Laxity Helps in Broadcast Scheduling*

Stanley P.Y. Fung[1], Francis Y.L. Chin[1], and Chung Keung Poon[2]

[1] Department of Computer Science, The University of Hong Kong, Hong Kong
{pyfung, chin}@cs.hku.hk
[2] Department of Computer Science, City University of Hong Kong, Hong Kong
ckpoon@cs.cityu.edu.hk

Abstract. We study the effect of laxity, or slack time, on the online scheduling of broadcasts with deadlines. The laxity of a request is defined to be the ratio between its span (difference between release time and deadline) and its processing time. All requests have a minimum guaranteed laxity. We give different algorithms and lower bounds on the competitive ratio for different ranges of values of laxity, which not only represents a tradeoff between the laxity and the competitive ratio of the system, but also bridges between interval scheduling and job scheduling techniques and results. We also give an improved algorithm for general instances in the case when requests can have different processing times.

1 Introduction

The application of broadcasting in networks has been receiving much attention recently. Broadcasting has an advantage over point-to-point communication in that it can satisfy the requests of different users, who are requesting the same piece of information, simultaneously by a single broadcast. The advantage is more clearly seen when most of the requests are asking for common information like popular movies and weather information. Broadcasting is of even greater importance with the growing popularity of wireless and satellite networks which are inherently broadcasting in nature. There are some commercial systems that make use of broadcasting technology. For example, in the DirecPC system [1], clients make requests over phone lines and the server broadcasts the data via a satellite. In this paper, we study algorithms for the online scheduling of broadcasts with deadlines.

The Model. In the literature on broadcast scheduling, there are many different research work based on different assumptions of the network model. Here, we focus on the model in which the server holds a number of pages while requests come in and ask for pages (pull-based model). At any time, the server is allowed to broadcast only one page and all the requests asking for the same page can be satisfied simultaneously. Note that the server is not allowed to break down a

* The work described in this paper was fully supported by two grants from the Research Grants Council of the Hong Kong SAR, China [CityU 1198/03E and HKU 7142/03E].

M. Coppo et al. (Eds.): ICTCS 2005, LNCS 3701, pp. 251–264, 2005.

page into different parts and send them over the network simultaneously. Also, we assume the receiver does not have any buffer to cache part of a page previously broadcasted by the server. This is an online problem, meaning that the server does not know the requests of the clients until they arrive, and the server have to determine which page to broadcast without knowing future requests.

Past Work. Broadcast scheduling was first studied for the case where requests do not have deadlines, and the objective is to minimize the maximum or the average flow time (also called response time, the time between arrival and completion of requests). Broadcast scheduling with deadlines is first studied in [11] and [12]. Each request is associated with a deadline, and the objective is to maximize the total profit of satisfied requests (completed before their deadlines). The two papers, however, considered different models in relation to how broadcasts can be preempted.

The preemptivity of online scheduling in general (and broadcasting in particular) can be classified into three different models. (For example see [8].) In the *nonpreemptive* model, a job that gets started must run to completion without interruption. In the *preemption-resume* model, jobs can be preempted and later resumed at the last point of execution. In the *preemption-restart* model, a job can be preempted, but its partial progress will be lost; hence, it must be restarted from the beginning if it is to be processed again later. In this case preemptions can also be called *abortions*. Note that the nonpreemptive and preemption-restart models are identical in the offline case.

While the preemption-resume and nonpreemptive models have been widely studied, there seems to be comparatively few results for the online scheduling of jobs in the preemption-restart model. Hoogeveen et al. [8] considered the case in which jobs have no weights (i.e. the objective is to maximize the utilization of processor), and Chrobak et al. [4] considered the case where in addition all jobs have equal length.

These three models have also been considered in online broadcast scheduling. Kalyanasundaram and Velauthapillai [11] considered the preemption-resume model, and Kim and Chwa [12] considered the preemption-restart model. The nonpreemptive broadcast model is also considered in a different context called *batching with incompatible job families* [9]. We consider the preemption-restart broadcasting model in this paper.

We distinguish between the case where all pages have the same length (the *unit-length* case), and the case where pages can have different lengths (the *variable-length* case). For the unit-length case, a 4.56-competitive algorithm is given in [14], while a lower bound of 4 follows from an interval scheduling problem [13]. For the variable-length case, there is a $(e\Delta + e + 1)$-competitive algorithm [14] where Δ is the ratio of maximum to minimum page lengths and $e \approx 2.718$. There is also a lower bound of $\sqrt{\Delta}$ [2].

Laxity. The power of broadcasting lies in its ability to combine multiple requests and serve them together by a single broadcast. For this to be effective, requests should have a certain amount of *laxity*, where the laxity of a request is defined to be the ratio of its span (the length of time interval between its deadline

and release time) and its processing time (the length of the page it requests). This allows the system to delay processing a request and to serve it together with some other requests for the same page that arrives later. If a request has no laxity, it must be scheduled immediately or else it is lost, and hence the power of broadcasting is not utilized. In fact, in all of the above-mentioned lower bounds, all requests are tight (with no laxity). To actually analyze the effect of broadcasting, we assume all requests have a minimum laxity $\alpha > 1$. Intuitively, with larger laxity, the system will be able to schedule more requests together. However a large laxity may be unsatisfactory to users. In this paper we analyze the relation between the laxity of requests and the total profit obtained.

