Online Scheduling to Minimize the Maximum Delay Factor

Chandra Chekuri* Benjamin Moseley†

Abstract

In this paper two scheduling models are addressed. First is the standard model (unicast) where requests (or jobs) are independent. The other is the broadcast model where broadcasting a page can satisfy multiple outstanding requests for that page. We consider online scheduling of requests when they have deadlines. Unlike previous models, which mainly consider the objective of maximizing throughput while respecting deadlines, here we focus on scheduling all the given requests with the goal of minimizing the maximum *delay factor*. The delay factor of a schedule is defined to be the minimum $\alpha \geq 1$ such that each request i is completed by time $a_i + \alpha(d_i - a_i)$ where a_i is the arrival time of request i and d_i is its deadline. Delay factor generalizes the previously defined measure of maximum stretch which is based only the processing times of requests [9, 11].

We prove strong lower bounds on the achievable competitive ratios for delay factor scheduling even with unit-time requests. Motivated by this, we consider resource augmentation analysis [24] and prove the following positive results. For the unicast model we give algorithms that are $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive in both the single machine and multiple machine settings. In the broadcast model we give an algorithm for same-sized pages that is $(2+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive. For arbitrary page sizes we give an algorithm that is $(4+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive.

1 Introduction

Scheduling requests (or jobs[1]) that arrive online is a fundamental problem faced by many systems and consequently there is a vast literature on this topic. A variety of models and performance metrics are studied in order to capture the requirements of a system. In this work, we consider a recently suggested performance measure called *delay factor* [14, 10] when each request has an *arrival time* (also referred to as release time) and a *deadline*. We consider both the traditional setting where requests are independent, and also the more recent setting of broadcast scheduling when different requests may ask for the same page (or data) and can be simultaneously satisfied by a single transmission of the page. We first describe the traditional setting, which we refer to as the *unicast* setting, to illustrate the definitions and and then describe the extension to the *broadcast* setting.

We assume that requests arrive *online*. The arrival time a_i, the deadline d_i, and the processing time ℓ_i of a request J_i are known only when i arrives. We refer to the quantity $S_i = (d_i - a_i)$ as the *slack* of request i. There may be a single machine or m identical machines available to process the requests. Consider an online scheduling algorithm A. Let f_i denote the completion time or finish time of J_i under A. Then the delay factor of A on a given sequence of requests σ is defined as $\alpha^A(\sigma) = \max\{1, \max_{J_i \in \sigma} \frac{f_i - a_i}{d_i - a_i}\}$. In other words α^A measures the factor by which A has delayed jobs in *proportion* to their slack. The goal of the scheduler is to minimize the (maximum) delay factor. We consider worst-case competitive analysis. An online algorithm A is r-competitive if for all request sequences σ, $\alpha^A(\sigma) \leq r\alpha^*(\sigma)$ where $\alpha^*(\sigma)$ is the delay factor of an optimal offline algorithm. Delay factor generalizes the previously studied maximum stretch measure introduced by Bender, Chakraborty and Muthukrishnan [9]. The maximum stretch of a schedule A is $\max_{J_i \in \sigma}(f_i - a_i)/\ell_i$ where ℓ_i is the length or processing time of J_i. By setting $d_i = a_i + \ell_i$ for each request J_i it can be seen that delay factor generalizes maximum stretch.

In the broadcast setting, multiple requests can be satisfied by the same transmission. This model is inspired by a number of recent applications — see [7, 2, 1, 8] for the motivating applications and the growing literature on this topic. More formally, there are n distinct pages or pieces of data that are available in the system, and clients can request a specific page at any time. This is called the *pull-model* since the clients initiate the request and we focus on this model in this paper (in the push-model the server transmits the pages according to some frequency). Multiple outstanding requests for the same page are satisfied by a single transmission of the page. We use $J_{(p,i)}$ to denote the i'th request for a page $p \in \{1, 2, \ldots, n\}$. We let $a_{(p,i)}$ and $d_{(p,i)}$ denote the arrival time and deadline of the request $J_{(p,i)}$. The finish time $f_{(p,i)}$ of a request $J_{(p,i)}$ is defined to be the earliest time af-

*Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801. chekuri@cs.uiuc.edu. Partially supported by NSF grants CCF 0728782 and CNS 0721899.

†Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801. bmosele2@uiuc.edu. Partially supported by NSF grant CNS 0721899.

[1]In this paper we use requests instead of jobs since we also address the broadcast scheduling problem where a request for a page is more appropriate terminology than a job.

ter $a_{(p,i)}$ when the page p is sequentially transmitted by the scheduler. Note that multiple requests for the same page can have the same finish time. The delay factor α^A for an algorithm A over a sequence of requests σ is now defined as $\max\{1, \max_{(p,i)\in\sigma} \frac{f_{(p,i)}-a_{(p,i)}}{d_{(p,i)}-a_{(p,i)}}\}$.

Motivation: There are a variety of metrics in the scheduling literature and some of the well-known and widely used ones are makespan and average response time (or flowtime). More recently, other metrics such as maximum and average stretch, which measure the waiting time in proportion to the size of a request, have been proposed [9, 26, 32]; these measures were motivated by applications in databases and web server systems. Related metrics include L_p norms of response times and stretch [6, 3, 16] for $1 \leq p < \infty$. In a variety of applications such as real-time systems and data gathering systems, requests have deadlines by which they desire to be fulfilled. In real-time systems, a *hard* deadline implies that it cannot be missed, while a *soft* deadline implies some flexibility in violating it. In online settings it is difficult to respect hard deadlines. Previous work has addressed hard deadlines by either considering periodic tasks or other restrictions [12], or by focusing on maximizing throughput (the number of jobs completed by their deadline) [28, 13, 33]. It was recently suggested by Chang et al. [14] that delay factor is a useful and natural relaxation to consider in situations with soft deadlines where we desire all requests to be satisfied. In addition, as we mentioned already, delay factor generalizes maximum stretch which has been previously motivated and studied in [9, 11].

Results: We give the first results for *online* scheduling for minimizing delay factor in both the unicast and broadcast settings. Throughout we assume that requests are allowed to be *preempted* if they have varying processing times. We first prove strong lower bounds on online competitiveness.

- For unicast setting no online algorithm is $\Delta^{0.4}/2$-competitive where Δ is the ratio between the maximum and minimum slacks.
- For broadcast scheduling with n unit-sized pages there is no $n/4$-competitive algorithm.

We resort to resource augmentation analysis, introduced by of Kalyanasundaram and Pruhs [24], to overcome the above lower bounds. In this analysis the online algorithm is given faster machines than the optimal offline algorithm. For $s \geq 1$, an algorithm A is s-speed r-competitive if A when given s-speed machine(s) achieves a competitive ratio of r. We prove the following.

- For unicast setting, for any $\epsilon \in (0, 1]$, there is an $(1+\epsilon)$-speed $O(1/\epsilon)$-competitive algorithm in both single and multiple machine cases. Moreover, the algorithm for the multiple machine case immediately dispatches an arriving request to a machine, and is non-migratory.
- For broadcast setting, for any $\epsilon \in (0, 1]$, there is a $(2+\epsilon)$-speed $O(1/\epsilon^2)$-competitive algorithm for unit-sized (or similar sized) pages. If pages can have varying length, then for any $\epsilon \in (0, 1]$, there is a $(4+\epsilon)$-speed $O(1/\epsilon^2)$-competitive algorithm.

