THEORY OF COMPUTING, Volume 8 (2012), pp. 165–195
www.theoryofcomputing.org

Online Scheduling to Minimize Maximum Response Time and Maximum Delay Factor*

Chandra Chekuri† Sungjin Im‡ Benjamin Moseley§

Received: July 28, 2010; published: May 14, 2012.

Abstract: This paper presents several online scheduling algorithms for two related performance metrics, namely maximum response time and maximum delay-factor, and also their weighted versions. The delay factor metric is new (introduced in Chang et al. (SODA'08)), while special cases of maximum weighted response time have been considered before. We study both the standard scheduling model where each arriving job requires its own processing, as well as the broadcast scheduling model where multiple requests/jobs can be simultaneously satisfied.

Motivated by strong lower bounds, we consider the resource augmentation model introduced in Kalyanasundaram and Pruhs (JACM'95) where the algorithm is given a machine with faster speed than the adversary. We give scalable algorithms; that is, algorithms that when given $(1+\varepsilon)$-speed are $O(\mathrm{poly}(1/\varepsilon))$-competitive for any fixed $\varepsilon > 0$. Our main contributions are for broadcast scheduling. Along the way we also show that the FIFO (first-in-first-out) algorithm is 2-competitive for broadcast scheduling even when pages have non-uniform sizes. We complement our algorithmic results by showing that a natural greedy

*This paper contains and extends results that appeared in two preliminary papers [23, 22].

†Partially supported by NSF grants CCF-0728782 and CNS-0721899.
‡Supported mainly by a Samsung Fellowship and partially by NSF grant CNS-0721899.
§Partially supported by NSF grant CNS-0721899.

ACM Classification: C.4, F.2.2

AMS Classification: 68M20, 68Q25, 68W25, 68W40, 90B35

Key words and phrases: online scheduling, resource augmentation, maximum delay factor, maximum weighted response time, broadcast scheduling

 DOI: 10.4086/toc.2012.v008a007

algorithm modeled after LWF (Longest-Wait-First) is, for any $s \geq 1$, not constant competitive for minimizing maximum delay factor when given an s-speed machine. The lower bound holds even in the restricted setting of standard scheduling of jobs with uniform size, and demonstrates the importance of the trade-off made in our algorithms.

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 paper we consider two related performance metrics, namely response time (also referred to as flowtime) and a recently suggested performance metric called *delay factor* [17]. We also address their weighted versions. In particular, we are interested in scheduling to minimize the maximum (weighted) response time (over all requests) or to minimize the maximum delay factor. We consider both the traditional setting where requests are independent and require separate processing from the machine and also the more recent setting of broadcast scheduling where 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*. A request J_i and its properties are known only when it arrives to the system. We use a_i to denote its arrival time and ℓ_i to denote its processing time. In the weighted case J_i has a non-negative weight w_i; the unweighted case corresponds to $w_i = 1$ for all i. Consider an online scheduling algorithm A. Let f_i^A denote the completion time or finish time of J_i under A. The response time of J_i under A is $f_i^A - a_i$, in other words the total duration spent by J_i in the system before its processing is finished. Now we define the delay factor metric. Here it is assumed that each request J_i has a *deadline* d_i that is known upon its arrival. We refer to the quantity $S_i = (d_i - a_i)$ as the *slack* of request J_i. If f_i^A is the finish time of J_i under some schedule A then its delay factor is defined as $\max\{1, (f_i^A - a_i)/(d_i - a_i)\}$. Thus, delay factor measures the ratio of the response time and the slack, unless the request finishes by its deadline in which case we set its delay factor to 1. In this paper we are interested in algorithms that minimize the maximum response time and the maximum delay factor. Given an online request sequence σ and an algorithm A, let $\alpha^A(\sigma) = \max_{J_i \in \sigma} \gamma_i^A$ where γ_i^A is either the response time or the delay factor of J_i in A; in the weighted case $\alpha^A(\sigma) = \max_{J_i \in \sigma} w_i \gamma_i^A$. We measure the performance of an algorithm A via worst-case competitive analysis: A is r-competitive if for all request sequences σ we have $\alpha^A(\sigma) \leq r\alpha^*(\sigma)$ where $\alpha^*(\sigma)$ is the value of an optimum offline schedule for σ.