The issue of laxity has been considered in the unicast (i.e. non-broadcast) context. It is also called *slack*, *patience* [7] or *stretch factor* [5]. Online job (unicast) scheduling with laxity is widely studied, for example by Kalyanasunaram and Pruhs [10] in the preemption-resume model, and by Goldman et al [6] and Goldwasser [7] in the nonpreemptive model. The assumption of minimum laxity has also been used in preemption-resume broadcast scheduling with deadlines [11], and in nonpreemptive broadcast (or batching) problems [9].

Interestingly, the effect of laxity also allows this problem to bridge between interval scheduling and job scheduling. In most previous results on broadcast scheduling, the techniques used are similar to interval scheduling in which the most important concern is whether a broadcast should be preempted. When a broadcast is completed without being preempted, for example, the next broadcast is usually the one with the most pending requests. However our results indicate that as laxity increases, the problem has more job scheduling flavor, which involves selection of jobs based on both weights and deadlines. We will bring techniques from job scheduling into this problem.

Our results. In this paper we consider the effect of laxity in the preemption-restart broadcasting model. In Section 3 we first give an algorithm when the laxity is smaller than 2. It adapts an abortion criteria which varies with laxity and how much the current broadcast has been completed. It achieves a tradeoff in the competitive ratios, with smaller competitive ratios for larger laxity. Next we give a simple 2.618-competitive algorithm for the case where the laxity is at least 2. It does not use preemption and thus also applies to the nonpreemptive case. Unlike previous algorithms which only consider abortion conditions, our algorithm also considers how to select broadcasts after another broadcast is completed. In Section 4 we give lower bounds on the competitive ratio for different ranges of laxity α: for example, $8/3$ for $1 < \alpha < 4/3$, $12/5$ for $4/3 < \alpha < 1.4$, 2 for $1.4 < \alpha < 3/2$, and $\max(1 + 1/\lceil\alpha\rceil, 5/4)$ for $\alpha \geq 2$. For $\alpha < 2$ we extend the technique of Woeginger's lower bound of interval scheduling, while for $\alpha \geq 2$ we make use of job scheduling results. The lower bound does not approach 1 even with arbitrarily large laxity. Finally in Section 5 we consider the variable page length case, giving a $(\Delta + 2\sqrt{\Delta} + 2)$-competitive algorithm for general instances, which improves the previous bound of $e\Delta + e + 1$. This algorithm makes use of a simple observation and its analysis is significantly simpler than earlier algorithms. We also state a lower bound result with laxity.

2 Notations

There are a number of pages in a server, each having a possibly different length. Requests arrive to the server online, i.e. no information about the request is known until it arrives. Each request j has a release time $r(j)$, a deadline $d(j)$, the page $p(j)$ that it requests, and a weight $w(j)$. All $r(j), d(j)$ and $w(j)$ are real numbers. We define the processing time of a request, $l(j)$, to be the length of the page it requests. A request j fully completed before its deadline gives a profit of $w(j) \times l(j)$. For the unit length case we assume all pages have length 1, and hence weights and profits are equivalent. Define $\alpha = \min_j\{(d(j) - r(j))/l(j)\}$ to be the minimum laxity of all requests. We assume there is a pool that contains all pending requests. Requests that cannot be completed by their deadlines even if started immediately will be automatically removed from the pool.

A broadcast of a page J serves all pending requests of J simultaneously. We consider the preemption-restart model, in which a broadcast can be preempted (aborted) at any time, but must be restarted from the beginning if it is to be broadcast again. Let $|J| = \sum_{p(j)=J} w(j) \times l(j)$ denote the profit of a page J, i.e. the total profit of all pending requests j for the page J at a particular time. This is the profit obtained of broadcasting J. Our objective is to maximize the total profit of requests completed before their deadlines. We sometimes abuse the terminology and use 'page', 'broadcast (of a page)', and 'set of requests (for a page)' interchangeably. A schedule is specified by a sequence of broadcasts $J_1, J_2, \ldots$ where each broadcast J_i starts at time $s(J_i)$. If $s(J_i)+l(J_i) > s(J_{i+1})$, J_i is aborted by J_{i+1}, otherwise it is completed.

We measure the performance of online algorithms by their competitive ratios. Let OPT denote the offline optimal algorithm. An online algorithm A is *R-competitive* if, for any instance I, the profit $A(I)$ obtained by A is at least $1/R$ times the profit $OPT(I)$ obtained by OPT.

3 Unit Length Pages: Upper Bounds

3.1 Minimum Laxity $\alpha < 2$

We first consider the unit-length case where the minimum laxity is less than 2. We only consider those α that are rational. Let $\alpha = 1 + p/m$ for integers p, m where $p < m$ and m is the minimum possible. Intuitively, our algorithm uses an 'abortion ratio' which increases with time while a page is being broadcast.