Our results for the unicast setting are related to, and borrow ideas from, previous work on minimizing L_p norms of response time and stretch [6] in the single machine and parallel machine settings [3, 16].

Our main result is for broadcast scheduling. Broadcast scheduling has posed considerable difficulties for algorithm design. In fact most of the known results are for the *offline* setting [25, 21, 22, 23, 5, 4] and several of these use resource augmentation. The difficulty in broadcast scheduling arises from the fact that the online algorithm may transmit a page multiple times to satisfy distinct requests for the same page, while the offline optimum, which knows the sequence in advance, can *save work* by gathering them into a single transmission. Online algorithms that maximize throughput [28, 13, 33, 17] get around this by eliminating requests. Few positive results are known in the online setting where all requests need to be scheduled [8, 19, 20] and the analysis in all of these is quite non-trivial. In contrast, our algorithm and analysis are direct and explicitly demonstrate the value of making requests wait for some duration so as to take advantage of potential future requests for the same page. We hope this idea can be further exploited in other broadcast scheduling contexts. We mention that even in the *offline* setting, only an LP-based 2-speed algorithm is known for delay factor with unit-sized pages [14].

Related Work: We refer the reader to the survey on online scheduling by Pruhs, Sgall and Torng [31] for a comprehensive overview of results and algorithms (see also [30]). For jobs with deadlines, the well-known earliest-deadline-first (EDF) algorithm can be used in the offline setting to check if all the jobs can be completed before their deadline. A substantial amount of literature exists in the real-time systems community in understanding and characterizing restrictions on the job sequence that allow for schedulability of jobs with deadlines when they arrive online or periodically. Previous work on soft deadlines is also concerned with characterizing inputs that allow for bounded tardiness. We refer the reader to [29] for the extensive literature on scheduling issues in real-time systems.

Closely related to our work is that on max stretch [9] where it is shown that no online algorithm is $O(P^{0.3})$ competitive even in the preemptive setting where P is ratio of the largest job size to the smallest job size. [9] also gives an $O(\sqrt{P})$ competitive algorithm which was further refined in [11]. Resource augmentation analysis for L_p norms of

response time and stretch from the work of Bansal and Pruhs [6] implicitly shows that the shortest job first (SJF) algorithm is a $(1+\epsilon)$-speed $O(1/\epsilon)$-competitive algorithm for max stretch. Our work shows that this analysis can be generalized for the delay factor metric. For multiple processors our analysis is inspired by the ideas from [3, 16]. In [10], the authors suggest delay factor as a performance measure under the name of maximum interval stretch. They consider a model in which the speed of processing a job increases as the job approaches its deadline. Implicit in their work is a resource augmentation result in the unicast setting when there exists an offline schedule that finishes all the jobs by their deadline.

Broadcast scheduling has seen a substantial amount of research in recent years; apart from the work that we have already cited we refer the reader to [15, 27], the recent paper of Chang et al. [14], and the surveys [31, 30] for several pointers to known results. Our work on delay factor is inspired by [14]. As we mentioned already, a good amount of the work on broadcast scheduling has been on offline algorithms including NP-hardness results and approximation algorithms (often with resource augmentation). For delay factor there is a 2-speed optimal algorithm in the *offline* setting and it is also known that unless P=NP there is no $2-\epsilon$ approximation [14]. In the online setting the following results are known. For maximum response time, it is shown in [8, 14] that first-in-first-out (FIFO) is 2-competitive. For average response time, Edmonds and Pruhs [19] give a $(4+\epsilon)$-speed $O(1/\epsilon)$-competitive algorithm; their algorithm is an indirect reduction to a complicated algorithm of Edmonds [18] for non-clairvoyant scheduling. They also show in [20] that longest-wait-first (LWF) is a 6-speed $O(1)$-competitive algorithm for average response time. Constant competitive online algorithms for maximizing throughput [28, 13, 33, 17] for unit-sized pages.

We describe our results for the unicast setting in Section 2 and for the broadcast settings in Section 3.

Notation: We let $S_i = d_i - a_i$ denote the slack of J_i in the unicast setting. When requests have varying processing times (or lengths) we use ℓ_i to denote the length of J_i. We assume without loss of generality that $S_i \geq \ell_i$. In the broadcast setting, (p,i) denotes the i'th request for page p. We assume that the requests for a page are ordered by time and hence $a_{(p,i)} \leq a_{(p,j)}$ for $i < j$. In both settings we use Δ to denote the ratio of maximum slack to the minimum slack in a given request sequence.

2 Unicast Scheduling

In this section we address the unicast case where requests are independent. We may thus view requests as jobs although we stick with the notation of requests. For a request J_i, recall that a_i, d_i, ℓ_i, f_i denote the arrival time, deadline, length, and finish time respectively. An instance with all $\ell_i = 1$ (or more generally the processing times are the same) is referred to as a unit-time instance. It is easy to see that preemption does not help much for unit-sized instances. Assuming that the processing times are integer valued then in the single machine setting one can reduce an instance with varying processing time to an instance with unit-times as follows. Replace J_i, with length ℓ_i, by ℓ_i unit-sized requests with the same arrival and deadline as that of J_i.

As we had remarked earlier, scheduling to minimize the maximum stretch is a special case of scheduling to minimize the maximum delay factor. In [9] a lower bound of $P^{1/3}$ is shown for online maximum stretch on a 1-speed machine where P is the ratio of the maximum processing time to the minimum processing time. They show that this bounds holds even when P is known to the algorithm. This implies a lower bound of $\Delta^{1/3}$ for minimizing the maximum delay factor. Here we improve the lower bound for maximum stretch to $P^{0.4}/2$ when the online algorithm is not aware of P. A proof can be found in the appendix.

THEOREM 2.1. *There is no 1-speed $\frac{P^{.4}}{2}$-competitive algorithm for online maximum stretch when P is not known in advance to the algorithm.*

COROLLARY 2.1. *There is no 1-speed $\frac{\Delta^{.4}}{2}$-competitive algorithm for delay factor scheduling when Δ is not known in advance with unit-time requests.*

In the next two subsections we show that with $(1+\epsilon)$ resource augmentation, simple algorithms achieve an $O(1/\epsilon)$ competitive ratio.

2.1 Single Machine Scheduling

We analyze the simple shortest-slack-first (**SSF**) algorithm which at any time t schedules the request with the shortest slack.

Algorithm: **SSF**
- At any time t schedule the request with with the minimum slack which has not been satisfied.