Now we discuss broadcast scheduling where multiple requests can be satisfied simultaneously. This model is motivated by applications in wireless and local area networks where information is transmitted over a broadcast medium [8, 2, 1, 10]; this allows all clients interested in a particular piece of information to access it at the same time. The model is also motivated by batch scheduling problems [27, 26, 45, 9] and more recent applications [25, 14]. In the formal model, there are n distinct pages or pieces of data that are available in the system, and a request from a client specifies a page of interest. This is called the *pull-model* since the clients initiate the request and we focus on this model in this paper (in the

[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.

push-model the server transmits the pages according to some frequency). Multiple outstanding requests for the same page p are satisfied by a single transmission of page p. We use $J_{p,i}$ to denote i'th request for a page $p \in \{1,2,\ldots,n\}$; here we assume that the requests for the same page are ordered by arrival time with ties broken arbitrarily. We let $a_{p,i}$ denote the arrival time of the request $J_{p,i}$. Requests may also have deadlines, denoted $d_{p,i}$ and non-negative weights $w_{p,i}$. The finish time $f^A_{p,i}$ of a request $J_{p,i}$ in a given schedule A is defined to be the earliest time after $a_{p,i}$ when the page p is transmitted by the schedule A. Note that multiple requests for the same page can have the same finish time. The response time and delay factor for a request $J_{p,i}$ are defined as $(f^A_{p,i} - a_{p,i})$ and $\max\{1,(f^A_{p,i} - a_{p,i})/(d_{p,i} - a_{p,i})\}$ respectively. As before, given an online request sequence σ and an algorithm A, we let $\alpha^A(\sigma) = \max_{J_{p,i}\in\sigma} \gamma^A_{p,i}$ where $\gamma^A_{p,i}$ is either the response time or the delay factor of $J_{p,i}$ in A; in the weighted case $\alpha^A(\sigma) = \max_{J_i\in\sigma} w_{p,i}\gamma^A_{p,i}$. Here too we are interested in competitive analysis. A significant portion of the broadcast scheduling literature focused on the case where all page sizes are the same which can be assumed to be one (unit) without loss of generality. We call this the uniform page size setting. We also consider the *non-uniform* page size setting where pages have potentially different sizes. When page sizes are non-uniform, one has to carefully define when a request for a page is satisfied if it arrives midway through the transmission of that page. In this paper we consider the sequential model [28], the most restrictive one, where the server broadcasts each page sequentially and a client receives the page sequentially without buffering; see [43] for the relationship between different models. When pages have non-uniform sizes, each page p is divided into an ordered list of uniform sized pieces $(1,p),(2,p),\ldots,(\ell_p,p)$ where ℓ_p is the size of page p. In the sequential setting, a client must receive the pieces in sequential order.

Motivation There are a variety of metrics in the scheduling literature and perhaps the best known metric in the online setting is to minimize the average (equivalently total) response time. Minimizing average response time can potentially delay some requests at the expense of others and can lead to starvation and unfairness. Other metrics can be and are used to overcome this issue. A more stringent metric is to minimize the maximum response time and for unicast scheduling it is easy to see that the non-preemptive first-in-first-out algorithm (FIFO) is optimal on a single machine. Interestingly, FIFO was considered for minimizing maximum response time in broadcast scheduling in one of the early papers on broadcast scheduling by Bartal and Muthukrishnan [10] where it was claimed to be 2-competitive. It was only fairly recently that a formal proof was established by Chang et al. [17]. Maximum response time and FIFO do not distinguish between requests of different sizes. Bender, Chakrabarti and Muthukrishnan [11] introduced the metric of minimizing the maximum stretch where the stretch of a request is the ratio between the response time to its processing time. They reasoned that requests are sensitive to delay in proportion to their processing time, and were also motivated by scheduling applications in databases and web servers; see [13, 37, 44] for further discussion.

The metrics we consider, in addition to their inherent interest, also generalize maximum stretch in different ways. One can easily see that stretch is a special case of weighted response time where the weight is the inverse of the processing time. Weights allow more flexibility in assigning priority to requests. Surprisingly, this natural generalization has not received attention in the literature. Delay factor is motivated by a variety of applications, in particular real-time systems, where requests naturally have deadlines associated with them. 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 [15], or by focusing on maximizing throughput (the number of requests completed by their deadline) [39, 16, 46]. It was recently suggested by Chang et al. [17] that delay factor is a useful and natural relaxation to consider in situations with soft deadlines where we still desire all requests to be satisfied. Moreover, we note that if we set $d_i = a_i + \ell_i$ (in the unicast setting) the delay factor of a request is identical to its stretch in any schedule.