Algorithm MultiLevel. Let $\beta > 1$ be the unique positive real root of $\beta - \beta^{1/m} - 1 = 0$. Suppose a page J starts being broadcast at time t. Then for $i = 1, 2, \ldots, m$, a new request for page J' arriving at $[t + (i-1)/m, t + i/m)$ (together with pending requests for J' in the pool [1]) will abort J if and only if $|J'| \geq \beta^{i/m}|J|$.

[1] It is possible that $J' = J$; we also consider requests currently being served by broadcast J to be in the pool if they can still be completed before their deadlines if restarted.

When a broadcast completes, the page with the maximum profit (for all pending requests) will be broadcast next.

The following table lists the choice of β for different values of laxity and the competitive ratios. Note that some of these ratios are larger than the 4.56 upper bound for the case of arbitrary instances given in [14].

α	1.2	1.25	1.33	1.4	1.5	1.6	1.67	1.75	1.8
β	2.164	2.221	2.325	2.164	2.618	2.164	2.325	2.221	2.164
R	4.714	4.638	4.510	4.448	4.236	4.221	4.08	4.04	4.026

Theorem 1. *For laxity* $1 < \alpha < 2$ *where* $\alpha = 1 + p/m$*, MultiLevel is* $(\frac{\beta}{\beta-1} + \beta^{2-\alpha} + 1)$*-competitive, where* β *is the root of* $\beta - \beta^{1/m} - 1 = 0$*.*

Proof. Let M denote the schedule produced by MultiLevel. Divide M into a set of *basic subschedules*, where each basic subschedule consists of zero or more aborted broadcasts followed by one completed broadcast. We charge the profits obtained by OPT to the basic subschedules in M. If every basic subschedule with a completed broadcast J receives charges at most R times that of $|J|$, then the algorithm is R-competitive. The general idea of the proof is to consider, for a fixed basic subschedule, the maximum-profit OPT schedule that is 'consistent' with the basic subschedule. For example, OPT cannot serve requests with profit much higher than that being served by M at the same time, unless it is already completed in M, because M will not ignore this request if it is still pending.

Without loss of generality we can assume all broadcasts made by M are of one of the lengths (time units) $1/m, 2/m, \ldots, 1$, since otherwise lengthening them to the closest length listed above will not decrease the profit obtained by OPT. Since the last broadcast in a basic subschedule must be completed, it must be of length 1. Note that all broadcasts of OPT must be of unit length since they are completed. (We can assume OPT will not make aborted broadcasts.)

Let $r = \beta^{1/m}$. We first consider the case that all basic subschedules satisfy two assumptions: (1) All broadcasts except the last one are of length $1/m$. (2) For any two consecutive broadcasts J and J' in the basic subschedule, $|J| = |J'|/r$. We will remove these assumptions later. Consider a basic subschedule. The maximum-profit OPT subschedule corresponding to this basic subschedule occurs when the total number of broadcasts in the basic subschedule is a multiple of m plus 2, with the OPT broadcasts as in Fig. 1. (The reason of this will become clear later in the proof.) Hence, let the basic subschedule be $(J_0, J_1, \ldots, J_m, \ldots, J_{km}, J_{km+1})$ for any $k \geq 0$. For those requests that are satisfied in an OPT broadcast started at time t and which are completed by M before time t, we charge their profits to the basic subschedule in M where it is completed. Let O_i be the set of requests in a broadcast started by OPT within J_i that are not completed by M before $s(O_i)$, i.e., they are still pending in M.

Consider the last broadcast O_{km+1} made by OPT. It consists of two parts: O^1_{km+1} which are those requests arriving on or before $s(J_{km+1}) + 1 - p/m$, and O^2_{km+1} which are those requests arriving after this time. For O^1_{km+1} we have $|O^1_{km+1}| < r^{m-p}|J_{km+1}|$ since otherwise they would abort J_{km+1}. For O^2_{km+1},

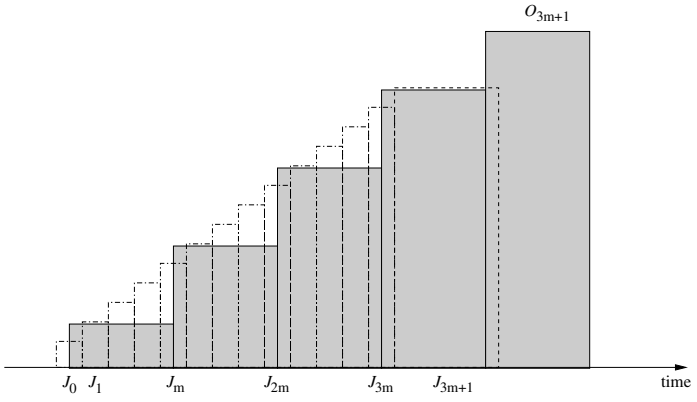

Fig. 1. A basic subschedule with $k = 3, m = 4$. Each rectangle represents an aborted or completed broadcast, and the height of a rectangle represents its profit. Empty rectangles are those of MultiLevel, shaded rectangles are those of OPT.