THEOREM 2.2. *The algorithm* **SSF** *is $(1+\epsilon)$-speed $(\frac{1}{\epsilon})$-competitive for minimizing the maximum delay factor in unicast scheduling.*

Proof. Consider an arbitrary request sequence σ and let α be the maximum delay factor achieved by **SSF** on σ. If $\alpha = 1$ there is nothing to prove, so assume that $\alpha > 1$. Let J_i be the request that witnesses α, that is $\alpha = (f_i - a_i)/S_i$. Note that **SSF** does not process any request with slack more than S_i in the interval $[a_i, f_i]$. Let t be the largest value less than or equal to a_i such that **SSF** processed only requests with slack at most S_i in the interval $[t, f_i]$. It follows that **SSF** had no requests with slack $\leq S_i$ just before t. The total work

that **SSF** processed in $[t, f_i]$ on requests with slack less than equal to S_i is $(1+\epsilon)(f_i - t)$ and all these requests arrive in the interval $[t, f_i]$. An optimal offline algorithm with 1-speed can do total work of at most $(f_i - t)$ in the interval $[t, f_i]$ and hence the earliest time by which it can finish these requests is $f_i + \epsilon(f_i - t) \geq f_i + \epsilon(f_i - a_i)$. Since all these requests have slack at most S_i and have arrived before f_i, it follows that $\alpha^* \geq \epsilon(f_i - a_i)/S_i$ where α^* is the maximum delay factor of the optimal offline algorithm with 1-speed machine. Therefore, we have that $\alpha/\alpha^* \leq 1/\epsilon$.

REMARK 2.1. *For unit-time requests, the algorithm that* non-preemptively *schedules requests with the shortest slack is a $(1+\epsilon)$-speed $\frac{2}{\epsilon}$-competitive for maximum delay factor.*

2.2 Multiple Machine Scheduling We now consider delay factor scheduling when there are m machines. To adapt **SSF** to this setting we use intuition from previous work on minimizing L_p norms of flow time and stretch [6, 3, 16]. We develop an algorithm that immediately dispatches an arriving request to a machine, and further does not migrate an assigned request to a different machine once it is assigned. Each machine essentially runs the single machine **SSF** algorithm and thus the only remaining ingredient to describe is the dispatching rule. For this purpose the algorithm groups requests into classes based on their slack. A request J_i is said to be in class k if $S_i \in [2^k, 2^{k+1})$. The algorithm maintains the total processing time of requests (referred to as *volume*) that have been assigned to machine x in each class k. Let $U^x_{=k}(t)$ denote the total processing time of requests of class k assigned to machine x by time t. With this notation, the algorithm **SSF-ID** (for **SSF** with immediate dispatch) can be described.

Algorithm: **SSF-ID**

- When a new request J_i of class k arrives at time t, assign it to a machine x where $U^x_{=k}(t) = \min_y U^y_{=k}(t)$.
- Use **SSF** on each machine separately.

The rest of this section is devoted to the proof of the following theorem.

THEOREM 2.3. **SSF-ID** *is a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive algorithm for online delay factor scheduling on m machines.*

We need a fair amount of notation. For each time t, machine x, and class k we define several quantities. For example $U^x_{=k}(t)$ is the total volume assigned to machine x in class k by time t. We use the predicate "$\leq k$" to indicate classes 1 to k. Thus $U^x_{\leq k}(t)$ is the total volume assigned to machine x in classes 1 to k. We let $R^x_{=k}(t)$ to denote the remaining processing time on machine x at time t and let $P^x_{=k}(t)$ denote the total volume that x has finished on requests in class k by time t. Note that $P^x_{=k}(t) = U^x_{=k}(t) - R^x_{=k}(t)$. All these quantities refer to the algorithm **SSF-ID**. We use $V^*_{=k}(t)$ and $V_{=k}(t)$ to denote the remaining volume of requests in class k in an optimal offline algorithm with speed 1 and **SSF-ID** with speed $(1+\epsilon)$, respectively. Observe that $V_{=k}(t) = \sum_x R^x_{=k}(t)$. The quantities $V^*_{\leq k}(t)$ and $V_{\leq k}(t)$ are defined analogously.

The algorithm **SSF-ID** balances the amount of processing time for requests with similar slack. Note that the assignment of requests is not based on the current volume of unfinished requests on the machines, rather the assignment is based on the volume of requests that were assigned in the past to different machines. We begin our proof by showing that the volume of processing time of requests less than or equal to some slack class is almost the same on the different machines at any time. Several of these lemmas are essentially the same as in [3].

OBSERVATION 1. *For any time t and two machines x and y, $|U^x_{=k}(t) - U^x_{=k}(t)| \leq 2^{k+1}$. This also implies that $|U^x_{\leq k}(t) - U^x_{\leq k}(t)| \leq 2^{k+2}$.*

Proof. The first inequality holds since all of the requests of class k are of size $\leq 2^{k+1}$. The second inequality follows easily form the first.

Proofs of the next two lemmas can be found in the appendix.

LEMMA 2.2. *Consider any two machines x and y. The difference in volume of requests that have already have been processed is bounded as $|P^x_{\leq k}(t) - P^y_{\leq k}(t)| \leq 2^{k+2}$.*

LEMMA 2.3. *At any time t the difference between the residual volume of requests that needs to be processed, on any two different machines, x and y is bounded as $|R^x_{\leq k}(t) - R^y_{\leq k}(t)| \leq 2^{k+3}$.*

COROLLARY 2.2. *At any time t, $V^*_{\leq k}(t) \geq V_{\leq k}(t) - m2^{k+3}$.*

Now we get to the proof of the upper bound on **SSF-ID**, when given $(1+\epsilon)$-speed, in a similar fashion to the single machine case. Consider an arbitrary request sequence σ and let J_i be the request that witnesses the delay factor α of **SSF-ID** on σ. Let k be the class of J_i. Therefore $\alpha = (f_i - a_i)/S_i$. Also, let x be the machine on which J_i was processed by **SSF-ID**. We use α^* to denote the delay factor of some fixed optimal offline algorithm that uses m machines of speed 1.

Let t be the last time before a_i when machine x processed a request of class $> k$. Note that $t \leq a_i$ since

x does not process any request of class $> k$ in the interval $[a_i, f_i]$. At time t we know by Corollary 2.2 that $V^*_{\leq k}(t) \geq V_{\leq k}(t) - m2^{k+3}$. If $f_i \leq a_i + 2^{k+4}$ then **SSF-ID** achieves a competitive ratio of 16 since J_i is in class k. Thus we will assume from now on that $f_i > a_i + 2^{k+4}$.

In the interval $I = [t, f_i)$, **SSF-ID** completes a total volume of of $(1+\epsilon)(f_i - t)$ on machine x. Using Lemma 2.2, any other machine y also processes a volume of $(1+\epsilon)(f_i - t) - 2^{k+3}$ in I. Thus the total volume processed by **SSF-ID** during I in requests of classes $\leq k$ is at least $m(1+\epsilon)(f_i - t) - m2^{k+3}$. During I, the optimal algorithm finishes at most $m(f_i - t)$ volume in classes $\leq k$. Combining this with Corollary 2.2, we see that

$$\begin{aligned} V^*_{\leq k}(f_i) &\geq V_{\leq k}(t) - m2^{k+3} + m(1+\epsilon)(f_i - t) \\ &\quad - m2^{k+3} \\ &\geq V_{\leq k}(t) + m(1+\epsilon)(f_i - t) - m2^{k+4} \\ &\geq \epsilon m(f_i - t). \end{aligned}$$

In the last inequality we use the fact that $f_i - t \geq f_i - a_i \geq 2^{k+4}$. Without loss of generality assume that no requests arrive exactly at f_i. Therefore $V^*_{\leq k}(f_i)$ is the total volume of requests in classes 1 to k that the optimal algorithm has left to finish at time f_i and all these requests have arrived before f_i. The earliest time that the optimal algorithm can finish all these requests is by $f_i + \epsilon(f_i - t)$ and therefore it follows that $\alpha^* \geq \epsilon(f_i - t)/2^{k+1}$. Since $\alpha \leq (f_i - a_i)/2^k$ and $t \leq a_i$, it follows that $\alpha \leq 2\alpha^*/\epsilon$.