1.1 Results

We give the first non-trivial results for online scheduling to minimize the maximum delay factor and weighted response time in both the unicast and broadcast settings. Throughout we assume that requests are allowed to be preempted without any penalty. We remark that weighted response time and delay factor, though formally not equivalent, behave similarly in terms of algorithmic development. At a heuristic level, one can interpret the term $1/(d_i - a_i)$ in the delay factor metric as the weight w_i. For this reason, we mainly discuss results for delay factor below and point out how they generalize to weighted response time.

We first prove strong lower bounds on online competitiveness for delay factor.

- 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 uniform sized pages there is no $n/4$-competitive algorithm.

We resort to resource augmentation analysis, introduced by of Kalyanasundaram and Pruhs [35], to overcome the above lower bounds. In this analysis the online algorithm is given a faster machine than the optimal offline algorithm. For $s \geq 1$, an algorithm A is said to be s-speed r-competitive if A, when given s-speed machine(s), achieves a competitive ratio of r compared to the optimum schedule given 1-speed machine(s). We prove the following.

- For unicast setting, for any $\varepsilon \in (0,1]$, there are $(1+\varepsilon)$-speed $O(1/\varepsilon)$-competitive algorithms 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. An algorithm is non-migratory if it processes each request on a single machine to which it is first assigned.
- For broadcast setting, for any $\varepsilon \in (0,1]$, there is a $(1+\varepsilon)$-speed $O(1/\varepsilon^2)$-competitive algorithm for uniform pages. For non-uniform sized pages, for any $\varepsilon \in (0,1]$, there is a $(1+\varepsilon)$-speed $O(1/\varepsilon^3)$-competitive algorithm.

Our results for minimizing maximum delay factor can be easily extended to the problems of minimizing maximum weighted response time and minimizing maximum weighted delay factor. We also address the problem of minimizing the maximum response time in broadcast scheduling. We already mentioned that FIFO is 2-competitive when pages have uniform sizes [17]. In this paper we show the following.

- FIFO is 2-competitive for minimizing the maximum response time for non-uniform sized pages in the broadcast model.

The above result was claimed in [10] but it was only in [17] that a formal proof was given for uniform sized pages with integer arrival times for all requests. The proof in [17] is short but delicate and it does not appear to generalize when page sizes are non-uniform. Our proof differs from the one in [17] and is inspired by our analysis for the delay factor metric. A competitive ratio of 2 is the best positive result that can be achieved; for any fixed $\varepsilon > 0$ there is a lower bound of $(2-\varepsilon)$ on the competitive ratio for minimizing the maximum response time, even if randomization is allowed [19].

Our final result is a lower bound on the performance of a simple greedy algorithm to minimize the maximum delay factor. Recall that minimizing the maximum delay factor metric is a generalization of the problem of minimizing the maximum response time. For the latter FIFO is optimal for unicast scheduling and is 2-competitive for broadcast scheduling. What are natural ways to generalize FIFO to minimize maximum delay factor and weighted response time? One natural algorithm that extends FIFO is to schedule the request in the queue that has the largest current delay factor (or current weighted response time). We call this greedy algorithm LF (longest first) since it can be seen as an extension of the well-studied algorithm LWF (longest-wait-first) for minimizing average response time. It is known that LWF is $O(1)$-competitive with $O(1)$-speed for average response time [29]. In [21] we showed that LF is $O(1)$-speed $O(1)$-competitive also for L_k norms of response time for small values of k. It is therefore natural to ask if LF is $O(1)$-speed $O(1)$-competitive for minimizing the maximum weighted response time and delay factor. We show that this is not the case even for unicast scheduling.

- For any constants $s, c > 1$, LF is not c-competitive with s-speed for minimizing maximum delay factor (or weighted response time) in unicast scheduling of uniform sized requests.

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 [7] in the single machine and parallel machine settings [3, 20]. Our main results are for broadcast scheduling. Broadcast scheduling has posed considerable difficulties for algorithm design. Much previous work has focused on the *offline* setting [36, 31, 32, 33, 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 [39, 16, 46, 24] get around this by eliminating requests. Few positive results are known in the online setting where all requests need to be scheduled [10, 28, 29, 30] and the analysis in all of these is quite non-trivial.

Our algorithm and analysis are direct and explicitly demonstrate the value of making requests wait for some duration to take advantage of potential future requests for the same page. We hope this idea can be further exploited in broadcast scheduling. We mention that prior to our work, even in the offline setting, the only algorithm known for minimizing the maximum delay factor was a 2-speed optimal algorithm that was based on rounding a linear-programming relaxation [17]. Our algorithm, when viewed as an offline algorithm, gives a $(1+\varepsilon)$-speed $O(1/\varepsilon^2)$-approximation for uniform page sizes (and $O(1/\varepsilon^3)$-approximation for non-uniform page sizes) and is quite simple.