they must still be pending in M after J_{km+1} completes due to their laxity, and hence the broadcast after J_{km+1} must have profit at least as large as $|O^2_{km+1}|$. We charge their profits to the next basic subschedule. Similarly, this basic subschedule may also receive a charge of profit at most $|J_0|$ from the previous basic subschedule. For O_i where $i = 0, m, 2m, \ldots, km$, we have $|O_i| < r|J_i|$. Finally, requests in J_{km+1} may be satisfied in OPT in some future time, and their profits, which are at most $|J_{km+1}|$ in total, are also charged to this basic subschedule. Without loss of generality we normalize the profits so that $|J_{km+1}| = 1$. Then the total OPT profits charged to this basic subschedule is at most

$$
\begin{aligned}
&|J_0| + r(|J_0| + |J_m| + |J_{2m}| + \cdots + |J_{km}|) + r^{m-p}|J_{km+1}| + |J_{km+1}| \\
&\leq 1/r^{km+1} + r(1/r^{km+1} + \cdots + 1/r) + r^{m-p} + 1 \\
&= \frac{1}{r^{km+1}} + \frac{1 - 1/r^{km+m}}{1 - 1/r^m} + r^{m-p} + 1 \\
&= \frac{1}{r^{km+1}} - \frac{1}{r^{km+m}(1 - 1/r^m)} + \frac{1}{1 - 1/r^m} + r^{m-p} + 1 \\
&= \frac{1}{\beta^k}\left(\frac{1}{\beta^{1/m}} - \frac{1}{\beta - 1}\right) + \frac{\beta}{\beta - 1} + \beta^{1-p/m} + 1.
\end{aligned}
$$

By choosing $\beta^{1/m} = \beta - 1$, the first term in the above expression is zero, i.e., the expression is invariant of the value of k. In this case the competitive ratio is

$$R \leq \frac{\beta}{\beta - 1} + \beta^{1-p/m} + 1 = \frac{\beta}{\beta - 1} + \beta^{2-\alpha} + 1.$$

We now remove assumptions (1) and (2). For any basic subschedule that does not satisfy the assumptions, we apply the following transformations to all broadcasts except the last one. First, for any two consecutive broadcasts J, J' in

a basic subschedule, if ℓ is the length of J served in the schedule (not the length of the page), we increase the profit of J to $|J'|/\beta^\ell$ if it is not already so. The schedule remains valid (all abortion ratios are satisfied) and the profits of OPT will not decrease, since the maximum possible profits of OPT broadcasts at any time (w.r.t. the basic subschedule) will not be decreased. Next, a broadcast of length i/m and profit $|J|$ for $i > 1$ is transformed to i broadcasts of length $1/m$, with profits $|J|, r|J|, r^2|J|, \ldots, r^{i-1}|J|$. Again the schedule remains valid, and the maximum possible OPT profits are not decreased. The transformed schedule satisfies the two assumptions, and the transformation can only increase the competitive ratio. □

3.2 Minimum Laxity $\alpha \geq 2$

Next we consider the case when the laxity is at least 2, i.e. the span of a request is at least twice its length. In this case we use a simple algorithm that never aborts, but is more careful in selecting a page to broadcast after a broadcast completes:

Algorithm EH. When a broadcast completes at time t, set H to be the page with the maximum profit among pending requests, E to be the page with the maximum profit among those pending requests with deadlines before $t+2$. Note that although E is chosen based on those requests with early deadlines, it may also contain requests with late deadlines. Let E' denote the subset of requests for page E with deadlines before $t+2$. Let $\beta = (\sqrt{5}+1)/2$. If $|H| \geq \beta|E'|$, broadcast H, else broadcast E. A page being broadcast is never aborted.

Theorem 2. *For $\alpha \geq 2$, EH is $(\sqrt{5}+3)/2 \approx 2.618$-competitive.*

Proof. We charge the profits obtained by OPT to the broadcasts in EH. Consider a completed broadcast J in EH. Let O be the single page started being broadcast by OPT when EH is broadcasting J. For requests in O that are completed by EH in or before J, we charge their profits to those earlier broadcasts made by EH. Below we only consider requests in O that are still pending in EH. We separate O into two parts: O_1 which are those requests in O having deadlines before $s(J)+2$, and O_2 which are the remaining requests. We distinguish the following cases.

Case 1: $J = H \neq E$. For O_1, they must be released before $s(J)$ by the laxity assumption. We have $|O_1| \leq |E'|$ since E is chosen to be the highest-profit page among those requests with deadlines before $s(J)+2$. So $|O_1| \leq |E'| \leq |J|/\beta$ by the choice of EH. For O_2, since they remain pending in EH when J completes, we charge their profits to the next broadcast after J. Similarly, this J receives a charge C from the previous broadcast, where $|C| \leq |J|$. Also, requests in J may be satisfied later by OPT, which we also charge their profits to this J in EH. Thus the total profits charged to J is at most $|C|+|O_1|+|J| \leq |J|+|J|/\beta+|J| = (2+1/\beta)|J|$.