Thus $\alpha \leq \max\{16, 2\alpha^*/\epsilon\}$ which finishes the proof of Theorem 2.3.

3 Broadcast Scheduling

We now move our attention to the broadcast model where multiple requests can be satisfied by the transmission of a single page. Most of the literature in broadcast scheduling is concerned with the case where all pages have the same size which is assumed to be unit. A notable exception is the work of Edmonds and Pruhs [19]. Here we consider both the unit-sized as well as arbitrary sized pages.

We start by showing that no 1-speed online algorithm can be $(n/4)$-competitive for delay factor where n is the total number of unit-sized pages. We then show in Section 3.1 that there is a $(2+\epsilon)$-speed $O(1/\epsilon^2)$-competitive algorithm for unit-sized pages. We prove this for the single machine setting and it readily extends to the multiple machine case. Finally, we extend our algorithm and analysis to the case of different page sizes to obtain a $(4+\epsilon)$-speed $O(1/\epsilon^2)$-competitive algorithm in Section 3.2. We believe that this can be extended to the multiple machine setting but leave it for future work.

THEOREM 3.1. *Every 1-speed online algorithm for broadcast scheduling to minimize the maximum delay factor is $\Omega(n)$-competitive where n is number of unit-sized pages.*

The proof of Theorem 3.1 can be found in the appendix.

3.1 A Competitive Algorithm for Unit-sized Pages

We now develop an online algorithm, for unit-sized pages, that is competitive given extra speed. We first note that, in this case, the arrival times are assumed to be integral. It can be seen that this assumption can only increase the delay factor by one. It is easy to check that unlike in the unicast setting, simple algorithms such as **SSF** fail to be constant competitive in the broadcast setting even with extra speed. The reason for this is that any simple algorithm can be made to do an arbitrary amount of "extra" work by repeatedly requesting the same page while the adversary can wait and finish all these requests with a single transmission. We use this intuition to develop a variant of **SSF** that *adaptively* introduces waiting time for requests. The algorithm uses a single real-valued parameter $c < 1$ to control the waiting period. The algorithm **SSF-W** (**SSF** with waiting) is formally defined below. We note that the algorithm is non-preemptive in that a request once scheduled is not preempted. As we mentioned earlier, for unit-sized requests, preemption is not very helpful. The algorithm keeps track of the maximum delay factor it has seen so far, α_t (we set $\alpha_0 = 1$). The value of α_t depends on unsatisfied requests where the delay factor of an unsatisfied request is its delay factor at time t. The important feature of the algorithm is that it considers requests for scheduling only after they have waited sufficiently long when compared to their *adaptive* slack.

Algorithm: **SSF-W**

- Let α_t be the maximum delay factor **SSF-W** has at time t.
- At time t, let $Q(t) = \{ J_{(p,i)} \mid J_{(p,i)}$ has not been satisfied and $t - a_{(p,i)} \geq c \cdot \alpha_t \cdot S_{(p,i)} \}$.
- If the machine is free at t, schedule the request in $Q(t)$ with the smallest slack *non-preemptively*.

We now analyze **SSF-W** when it is given a $(2+\epsilon)$-speed machine. Let σ be an arbitrary sequence of requests. Consider the *first* time t where **SSF-W** achieves the maximum delay factor $\alpha^{\text{SSF-W}}$. At time t, **SSF-W** must have finished a request $J_{(p,k)}$ which caused **SSF-W** to have this delay factor. Hence, **SSF-W** has a maximum delay factor of $(f_{(p,k)} - a_{(p,k)})/S_{(p,k)}$ where $f_{(p,k)}$ is the time **SSF-W** satisfies request $J_{(p,k)}$. We let OPT denote some fixed offline optimum algorithm and let α^* denote the optimum delay factor.

We now prove the most interesting difference between unicast and broadcast scheduling. The following lemma shows that forcing a request to wait in the queue, for a small

period of time, can guarantee that our algorithm is satisfying as many requests as OPT by a single broadcast unless OPT has a similar delay factor.

Since $J_{(p,k)}$ defines $\alpha^{\textbf{SSF-W}}$, we observe that from time $t' = a_{(p,k)} + c(f_{(p,k)} - a_{(p,k)})$, the request $J(p,k)$ is ready to be scheduled and hence the algorithm is continuously busy in the interval $I = [t', f_{(p,k)}]$ processing requests of slack no more than that of $J_{(p,k)}$.

LEMMA 3.1. *Consider the interval $I = [t', f_{(p,k)})$. Suppose two distinct requests $J_{(x,j)}$ and $J_{(x,i)}$ for the same page x were satisfied by* **SSF-W** *during I at different times. If* OPT *satisfies both of these requests by a single broadcast then $\alpha^{\textbf{SSF-W}} \leq \frac{1}{c^2}\alpha^*$.*

Proof. Without loss of generality assume that $i > j$; therefore $a_{(x,j)} \leq a_{(x,i)}$. Request $J_{(x,i)}$ must have arrived during I, otherwise **SSF-W** would have satisfied $J_{(x,i)}$ when it satisfied $J_{(x,j)}$. We observe that $\alpha_{t'} \geq \frac{c(f_{(p,k)} - a_{(p,k)})}{S_{(p,k)}} \geq c\alpha^{\textbf{SSF-W}}$ since $J_{(p,k)}$ is still alive at t'.

Since $J_{(x,j)}$ was scheduled after t', it follows that **SSF-W** would have made it wait at least $c\alpha_{t'} S_{(x,j)}$ which implies that

$$f_{(x,j)} \geq a_{(x,j)} + c\alpha_{t'} S_{(x,j)}.$$

Note that **SSF-W** satisfies $J_{(x,i)}$ by a separate broadcast from $J_{(x,j)}$ which implies that $a_{(x,i)} \geq f_{(x,j)}$. However, OPT satisfies both requests by the same transmission which implies that OPT finishes $J_{(x,j)}$ no earlier than $a_{(x,i)}$. Therefore the delay factor of OPT is at least the delay factor for $J_{(x,j)}$ in OPT which implies that

$$\begin{aligned} \alpha^* &\geq \frac{a_{(x,i)} - a_{(x,j)}}{S_{(x,j)}} \geq \frac{f_{(x,j)} - a_{(x,j)}}{S_{(x,j)}} \\ &\geq \frac{c\alpha_{t'} S_{(x,j)}}{S_{(x,j)}} \geq c\alpha_{t'} \geq c^2 \alpha^{\textbf{SSF-W}}. \end{aligned}$$

Note that previous lemma holds for any two requests scheduled by **SSF-W** during interval I regardless of when OPT schedules them, perhaps even after $f_{(p,k)}$.