1.2 Other related work

We refer the reader to the survey on online scheduling by Pruhs, Sgall and Torng [42] for a comprehensive overview of results and algorithms (see also [41]). For requests with deadlines, the well-known earliest-deadline-first (EDF) algorithm can be used in the offline setting to check if all the requests 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 request sequence that allow for schedulability of requests 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 [40] for the extensive literature on scheduling issues in real-time systems.

Closely related to our work is that on minimizing the maximum stretch [11] where it is shown that no online algorithm is $O(P^{0.3})$ competitive where P is the ratio of the largest job size to the smallest job size. [11] also gives an $O(\sqrt{P})$ competitive algorithm which was further refined in [13]. Resource augmentation analysis for L_p norms of response time and stretch from the work of Bansal and Pruhs [7] implicitly shows that the shortest job first (SJF) algorithm is a $(1+\varepsilon)$-speed $O(1/\varepsilon)$-competitive algorithm for minimizing the maximum 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, 20]. In [12], the authors suggest the delay factor metric as a performance measure under the name of maximum interval stretch. They consider a model where the processing speed a request receives increases as the request 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 requests 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 [18, 38], the recent paper of Chang et al. [17], and the surveys [42, 41] for several pointers to known results. Our work on delay factor is inspired by [17]. As we mentioned already, a large amount of the work on broadcast scheduling has been on offline algorithms including NP-hardness results and approximation algorithms (often with resource augmentation). For minimizing the maximum delay factor there is a 2-speed optimal algorithm in the *offline* setting and it is also known that unless $\mathsf{P} = \mathsf{NP}$ there is no $(2-\varepsilon)$ approximation [17]. Besides the works already mentioned, in the online setting the following results are currently known. For minimizing the average response time, several algorithms that are $O(1)$-competitive with $O(1)$-speed were developed starting with the work of Edmonds and Pruhs [28]. This work has recently culminated in scalable algorithms that are $O(1)$-competitive with $(1+\varepsilon)$-speed for any fixed $\varepsilon > 0$ [34, 6]. Constant competitive online algorithms were given for maximizing throughput [39, 16, 46, 24] when pages have uniform sizes.

Notation We let $S_i = d_i - a_i$ denote the slack of J_i in the unicast setting. When requests have non-uniform 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, $J_{p,i}$ denotes the i'th request for page p. We assume that the requests for a page are ordered by arrival 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. When pages have non-uniform sizes, ℓ_p denotes the size of page p. For an interval $I = [a,b]$ on the real line we use $|I|$ for the length of the interval $(b-a)$. We say a request is alive at t if has arrived by t but has not yet finished.

Organization We discuss algorithms for unicast scheduling in Section 2. Our main results on broadcast scheduling are presented in Section 3. We mainly describe algorithms for minimizing the maximum delay factor. These can be extended relatively easily for weighted delay factor and weighted response time.

We give details of this extension only for broadcast scheduling, in Section 4. The lower bound for LF is described in Section 5.

2 Unicast scheduling

In this section we address the unicast setting where requests are independent and consider the delay factor metric. For a request J_i, recall that a_i, d_i, ℓ_i, f_i denote the arrival time, deadline, processing time/size, and finish time, respectively. An instance with all $\ell_i = 1$ (or more generally the processing times are uniform) is referred to as a uniform processing time instance. It is easy to see that preemption does not help much for uniform processing time instances when the maximum delay factor metric is considered. Assuming that the processing times are integer valued, in the single machine setting one can reduce an instance with non-uniform processing times to an instance with uniform processing times as follows: replace J_i, with processing time ℓ_i by ℓ_i uniform sized requests with the same arrival time and deadline as that of J_i.

We remarked earlier that scheduling to minimize the maximum stretch is a special case of scheduling to minimize the maximum delay factor. In [11] a lower bound of $P^{1/3}$ is shown for minimizing the maximum stretch where P is the ratio of the maximum processing time to the minimum processing time. They show that this lower bound 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 minimizing maximum stretch to $P^{0.4}/2$ when the online algorithm is not aware of P. The lower bound example is similar to that given in [11].