Case 2: $J = E$. Similar to Case 1 we have $|O_1| \leq |E'|$ and O_2 is charged to the next broadcast. J also receives a charge C from the previous broadcast,

where $|C| \leq |H| \leq \beta|E'| \leq \beta|E|$. Requests in E' cannot be started by OPT after $s(J)+1$ since their deadlines is earlier than $s(J)+2$, so J may receive a 'future' charge only from requests in $E - E'$. Thus the total charge to J is at most $|C| + |O_1| + |E - E'| \leq \beta|E| + |E'| + |E - E'| = (1+\beta)|J|$.

By setting $\beta = (\sqrt{5}+1)/2$, the total charge to a broadcast J in any case is at most $\frac{\sqrt{5}+3}{2}|J|$. Hence the algorithm is $(\sqrt{5}+3)/2$-competitive. □

Remark: Note that EH is a non-preemptive algorithm. Since OPT does not use abortions, EH also applies with the same competitive ratio in the nonpreemptive case. This nonpreemptive broadcast problem is also studied in [9] where a 3-competitive algorithm is given.

4 Unit Length Pages: Lower Bounds

4.1 Minimum Laxity $\alpha < 3/2$

We first describe our lower bounds for the unit-length case when the minimum laxity α is smaller than 3/2. Let $R = 4 - \epsilon$ for some arbitrarily small $\epsilon > 0$. Define $lax = \alpha - 1$. Choose a small positive number $d \ll 1 - lax$, and another small positive δ. Define $\delta_i = \delta/2^i$. These are small constants we need to use later. We divide the proof into two steps.

(1) Description of instance construction. The proof is based on a modification of Woeginger's lower bound construction for interval scheduling [13]. In the following each request is asking for a different page. For $0 < v \leq w$ and $d, \delta > 0$, define $\mathrm{SET}(v, w, d, \delta)$ to be a set of requests $\{j_1, j_2, \ldots, j_q\}$ with the following properties:

- The weights $w(j_i)$ of the requests fulfill $w(j_1) = v$, $w(j_q) = w$ and $w(j_i) < w(j_{i+1}) \leq w(j_i) + \delta$ for $1 \leq i \leq q-1$.
- The release time $r(j_i)$ and deadline $d(j_i)$ of the requests fulfill $0 = r(j_1) < r(j_2) < \cdots < r(j_q) < d$, $1 + lax = d(j_1) < d(j_2) < \cdots < d(j_q) < 1 + lax + d$.
- All requests have length 1 and laxity α.

Since $d(j_q) - r(j_1) < 1 + lax + d < 2$, any algorithm can complete at most one request in each set.

Fix an online algorithm A. At $t = 0$, $\mathrm{SET}(v_0, w_0, d_0, \delta_0) = \mathrm{SET}(1, R, d, \delta)$ arrives. Define $LST_0 = lax + d$; this is the latest time that A must fix its decision and start broadcasting a page, or else it will fail to meet the deadline of any request. Let $t_0 \leq LST_0$ be the earliest time such that during the time interval $[t_0, LST_0]$, A is broadcasting the same page. That means A fixed its decision at time t_0. Without loss of generality we can assume that A will not switch to broadcast other requests in the same set after this time. Let b_0 be the page being broadcast by A at time LST_0. If $w(b_0) = 1$, no more request arrives. Otherwise, a shifted copy of $\mathrm{SET}(v_1, w_1, d_1, \delta_1)$ arrives, where $v_1 = w(b_0)$, $w_1 = Rv_1 - v_1$, $d_1 \ll d$ is smaller than the time between $d(b_0)$ and $d(a_0)$, and a_0 is the request immediately preceding b_0 in the set. The first and last request in this set arrive

at times r_1 and r'_1 respectively, where $t_0 + 1 - lax - d_1 < r_1 < r'_1 < t_0 + 1 - lax$. Define LST_1 to be the latest time when A must fix its decision on which request in the second set to broadcast. It is given by $LST_1 = r'_1 + lax$.

The numbers are defined so that they have the following two properties:

- $r_1 > LST_0$: this is because $r_1 > t_0 + 1 - lax - d_1 \geq 1 - lax - d_1$, and $LST_0 = lax + d$, as long as $d_1 < 1 - 2lax - d$ the property holds. Since $lax < 1/2$, d and d_1 can always be chosen small enough.
- $LST_1 < t_0 + 1$: this is because $LST_1 = r'_1 + lax < (t_0 + 1 - lax) + lax$.

The first property ensures that the adversary can decide the weights of the requests in the new set after A fixed its choice of broadcast. The second property ensures that A cannot serve a request in both sets. Hence, if A completes the broadcast of page b_0, it cannot serve any more requests, and no more requests are released. Otherwise, A aborts b_0 and starts a new page in the new set. Then the same process repeats.

In general, suppose A is broadcasting a page in the $(i+1)^{th}$ SET$(v_i, w_i, d_i, \delta_i)$. Let LST_i be the latest time A must fix its decision to broadcast a page in this set; it is given by $LST_i = r'_i + lax$. Let t_i be the earliest time such that in $[t_i, LST_i]$, A is broadcasting the same page. Call this page b_i. If $w(b_i) = v_i$, no more requests are released. Otherwise, a new $(i+2)^{th}$ SET$(v_{i+1}, w_{i+1}, d_{i+1}, \delta_{i+1})$ is released, where $v_{i+1} = w(b_i)$, $w_{i+1} = \max(Rv_{i+1} - \sum_{j=1}^{i+1} v_j, v_i)$, d_{i+1} is the time between $d(b_i)$ and $d(a_i)$, and a_i is the request immediately preceding b_i.The first and last request in this new set arrive at times r_{i+1} and r'_{i+1} respectively, where $t_i + 1 - lax - d_{i+1} < r_{i+1} < r'_{i+1} < t_i + 1 - lax$ (see Fig. 2).