LEMMA 3.2. *Consider the interval $I = [t', f_{(p,k)}]$. Any request which* **SSF-W** *scheduled during I must have arrived after time $a_{(p,k)} - (1-c)(f_{(p,k)} - a_{(p,k)})$.*

Proof. For sake of contradiction, assume that a request $J_{(x,j)}$ scheduled by **SSF-W** on the interval I has arrival time less than $a_{(p,k)} - (1-c)(f_{(p,k)} - a_{(p,k)})$. Since **SSF-W** finishes this request during I, $f_{(x,j)} \geq a_{(p,k)} + c(f_{(p,k)} - a_{(p,k)})$. Also, as we observed before, all requests scheduled during I by **SSF-W** have slack no more than that of $J_{(p,k)}$ which implies that $S_{(x,j)} \leq S_{(p,k)}$. However this implies that the delay factor of $J_{(x,j)}$ is at least

$$\begin{aligned} \frac{f_{(x,j)} - a_{(x,j)}}{S_{(x,j)}} &\geq (a_{(p,k)} + c(f_{(p,k)} - a_{(p,k)}) - (a_{(p,k)} \\ &\quad -(1-c)(f_{(p,k)} - a_{(p,k)}))\frac{1}{S_{(x,j)}} \\ &\geq \frac{f_{(p,k)} - a_{(p,k)}}{S_{(x,j)}} \geq \frac{f_{(p,k)} - a_{(p,k)}}{S_{(p,k)}}. \\ &\geq \alpha^{\textbf{SSF-W}} \end{aligned}$$

This is a contradiction to the fact that $J_{(p,k)}$ is the first request that witnessed the maximum delay factor of **SSF-W**.

Now we are ready to prove the competitiveness of **SSF-W**.

LEMMA 3.3. *The algorithm* **SSF-W** *when given a $(2+\epsilon)$-speed machines satisfies $\alpha^{\textbf{SSF-W}} \leq \max\{\frac{1}{c^2}, \frac{1}{\epsilon - c\epsilon - c}\}\alpha^*$.*

Proof. The number of broadcasts which **SSF-W** transmits during the interval $I = [t', f_{(p,k)}]$ is

$$(2+\epsilon)(f_{(p,k)} - t') \geq (2+\epsilon)(1-c)(f_{(p,k)} - a_{(p,k)}).$$

From Lemma 3.2, all the requests processed during I have arrived no earlier than $a_{(p,k)} - (1-c)(f_{(p,k)} - a_{(p,k)})$. Also, each of these requests has slack no more than $S_{(p,k)}$. We restrict attention to the requests satisfied by **SSF-W** during I. We consider two cases

First, if there are two requests for the same page that **SSF-W** satisfies via distinct broadcasts but OPT satisfies using one broadcast, then by Lemma 3.1, $\alpha^{\textbf{SSF-W}} \leq \frac{1}{c^2}\alpha^*$ and we are done.

Second, we assume that OPT does not merge two requests for the same page whenever **SSF-W** does not do so. It follows that OPT also has to broadcast $(2+\epsilon)(1-c)(f_{(p,k)} - a_{(p,k)})$ pages to satisfy the requests that **SSF-W** did during I. Since these requests arrived no earlier than $a_{(p,k)} - (1-c)(f_{(p,k)} - a_{(p,k)})$, OPT, which has a 1-speed machine, can finish them at the earliest by

$$\begin{aligned} &(2+\epsilon)(1-c)(f_{(p,k)} - a_{(p,k)}) + a_{(p,k)} \\ &\qquad -(1-c)(f_{(p,k)} - a_{(p,k)}) \\ &\geq f_{(p,k)} + (\epsilon - c - c\epsilon)(f_{(p,k)} - a_{(p,k)}). \end{aligned}$$

Since each of these requests has slack at most $S_{(p,k)}$ and arrived no later than $f_{(p,k)}$, we have that

$$\begin{aligned} \alpha^* &\geq (f_{(p,k)} + (\epsilon - c - c\epsilon)(f_{(p,k)} - a_{(p,k)}) - f_{(p,k)})/S_{(p,k)} \\ &\geq (\epsilon - c - c\epsilon)(f_{(p,k)} - a_{(p,k)})/S_{(p,k)} \\ &\geq (\epsilon - c - c\epsilon)\alpha^{\textbf{SSF-W}}. \end{aligned}$$

The previous lemma yields the following theorem.

THEOREM 3.2. *With $c = \epsilon/2$,* **SSF-W** *is a $(2+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive algorithm for minimizing the maximum delay factor in broadcast scheduling with unit-sized pages.*

It may appear that **SSF-W** needs knowledge of ϵ. However, another way to interpret Lemma 3.3 is that for any fixed constant c, **SSF-W** with parameter c is constant competitive in all settings where its machine is at least $(2+2\sqrt{c})$ times the speed of the optimal algorithm. Of course, it would be ideal to have an algorithm scales with ϵ without any knowledge of ϵ. We leave the existence of such an algorithm for future work.

Now consider having m machines where we have $(2+\epsilon)$-speed. Since we are using unit time requests, this is analogous to OPT having one m-speed machine and **SSF-W** having a $(m(2+\epsilon))$-speed machine. Thus, one can extend the above analysis to the multiple machine setting with unit-sized pages in a straight forward fashion.

3.2 Varying Page Sizes In this section we generalize our algorithm for unit-sized pages to the setting where each page has potentially a different page size. We let ℓ_p denote the length of page p. In this setting we allow preemption of transmissions. Suppose the transmission of a page p is started at time t_1 and ends at time t_2; p may be preempted for other transmissions and hence $t_2 - t_1 \geq p$. A request for a page p is satisfied by the transmission of p during the interval $[t_1, t_2]$ only if the request arrives before t_1. It is possible that the transmission of a page p is abandoned and *restarted* due to the arrival of a new request for p with a smaller slack. This may lead to further wasted work by the algorithm and increases the complexity of the analysis. Here we show that a natural adaptation of **SSF-W** is competitive even in this more general setting if it is given $(4+\epsilon)$-speed.

We outline the details of modifications to **SSF-W**. As before, at any time t, the algorithm considers broadcasting a request $J_{(p,i)}$ if $t - a_{(p,i)} \geq c\alpha_t S_{(p,i)}$; these are requests that have waited long enough. Among these requests, the one with the smallest slack is scheduled. Note that the waiting is only for requests that have not yet been started; any request that has already started transmission is available to be scheduled. The algorithm breaks ties arbitrarily, yet ensures that if a request $J_{(p,k)}$ is started before a request $J_{(p',j)}$ then $J_{(p,k)}$ will be finished before request $J_{(p',j)}$. Note that the algorithm may preempt a request $J_{(p,i)}$ by another request $J_{(p,k)}$ for the same page p even though $i < k$ if $S_{(p,k)} < S_{(p,i)}$. In this case the transmission of $J_{(p,i)}$ is effectively abandoned. Note that transmission of a page p may be repeatedly abandoned.