Theorem 2.1. *Any* 1*-speed deterministic algorithm has competitive ratio at least $P^{0.4}/2$ for minimizing the maximum stretch when P is not known in advance to the algorithm.*

Proof. P will denote the ratio of the maximum to minimum processing time. For the example created, P will depend on the decisions the online algorithm makes. For the sake of contradiction, assume that some online algorithm $\mathcal{A}$ achieves a competitive ratio better than $P^{0.4}/2$. Now consider the following example, where L is chosen so that the parameters in the following example are integral.

`Type 1`: At time 0 a request with processing time L arrives.

`Type 2`: At times $L - L^{0.6}, L, L + L^{0.6}, \cdots, L^{1.16} - L^{0.6}$, that is at times $L - L^{0.6} + L^{0.6} \cdot i$ for integral $i \in [0, L^{0.56} - L^{0.4}]$, let a request with processing time $L^{0.6}$ arrive.

Consider time $L^{1.16}$. If this is the entire sequence of requests, then the optimal schedule can be described as follows. It schedules these requests in a first-in-first-out fashion. The optimal schedule finishes the request of type 1 by time L, and hence the stretch of this request is 1. The requests of type 2 finish $2L^{0.6}$ time units after their arrival. Thus, the maximum stretch of any request in the optimal schedule is $2L^{0.6}/L^{0.6} = 2$.

The ratio of maximum to minimum processing time of the requests seen so far is $P = L/L^{0.6} = L^{0.4}$. Thus, the maximum stretch algorithm $\mathcal{A}$ can have is $(L^{0.4})^{0.4}/2 = L^{0.16}/2$, by assumption. Suppose $\mathcal{A}$ does not finish the request of type 1 by time $L^{1.16}$. In this case, the stretch of this request in the algorithm's schedule will be at least $L^{1.16}/L = L^{0.16}$. Thus, the algorithm has a competitive ratio at least $L^{0.16}/2 = P^{0.4}/2$, a contradiction. Therefore, by time $L^{1.16}$ the algorithm $\mathcal{A}$ must have finished the request of type 1. Immediately after this time, requests of type 3 arrive.

`Type 3`: Starting at time $L^{1.16}$ a set of $L^{1.2}-L^{0.6}$ unit processing time requests arrive as follows. These requests arrive one after another; one such request arrives at each integer time step during $[L^{1.16}, L^{1.16}+L^{1.2}-L^{0.6}]$.

This is the entire request sequence. We now analyze the behavior of an optimum schedule for this sequence and compare it to that of $\mathcal{A}$. An optimal schedule for this sequence schedules the request of type 1 from time 0 until time $L-L^{0.6}$. At time $L-L^{0.6}$, the type 1 request has $L^{0.6}$ remaining processing time left; the optimal solution schedules requests of type 2 and type 3 requests as they arrive. It is easy to verify that the stretch of requests of type 2 and 3 is 1 in this schedule. At time $L^{1.16}+L^{1.2}-L^{0.6}$ the optimum schedule finishes the request of type 1. Thus the maximum stretch of this schedule is for the type 1 request and is equal to $(L^{1.16}+L^{1.2})/L \leq 2L^{0.2}$.

As we argued earlier, $\mathcal{A}$ must have completely scheduled the request of type 1 by time $L^{1.16}$. Thus the last request it finishes is either of type 2 or type 3. $\mathcal{A}$ has a volume of at least $L^{0.6}$ of type 2 requests left to complete at time $L^{1.16}$. If the last request completed by $\mathcal{A}$ is of type 2 then this request must have waited for all requests of type 3 to finish. Since the arrival time of a type 2 request is at most $L^{1.16}-L^{0.6}$ and the request is completed by time $L^{1.16}+L^{1.2}-L^{0.6}$ at the earliest (this time is the latest arrival of a type 3 request), the total time this request waits to be satisfied is $L^{1.16}+L^{1.2}-L^{0.6}-(L^{1.16}-L^{0.6})=L^{1.2}$. Thus the stretch of this request is at least $L^{1.2}/L^{0.6}=L^{0.6}$. If the last request satisfied by the algorithm is of type 3, then this request must have waited for $L^{0.6}$ time since a $L^{0.6}$ volume of processing time of type 2 requests remained in the algorithm's queue when type 3 requests began arriving. The stretch of this request is therefore at least $L^{0.6}$. Notice that the ratio of maximum to minimum processing time is $P=L$. In either case, the competitive ratio of the algorithm is at least $L^{0.6}/2L^{0.2}=L^{0.4}/2=P^{0.4}/2$, a contradiction. □

Recall that Δ is the ratio of the maximum slack to the minimum slack in a given sequence of requests. From the above we have the following corollary for delay factor.