Again we have the following two properties:

- $r_{i+1} > LST_i$: since $r_{i+1} > t_i + 1 - lax - d_{i+1} \geq r_i + 1 - lax - d_{i+1}$, and $LST_i \leq r_i + d_i + lax$, as long as $d_{i+1} < 1 - 2lax - d_i$ this property holds.
- $LST_{i+1} < t_i + 1$: this is because $LST_{i+1} < (t_i + 1 - lax) + lax = t_i + 1$.

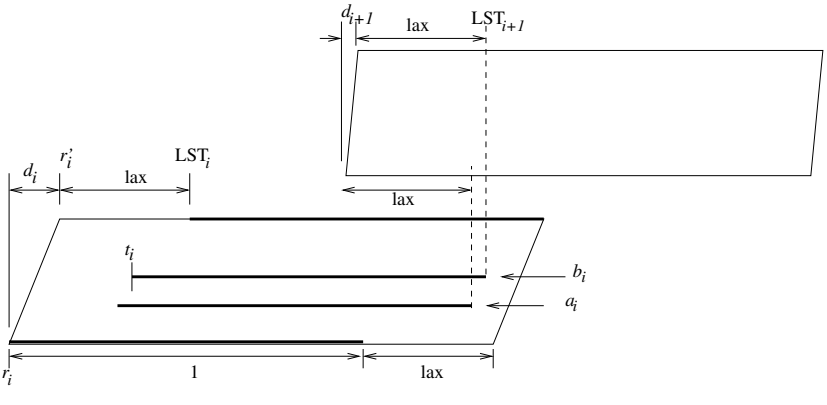

Fig. 2. SET$(v_i, w_i, d_i, \delta_i)$ and SET$(v_{i+1}, w_{i+1}, d_{i+1}, \delta_{i+1})$

A either completes b_i and no more requests are released, or aborts and broadcasts a page in the new set, and then the same process repeats. In the beginning, we have $w_i = Rv_i - \sum_{j=1}^{i} v_j > v_i$, and $v_i < v_{i+1} \leq w_i$, so that the v_i's form an increasing sequence satisfying the recurrence $v_{i+1} \leq Rv_i - \sum_{j=1}^{i} v_j$, and it is shown in [13] that this sequence cannot be an infinite increasing sequence for $R < 4$. Therefore, eventually we have $w_i = v_i > Rv_i - \sum_{j=1}^{i} v_i$ after a finite number of steps. At this final step the set contains only one request with profit v_i. No matter how A chooses, it obtains a profit of v_i. We conclude that A can only serve one request in total.

(2) Profits by the optimal offline algorithm. We now consider the OPT schedule for the instance. Ideally, we want OPT to schedule a request in each set (as in the original construction in [13]), but it may not be possible due to the earlier arrival of the sets (in the original construction a new set only arrives just before the deadlines of the previous set). Recall b_i is the request chosen by A in the $(i+1)$-th set, and a_i is the request that just precedes b_i. Suppose A completes a broadcast of page b_{i-1} in the i-th set, of profit v_i, while attempted to broadcast but aborted $b_0, b_1, \ldots, b_{i-2}$ in all earlier sets. OPT broadcasts $a_0, a_1, \ldots, a_{i-1}$ together with w_i, assuming that is feasible. (If A is forced to serve the single request in the last set the analysis is similar.) Then similar to [13] the total profit by OPT is at least $(4 - 2\epsilon)v_i$, so OPT obtains a profit arbitrarily close to 4 times what A obtains. Below we consider how OPT schedules a subset of these requests.

We resort to figures and defer the formal proof to the full paper. For any two consecutive sets, OPT can serve a_i in the first set together with any request in second set. However, for any three consecutive sets, OPT can only choose requests from two sets (see Fig. 3).

If $\alpha < 4/3$, we can partition the sets into groups of three, and choose the heavier two requests from each group without interfering other groups. (see Fig. 3). Thus OPT obtains a $2/3$ of the profits of those requests.