We now analyze the algorithm assuming that it has a $(4+\epsilon)$-speed advantage over the optimal offline algorithm. The extra factor in speed is needed in our analysis to handle the extra wasted work due to potential retransmission of a page p after a large portion of it has already been transmitted. As before, let σ be a sequence of requests and let t be the *first* time **SSF-W** achieves the maximum delay factor $\alpha^{\textbf{SSF-W}}$. At time t, it must be the case that a request $J_{(p,k)}$ was finished which caused **SSF-W** to have his maximum delay factor. Hence, **SSF-W** has a maximum delay factor of $(f_{(p,k)} - a_{(p,k)})/S_{(p,k)}$ where $f_{(p,k)}$ is the time **SSF-W** satisfied request $J_{(p,k)}$.

As with the case with unit time requests, at time $t' = a_{(p,k)} + c(f_{(p,k)} - a_{(p,k)})$ the request $J_{(p,k)}$ is ready to be scheduled and the algorithm is busy on the interval $I = [t', f_{(p,k)}]$ processing requests of slack at most $S_{(p,k)}$.

We say that a request $J_{(p,i)}$ is *started* at time t if t is the first time at which the algorithm picked $J_{(p,i)}$ to transmit its page. Multiple requests may be waiting for the same page p but only the request with the smallest slack that is picked by the algorithm is said to be started. Thus a request may be satisfied although it is technically not started. Also, a request $J_{(p,i)}$ that is started may be abandoned by the start of another request for the same page.

The lemma below is analogous to Lemma 3.1 but requires a more careful statement since requests may now be started and abandoned.

LEMMA 3.4. *Consider two distinct requests $J_{(x,j)}$ and $J_{(x,i)}$ for the same page x where $i > j$ such that they are both satisfied by* OPT *via the same transmission. If* **SSF-W** *starts $J_{(x,j)}$ in $[t', f_{(p,k)}]$ before the arrival of $J_{(x,i)}$, then $\alpha^{\textbf{SSF-W}} \leq \frac{1}{c^2}\alpha^*$.*

Observe that the request $J_{(x,j)}$ may be satisfied together with $J_{(x,i)}$ even though it starts before the arrival of $J_{(x,i)}$.

Proof. As before, $\alpha_{t'} \geq \frac{c(f_{(p,k)} - a_{(p,k)})}{S_{(p,k)}} \geq c\alpha^{\textbf{SSF-W}}$ since $J_{(p,k)}$ is still alive at t'. Since $J_{(x,j)}$ is started after t', it follows that **SSF-W** would have made it wait at least $c\alpha_{t'}S_{(x,j)}$. Let $t \geq t'$ be the start time of $J_{(x,j)}$. Therefore $t \geq a_{(x,j)} + c\alpha_{t'}S_{(x,j)}$. By our assumption, $t < a_{(x,i)}$ and therefore $a_{(x,i)} > a_{(x,j)} + c\alpha_{t'}S_{(x,j)}$.

Since OPT satisfies these two requests by the same transmission, the finish time of $J_{(x,j)}$ in OPT is at least $a_{(x,i)}$. Therefore,

$$\alpha^* \geq \frac{a_{(x,i)} - a_{(x,j)}}{S_{(x,j)}} \geq \frac{c\alpha_{t'}S_{(x,j)}}{S_{(x,j)}} \geq c\alpha_{t'} \geq c^2\alpha^{\textbf{SSF-W}}.$$

The proof of the lemma below is very similar to that of Lemma 3.2.

LEMMA 3.5. *Consider the interval $I = [t', f_{(p,k)}]$. Any request which is alive with slack $\leq S_{(p,k)}$, but unsatisfied by* **SSF-W** *at time t' must have arrived after time $a_{(p,k)} - (1-c)(f_{(p,k)} - a_{(p,k)})$.*

Now we are ready to prove the competitiveness of **SSF-W**. Although the outline of the proof is similar to that of Lemma 3.3, it requires more careful reasoning to handle the impact of abandoned transmissions of pages. Here is where we crucially rely on the speed of $(4+\epsilon)$.

LEMMA 3.6. *The algorithm* **SSF-W** *when given a* $(4+\epsilon)$*-speed machine satisfies* $\alpha^{\textbf{SSF-W}} \leq \max\{\frac{1}{c^2}, \frac{2}{(\epsilon - c\epsilon - 2c)}\}\alpha^*$.

Proof. We consider the set of requests satisfied by **SSF-W** during the interval $I = [t', f_{(p,k)}]$. All of these requests have slack at most $S_{(p,k)}$, and from Lemma 3.5 and the property of the algorithm, have arrived no earlier than $a_{(p,k)} - (1-c)(f_{(p,k)} - a_{(p,k)})$. Since **SSF-W** is busy throughout I, the volume of broadcasts it transmits during I is $(4+\epsilon)(f_{(p,k)} - t') \geq (4+\epsilon)(1-c)(f_{(p,k)} - a_{(p,k)})$.

We now argue that either Lemma 3.4 applies in which case $\alpha^{\textbf{SSF-W}} \leq \alpha^*/c^2$, or OPT has to transmit a comparable amount of volume to that of **SSF-W**.

Fix a page x and consider the transmissions for x that **SSF-W** does during I. Let $J_{(x,i_1)}, J_{(x,i_2)}, \ldots, J_{(x,i_r)}$ be distinct requests for x which cause these transmissions. Amongst these, only $J_{(x,i_1)}$ may have started before t', the rest start during I. Note that we are not claiming that these transmissions are satisfied separately; some of them may be preempted and then satisfied together. Observe that if $J_{(x,i_h)}$ starts at some time t then it implies that no request $J_{(x,i_{h'})}$ for $h' > h$ has arrived by time t. Therefore by Lemma 3.4, if OPT satisfies any two of these requests that **SSF-W** started in I by the same transmission, $\alpha^{\textbf{SSF-W}} \leq \alpha^*/c^2$ and we are done.

Otherwise, OPT satisfies each of $J_{(x,i_2)}, \ldots, J_{(x,i_r)}$ by separate transmissions. (If $J_{(x,i_1)}$ was started by **SSF-W** before t', OPT could satisfy $J_{(x,i_1)}$ and $J_{(x,i_2)}$ together and we would not be able to invoke Lemma 3.4). Therefore if $r \geq 2$ then the total volume of transmissions that OPT does to satisfy these requests for page x is at least $(r-1)\ell_x$ while **SSF-W** does at most $r\ell_x$. If $r = 1$ then both OPT and **SSF-W** transmit page x once for its entire page length. In either case, the total volume of transmissions that OPT does is at least half those of **SSF-W**. Since x was arbitrary, it follows that the total volume of transmissions that OPT does to satisfy requests that **SSF-W** satisfies during I is at least $\frac{1}{2}(4+\epsilon)(1-c)(f_{(p,k)} - a_{(p,k)})$.

