consider an 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, F. Wang, On the competitiveness of on-line real-time task scheduling, Real-Time Systems 4 (1992) 125–144.
[2] A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press, New York, 1998.
[3] E.-C. Chang, C. Yap, Competitive online scheduling with level of service, in: Proc. seventh Internat. Computing and Combinatorics Conf., Lecture Notes in Computer Science, Vol. 2108,2001, pp. 453–462.
[4] F.Y.L. Chin, S.P.Y. Fung, Online scheduling with partial job values and bounded importance ratio, in: Proc. Internat. Computer Symp.,2002, pp. 787–794.
[5] F.Y.L. Chin, S.P.Y. Fung, Online scheduling with partial job values: does timesharing or randomization help?, Algorithmica 37 (3) (2003) 149–164.
[6] M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichý, N. Vakhania, Preemptive scheduling in overloaded systems, J. Comput. System Sci. 67 (1) (2003) 183–197.
[7] B. Kalyanasunaram, K. Pruhs, Speed is as powerful as clairvoyance, J. ACM 47 (4) (2000) 617–643.
[8] C.-Y. Koo, T.-W. Lam, T.-W. Ngan, K.-K. To, Extra processors versus future information in optimal deadline scheduling, in: Proc. 15th ACM Symp. on Parallel Algorithms and Architectures,2002, pp. 133–142.
[9] G. Koren, D. Shasha, D^{over}: an optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems, SIAM J. Comput. 24 (1995) 318–339.
[10] T.-W. Lam, K.-K. To, Performance guarantee for online deadline scheduling in the presence of overload, in: Proc. 12th ACM-SIAM Symp. on Discrete Algorithms,2001, pp. 755–764.
[11] J.W.S. Liu, K.-J. Lin, W.-K. Shih, A.C. shi Yu, J.-Y. Chung, W. Zhao, Algorithms for scheduling imprecise computations, IEEE Comput. 24 (5) (1991) 58–68.
[12] C.A. Philips, C. Stein, E. Torng, J. Wein, Optimal time-critical scheduling via resource augmentation, in: Proc. 29th ACM Symp. on Theory of Computing,1997, pp. 140–149.
[13] D.D. Sleator, R.E. Tarjan, Amortized efficiency of list update and paging rules, Commun. ACM 28 (2) (1985) 202–208.
[14] A.C.-C. Yao, Probabilistic computations: toward a unified measure of complexity, in: Proc. 18th IEEE Symp. on Foundations of Computer Science,1977, pp. 222–227.