For $4/3 \leq \alpha < 3/2$, this is not always possible, but we can divide the sets into larger groups and apply the same idea. For example, when $\alpha = 1.4$, we divide the sets into groups of five, and it is always possible to serve requests in the second, fourth and fifth set (see Fig. 3). Although we are not selecting

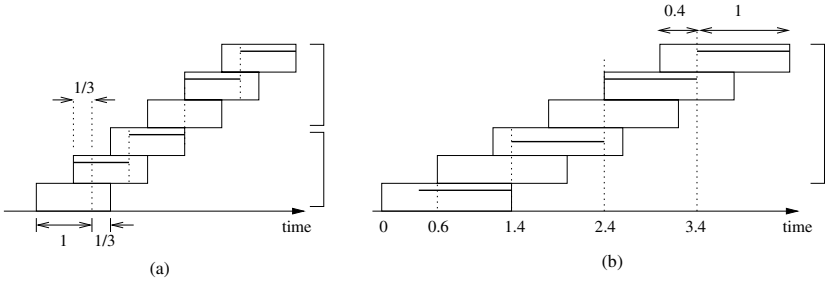

Fig. 3. Each rectangle represents a set. (a) when $lax < 1/3$, choose two sets out of every three. (b) when $lax < 0.4$, choose three sets out of every five.

the heaviest three requests, we can still guarantee a 3/5 fraction of profits, by the following simple observation: for any $0 \le v_1 \le v_2 \le v_3 \le v_4 \le v_5$, we have $\frac{v_2+v_4+v_5}{v_1+v_2+v_3+v_4+v_5} \ge \frac{3}{5}$. Hence, we have a lower bound of $4 \times 3/5 = 2.4$. Similarly we can obtain a lower bound of 16/7 when $\alpha < 10/7$, and a lower bound of 2 when $\alpha < 3/2$. In general we have:

Theorem 3. *For any integer $k \ge 1$ and $\alpha < 1 + k/(2k+1) < 3/2$, no deterministic algorithm is better than $4(k+1)/(2k+1)$-competitive.*

4.2 Minimum Laxity $\alpha \ge 2$

Next we consider the case where the minimum laxity is at least 2.

Theorem 4. *For $\alpha \ge 2$, no deterministic algorithm is better than $(1 + 1/\lceil\alpha\rceil)$-competitive.*

Proof. We only consider the case where α is an integer; otherwise we round it up to the nearest integer to obtain the result. All requests in this proof have weight 1 and laxity exactly α. At time 0, α requests each asking for a different page arrive. Without loss of generality assume the online algorithm A broadcasts a page J. Just before it finishes (at time $1 - \epsilon$) another request for J arrives. No matter A aborts the current broadcast or not, it can satisfy at most α requests. OPT can broadcast J at time 1 and some other page at time 0, satisfying all $\alpha + 1$ requests. □

In [3] we considered an s-uniform unit job scheduling problem in which each job arrives at integer time, is of unit length, has span exactly s (an integer) and no broadcasting is allowed. We gave a lower bound of $\frac{5}{4} - \Theta(\frac{1}{s})$ for the competitive ratio of this problem.[2] Since the case of broadcast with laxity at least α is a generalized case of unicast with laxity exactly s, where $s \ge \alpha$, which are s-uniform instances, the lower bound carries to the broadcast problem. (Since all time parameters are integers and jobs are of unit length, the ability of abortion in broadcasting needs not be used.) For s sufficiently large the lower bound is arbitrarily close to 5/4. We therefore have:

Theorem 5. *For any laxity α, no deterministic or randomized algorithm can be better than 5/4-competitive.*

The theorem implies that even with arbitrarily large laxity, the competitiveness cannot approach 1. This bound is stronger than that in Theorem 4 for sufficiently large α.

5 Variable Length Pages

Without laxity assumptions, there is a $(e\Delta + e + 1)$-competitive algorithm [14] for the variable-length case, where $\Delta = \lceil$maximum page length/minimum page

[2] The conference version does not contain this result.

length⌉, and $e \approx 2.718$ is the base of the natural logarithm. In this section we give an improved algorithm ACE (for 'Another Completes Earlier'), which makes use of a simple observation: if the broadcast of a page for a newer request has the same or larger profit and an earlier completion time than the page currently being broadcast, we should abort the current page in favour of the newer page.

Algorithm ACE. Let $\beta = 1 + \sqrt{\Delta}$. Let J be the page currently being broadcast. A new request for page J' (together with all pending requests for J' in the pool) will abort J if either one of the following holds: (1) $|J'| \geq \beta|J|$, or (2) $|J'| \geq |J|$ and the completion time of J' (if we start J' now) is earlier than the completion time of J (if we continue to broadcast J). When a broadcast completes, broadcast a page with the maximum profit among pending requests.

Theorem 6. *ACE is* $(\Delta + 2\sqrt{\Delta} + 2)$*-competitive.*

Proof. (Sketch) Without loss of generality assume the shortest page is of length 1 and the longest is of length Δ. Let A be the schedule produced by ACE. As before we divide the schedule into a set of basic subschedules, and we focus on a single basic subschedule. We can assume that all abortions in A are due to condition (1) of the algorithm, and further we assume that all broadcasts made in A are Δ units long (we omit the details in this version of the paper).

Consider a basic subschedule $(J_1, \ldots, J_k)$. We have $|J_i| \leq |J_k|/\beta^{k-i}$ since all abortions are due to condition (1). Consider the broadcasts started by OPT within the broadcast of J_i. There can be at most Δ such broadcasts. If the requests in these broadcasts are completed in A before they are started in OPT, we charge their profits to those earlier broadcasts in A. Now consider those requests that are not completed in A before. For any OPT broadcast O completed before J_i is completed or aborted, we have $|O| < |J_i|$ since otherwise they would abort J_i in A. For an OPT broadcast O that is completed after J_i is completed or aborted, we have $|O| < \beta|J_i|$ or else it would also abort $|J_i|$ in A. In this case no more OPT broadcasts after this O are charged to this J_i.