From Lemma 3.5, all the requests that **SSF-W** processes during I arrived no earlier than $a_{(p,k)} - (1-c)(f_{(p,k)} - a_{(p,k)})$. Since OPT has a 1-speed machine, it follows that OPT can finish these requests only by time

$$\frac{1}{2}(4+\epsilon)(1-c)(f_{(p,k)} - a_{(p,k)}) + a_{(p,k)} - (1-c)(f_{(p,k)} - a_{(p,k)})$$

$$\geq f_{(p,k)} + \frac{1}{2}(\epsilon - 2c - c\epsilon)(f_{(p,k)} - a_{(p,k)}).$$

Since each of these requests have slack at most $S_{(p,k)}$ and arrive no later than $f_{(p,k)}$,

$$\begin{aligned}
\alpha^* &\geq (f_{(p,k)} + \frac{1}{2}(\epsilon - 2c - c\epsilon)(f_{(p,k)} - a_{(p,k)}) \\
&\quad - f_{(p,k)})/S_{(p,k)} \\
&\geq \frac{1}{2}(\epsilon - 2c - c\epsilon)(f_{(p,k)} - a_{(p,k)})/S_{(p,k)} \\
&\geq \frac{1}{2}(\epsilon - 2c - c\epsilon)\alpha^{\textbf{SSF-W}}.
\end{aligned}$$

We thus obtain the following.

THEOREM 3.3. *With* $c = \epsilon/3$*,* **SSF-W** *is a* $(4+\epsilon)$*-speed* $O(\frac{1}{\epsilon^2})$*-competitive algorithm for minimizing the maximum delay factor in broadcast scheduling with arbitrary page sizes.*

4 Concluding Remarks

In this paper we have initiated the study of online algorithms for minimizing delay factor when requests have deadlines. Our main result is broadcast scheduling where the algorithm and analysis demonstrates the utility of making requests wait. We hope that this and related ideas are helpful in understanding other performance measures in the broadcast setting. Particularly, can 'waiting' combined with some known algorithm, like most requests first, be used to improve the current best known online algorithm for minimizing the average response time? Another interesting problem is whether there is a $(1+\epsilon)$-speed $O(1)$-competitive algorithm for delay factor. Our algorithm has a parameter that controls the waiting time. Is there an algorithm that avoids taking an explicit parameter and "learns" it along the way?

Acknowledgments: We thank Samir Khuller for clarifications on previous work and for his encouragement.

References

[1] S. Acharya, M. Franklin, and S. Zdonik. Dissemination-based data delivery using broadcast disks. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 2(6):50–60, Dec 1995.

[2] Demet Aksoy and Michael J. Franklin. "rxw: A scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Trans. Netw.*, 7(6):846–860, 1999.

[3] Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 11–18, 2003.

[4] Nikhil Bansal, Moses Charikar, Sanjeev Khanna, and Joseph (Seffi) Naor. Approximating the average response time in broadcast scheduling. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 215–221, 2005.

[5] Nikhil Bansal, Don Coppersmith, and Maxim Sviridenko. Improved approximation algorithms for broadcast scheduling. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 344–353, 2006.

[6] Nikhil Bansal and Kirk Pruhs. Server scheduling in the l_p norm: a rising tide lifts all boat. In *STOC*, pages 242–250, 2003.

[7] Amotz Bar-Noy, Randeep Bhatia, Joseph (Seffi) Naor, and Baruch Schieber. Minimizing service and operation costs of periodic scheduling. *Math. Oper. Res.*, 27(3):518–544, 2002.

[8] Yair Bartal and S. Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 558–559, 2000.

[9] Michael A. Bender, Soumen Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 270–279, 1998.

[10] Michael A. Bender, Raphaël Clifford, and Kostas Tsichlas. Scheduling algorithms for procrastinators. *J. Scheduling*, 11(2):95–104, 2008.

[11] Michael A. Bender, S. Muthukrishnan, and Rajmohan Rajaraman. Improved algorithms for stretch scheduling. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 762–771, 2002.

[12] A. Burns and S. Baruah. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering*, 2(1):74–97, 2008.

[13] Wun-Tat Chan, Tak Wah Lam, Hing-Fung Ting, and Prudence W. H. Wong. New results on on-demand broadcasting with deadline via job scheduling with cancellation. In Kyung-Yong Chwa and J. Ian Munro, editors, *COCOON*, volume 3106 of *Lecture Notes in Computer Science*, pages 210–218, 2004.

[14] Jessica Chang, Thomas Erlebach, Renars Gailis, and Samir Khuller. Broadcast scheduling: algorithms and complexity. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 473–482. Society for Industrial and Applied Mathematics, 2008.

[15] Moses Charikar and Samir Khuller. A robust maximum completion time measure for scheduling. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 324–333, 2006.

[16] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In László Babai, editor, *STOC*, pages 363–372, 2004.

[17] Marek Chrobak, Christoph Dürr, Wojciech Jawor, Lukasz Kowalik, and Maciej Kurowski. A note on scheduling equal-length jobs to maximize throughput. *J. of Scheduling*, 9(1):71–73, 2006.

[18] Jeff Edmonds. Scheduling in the dark. *Theor. Comput. Sci.*, 235(1):109–141, 2000.

[19] Jeff Edmonds and Kirk Pruhs. Multicast pull scheduling: When fairness is fine. *Algorithmica*, 36(3):315–330, 2003.

[20] Jeff Edmonds and Kirk Pruhs. A maiden analysis of longest wait first. *ACM Trans. Algorithms*, 1(1):14–32, 2005.

[21] Thomas Erlebach and Alexander Hall. Np-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 194–202, 2002.

[22] Rajiv Gandhi, Samir Khuller, Yoo-Ah Kim, and Yung-Chun (Justin) Wan. Algorithms for minimizing response time in broadcast scheduling. *Algorithmica*, 38(4):597–608, 2004.

[23] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *J. ACM*, 53(3):324–360, 2006.

[24] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[25] Bala Kalyanasundaram, Kirk Pruhs, and Mahendran Velauthapillai. Scheduling broadcasts in wireless networks. *Journal of Scheduling*, 4(6):339–354, 2000.

[26] David Karger, C. Stein, and Joel Wein. Scheduling algorithms. In M.J. Atallah, editor, *Handbook on Algorithms and Theory of Computation*, chapter 34. 1999.

[27] Samir Khuller and Yoo Ah Kim. Equivalence of two linear programming relaxations for broadcast scheduling. *Oper. Res. Lett.*, 32(5):473–478, 2004.

[28] Jae-Hoon Kim and Kyung-Yong Chwa. Scheduling broadcasts with deadlines. *Theor. Comput. Sci.*, 325(3):479–488, 2004.

[29] Insup Lee, Joseph Y-T. Leung, and Sang Son, editors. *Handbook of Real-Time and Embedded Systems*. CRC Press, 2007.

[30] Kirk Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Perform. Eval. Rev.*, 34(4):52–58, 2007.

[31] Kirk Pruhs, Jiri Sgall, and Eric Torng. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter Online Scheduling. 2004.