Therefore, in the worst case, OPT schedules at most $(\Delta - 1)$ length-1 broadcasts each of profit at most $|J_i|$, followed by a length-1 broadcast of profit at most $\beta|J_i|$. Therefore, the total OPT profit corresponding to J_i is at most $(\Delta - 1 + \beta)|J_i|$. Summing over all J_i's, and considering that J_k may be served in OPT later and will be charged to this basic subschedule, we have that the total OPT profits charged to this basic subschedule is at most

$$\sum_{i=1}^{k}(\Delta-1+\beta)|J_i|+|J_k| \leq (\Delta-1+\beta)\sum_{i=1}^{k}\frac{|J_k|}{\beta^{k-i}}+|J_k| < \left(\frac{\beta(\Delta-1+\beta)}{\beta-1}+1\right)|J_k|.$$

Hence the competitive ratio is $\frac{\beta(\Delta-1+\beta)}{\beta-1} + 1$. By setting $\beta = 1 + \sqrt{\Delta}$, this ratio is minimized and is equal to $\Delta + 2\sqrt{\Delta} + 2$. □

In the variable-length case, without assumptions on laxity, there is a lower bound of $\sqrt{\Delta}$ on the competitive ratio. Here we note that laxity may not help much in this case.

Note that laxity is related to resource augmentation using a faster processor, as discussed in [10]. If the online algorithm has a speed-s processor, then the laxity of all jobs become at least s (for the online algorithm). Thus any algorithm for laxity-s instances can be applied. Since OPT does not have this laxity advantage, the competitive ratio in this case will be even smaller (or the same) compared with the case where both the offline and online algorithm receive jobs with laxity (which is the standard case). Therefore the existence of an R-competitive online algorithm for laxity-s instances implies an s-speed R-competitive algorithm for general instances. On the other hand, a lower bound of R on s-speed online algorithm on general instances implies the same lower bound on laxity-s instances.

In [12] it was shown that using resource augmentation does not drastically improve the competitive ratio: no deterministic online algorithm with a constant speedup (can broadcast a constant number of pages more than the offline algorithm) can be constant competitive. In fact, the proof shows something stronger, that no deterministic online algorithm can be better than $O(\sqrt{\Delta})$-competitive with constant speedup.

Therefore, the lower bound for faster processor implies a lower bound of competitiveness with laxity:

Theorem 7. *For constant α, any deterministic online algorithm has competitive ratio $\Omega(\sqrt{\Delta})$.*

References

1. DirecPC Homepage, http://www.direcpc.com.
2. W.-T. Chan, T.-W. Lam, H.-F. Ting and P. W. H. Wong, New results on on-demand broadcasting with deadline via job scheduling with cancellation, in *Proc. 10th COCOON*, LNCS 3106, pages 210–218, 2004.
3. F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, J. Sgall and T. Tichý, Online competitive algorithms for maximizing weighted throughput of unit jobs, *to appear in Journal of Discrete Algorithms*, a preliminary version appeared in Proc. 21st STACS, 2004.
4. M. Chrobak, W. Jawor, J. Sgall and T. Tichý, Online scheduling of equal-length jobs: randomization and restarts help, in *Proc. 31st ICALP*, LNCS 3142, pages 358–370, 2004.
5. B. DasGupta and M. A. Palis, On-line real-time preemptive scheduling of jobs with deadlines on multiple machines, *Journal of Scheduling*, **4**:297–312, 2001.
6. S. A. Goldman, J. Parwatikar and S. Suri, Online scheduling with hard deadlines, *Journal of Algorithms*, **34**(2):370–389, 2000.
7. M. H. Goldwasser, Patience is a virtue: the effect of slack on competitiveness for admission control, *Journal of Scheduling*, **6**:183–211, 2003.
8. H. Hoogeveen, C. N. Potts and G. J. Woeginger, On-line scheduling on a single machine: maximizing the number of early jobs, *Operations Research Letters*, **27**(5):193–197, 2000.
9. R. Y. S. Hung and H. F. Ting, Online scheduling a batch processing system with incompatible job families, 2005, manuscript.

10. B. Kalyanasundaram and K. Pruhs, Speed is as powerful as clairvoyance, *Journal of the ACM*, **47**(4):617–643, 2000.
11. B. Kalyanasundaram and M. Velauthapillai, On-demand broadcasting under deadline, in *Proc. 11th ESA*, LNCS 2832, pages 313–324, 2003.
12. J.-H. Kim and K.-Y. Chwa, Scheduling broadcasts with deadlines, *Theoretical Computer Science*, **325**(3):479–488, 2004.
13. G. J. Woeginger, On-line scheduling of jobs with fixed start and end times, *Theoretical Computer Science*, **130**(1):5–16, 1994.
14. F. Zheng, S. P. Y. Fung, F. Y. L. Chin, C. K. Poon and Y. Xu, Improved on-line broadcast scheduling with deadlines, *Submitted for publication*.