[32] Jiri Sgall. On-line scheduling. In *Developments from a June 1996 seminar on Online algorithms*, pages 196–231. Springer-Verlag, 1998.

[33] Feifeng Zheng, Stanley P. Y. Fung, Wun-Tat Chan, Francis Y. L. Chin, Chung Keung Poon, and Prudence W. H. Wong. Improved on-line broadcast scheduling with deadlines. In Danny Z. Chen and D. T. Lee, editors, *COCOON*, volume 4112 of *Lecture Notes in Computer Science*, pages 320–329, 2006.

A Appendix

A.1 Proof of Theorem 2.1

Proof. For sake of contradiction, assume that some algorithm that achieves a competitive ratio better than $\frac{P^{.4}}{2}$ exists. Now consider the following example.

> `Type 1`: At time 0 let the client request a page with processing time and deadline P. This request has slack P.
>
> `Type 2`: At times $P - P^{.6}, P, P + P^{.6}, \ldots, P^{1.16} - P^{.6}$ let the client request a page with processing time $P^{.6}$ and a deadline $P^{.6}$

time units after its arrival time. These requests have slack $P^{.6}$.

Consider time $P^{1.16}$. Assume that this is all of the requests which the client makes. The optimal solution schedules these requests in a first in first out fashion. The optimal schedule finishes request type 1 by its deadline. The requests of type 2 then finish at $P^{.6}$ time units after their deadline. Thus, the delay factor for the optimal schedule is $2P^{.6}/P^{.6} = 2$.

The maximum ratio of maximum to minimum slack values seen so far is $\frac{P}{P^{.6}} = P^{.4}$. Thus, the maximum delay factor our algorithm can have is $(P^{.4})^{.4}/2 = P^{.16}/2$. Consider having the request of type 1 still in the deterministic algorithms queue. At time $P^{1.16}$, the algorithm has achieved a delay factor of at least $\frac{P^{1.16}}{P} = P^{.16}$. Thus, the algorithm has a competitive ratio of at least $\frac{P^{.16}}{2}$, a contradiction. Therefore, at time $P^{1.16}$ the algorithm must have finished the request of type 1. Now, immediately after this time, requests of type 3 arrive.

> `Type 3`: Starting at time $P^{1.16}$ the client requests $P^{1.2} - P^{.6}$ unit processing time requests each with a deadline one time unit after their arrival time. These requests arrive one after another, each time unit. The slack of these requests is 1.

These are all of the requests which are sent. The optimal solution schedules the request of type 1 until time $P^{.4}$, thus has $P^{.6}$ processing time left to finish this request. Then the optimal solution schedules the type 2 and type 3 requests as they arrive, giving them a delay factor of 1. At time $P^{1.16} + P^{1.2} - P^{.6}$ the optimal solution schedules the request of type 1 to completion. Thus delay factor of this solution is $\frac{P^{1.16}+P^{1.2}}{P} \leq 2P^{.2}$.

Our algorithm must have scheduled the request of type 1 by time $P^{1.16}$. Thus the last request it finishes is either of type 2 or type 3. If the request is of type 2 then this request must have waited for all requests of type 3 to finish along with its processing time, thus the delay factor is at least $\frac{P^{1.2}+P^{.6}}{P^{.6}} \geq P^{.6}$. If the last request satisfied by the algorithm is of type 3, then this request must have waited for a request of type 2 to finish, so the delay factor is at least $P^{.6}$. In either case, the competitive ratio of the algorithm is at least $\frac{P^{.6}}{2P^{.2}} = \frac{P^{.4}}{2}$, a contradiction.

A.2 Proof of Lemma 2.2

Proof. Suppose the lemma is false. Then there is a first time t_0 when $P^x_{\leq k}(t_0) - P^y_{\leq k}(t_0) = 2^{k+2}$ and small constant $\delta t > 0$ such that $P^x_{\leq k}(t_0 + \delta t) - P^y_{\leq k}(t_0 + \delta t) > 2^{k+2}$. Let $t' = t_0 + \delta t$. For this to occur, x processes a request of class $\leq k$ during the interval $I = [t_0, t']$ while y processes a request of class $> k$. Since each machine uses **SSF**, it must be that y had no requests in classes $\leq k$ during I which implies that $U^y_{\leq k}(t') = P^y_{\leq k}(t')$. Therefore,

$$U^y_{\leq k}(t') = P^y_{\leq k}(t') < P^x_{\leq k}(t') - 2^{k+2} \leq U^x_{\leq k}(t') - 2^{k+2},$$

since $P^x_{\leq k}(t') \leq U^x_{\leq k}(t')$. However, this implies that

$$U^y_{\leq k}(t') < U^x_{\leq k}(t') - 2^{k+2},$$

a contradiction to Observation 1.

A.3 Proof of Lemma 2.3

Proof. Combining Observation 1, Lemma 2.2, and the fact that $R(t) = U(t) - P(t)$ by definition then,

$$|R^x_{\leq k}(t) - R^y_{\leq k}(t)| \leq |U^x_{\leq k}(t) - U^y_{\leq k}(t)| + |P^x_{\leq k}(t) - P^y_{\leq k}(t)| \leq 2^{k+3}.$$

A.4 Proof of Theorem 3.1

Proof. Let A be any online 1-speed algorithm. We consider the following adversary. At time 0, the adversary requests pages $1, \ldots, \frac{n}{2}$, all which have a deadline of $\frac{n}{2}$. Between time 1 and $\frac{n}{4}$ the client requests whatever page the online algorithm A broadcasts immediately after that request is broadcast; this new request also has a deadline of $\frac{n}{2}$. It follows that at time $t = \frac{n}{2}$ the online algorithm A has $\frac{n}{4}$ requests for distinct pages in its queue. However, the adversary can finish all these requests by time $\frac{n}{2}$. Then starting at time $\frac{n}{2}$ the adversary requests $\frac{n}{2}$ new pages, say $\frac{n}{2} + 1, \ldots, n$. These new pages are requested, one at each time step, in a cyclic fashion for n^2 cycles. More formally, for $i = 1, \ldots, n/2$, page $\frac{n}{2} + i$ is requested at times $j \cdot (\frac{n}{2}) + i - 1$ for $j = 1, \ldots, n$. Each of these requests has a slack of one which means that their deadline is one unit after their arrival. The adversary can satisfy these requests with delay since it has no queue at any time; thus its maximum delay factor is 1. However, the online algorithm A has $\frac{n}{4}$ requests in its queue at time $\frac{n}{2}$; each of these has a slack of $\frac{n}{2}$. We now argue that the delay factor of A is $\Omega(n)$. If the algorithm satisfies two slack 1 requests for the same page by a single transmission, then its delay factor is $n/2$; this follows since the requests for the same page are $n/2$ time units apart. Otherwise, the algorithm does not merge any requests for the same page and hence finishes the the last request by time $n/2 + n^2/2 + n/4$. If the last request to be finished is a slack 1 request, then its delay factor is at least $n/4$ since the last slack 1 requests is released at time $n/2 + n^2/2$. If the last request to be finished is one of the requests with slack $n/2$, then its delay factor is at least $n^2/2/(n/2) = \Omega(n)$.