Corollary 2.2. *Any deterministic online algorithm has competitive ratio at least $\Delta^{0.4}/2$ for minimizing the maximum delay factor when requests have uniform sizes and Δ is not known in advance to the algorithm.*

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

2.1 Single machine scheduling

In this section we consider a simple greedy algorithm for minimizing the maximum delay factor on a single machine when requests have non-uniform sizes. We analyze the simple shortest-slack-first (SSF) algorithm which at any time t schedules the request with the shortest slack. Recall that the slack of a request J_i is d_i-a_i and that we have assumed without loss of generality that all requests have uniform sizes.

Algorithm: SSF

- Let $Q(t)$ be the set of alive requests at t.
- Let J_i be the request with the minimum slack among requests in $Q(t)$, ties broken by arrival time.
- Preempt the current request and schedule J_i if it is not being processed.

Theorem 2.3. *The algorithm SSF is an $(1+\varepsilon)$-speed $1/\varepsilon$-competitive online algorithm for minimizing the maximum delay factor in the unicast setting.*

Proof. Consider an arbitrary request sequence σ and let α^{SSF} be the maximum delay factor achieved by SSF on σ. If $\alpha^{\mathrm{SSF}} = 1$ there is nothing to prove, so assume that $\alpha^{\mathrm{SSF}} > 1$. Let J_i be the request that witnesses α^{SSF}, that is $\alpha^{\mathrm{SSF}} = (f_i - a_i)/S_i$. 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 minimum value less than or equal to a_i such that SSF was busy processing 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 $(1+\varepsilon)(f_i - t)$ 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 $f_i + \varepsilon(f_i - t) \geq f_i + \varepsilon(f_i - a_i)$. Since all these requests have slack at most S_i and have arrived before f_i, it follows that $\alpha^* \geq \varepsilon(f_i - a_i)/S_i$ where α^* is the maximum delay factor of the optimal offline algorithm with 1-speed machine. Therefore, we have that $\alpha^{\mathrm{SSF}}/\alpha^* \leq 1/\varepsilon$. □

Remark 2.4. For uniform processing time requests, the algorithm that *non-preemptively* schedules requests with the shortest slack is a $(1+\varepsilon)$-speed $2/\varepsilon$-competitive online algorithm for minimizing the maximum delay factor.

2.2 Multiple machine scheduling

We now consider minimizing the maximum delay factor when there are m machines. In the multiple machine setting, at any time a machine can choose to process at most one request and a request can be processed by at most one machine at a given time. A request is allowed to migrate and be processed by different machines at different times; as we remarked earlier, a schedule or algorithm is non-migratory if it does not migrate any request. To adapt SSF to this setting we take intuition from previous work on minimizing L_k norms of flow time and stretch [7, 3, 20]. 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 assigned to machine x by time t of class k. 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 2.5. *SSF-ID is a $(1+\varepsilon)$-speed $O(1/\varepsilon)$-competitive online algorithm for minimizing the maximum delay factor on m identical 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 $(1+\varepsilon)$, 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 for any k the total volume of requests of class at most k that is assigned is balanced across the machines. This easily follows from the dispatching policy of the algorithm. Several of the lemmas below and their proofs follow from the work in [3, 20].

Proposition 2.6. *For any time t and two machines x and y, $|U^x_{=k}(t) - U^y_{=k}(t)| \leq 2^{k+1}$. This also implies that $|U^x_{\leq k}(t) - U^y_{\leq k}(t)| \leq 2^{k+2}$.*

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 from the first. □

Lemma 2.7. *Consider any two machines x and y. At any time t, the difference in volume of requests that already have been processed is bounded as $|P^x_{\leq k}(t) - P^y_{\leq k}(t)| \leq 2^{k+2}$.*

Proof. Suppose the lemma is false. Then there is the 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 $P^x_{\le k}(t') \le U^x_{\le k}(t')$. However, this implies that

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

a contradiction to Proposition 2.6. □

Lemma 2.8. *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 $|R^x_{\le k}(t) - R^y_{\le k}(t)| \le 2^{k+3}$.*

Proof. Combining Proposition 2.6, Lemma 2.7, and the fact that $R^x_{\le k}(t) = U^x_{\le k}(t) - P^x_{\le k}(t)$ for any x and k by definition then,

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

□

Corollary 2.9. *At any time t, $V^*_{\le k}(t) \ge V_{\le k}(t) - m2^{k+3}$.*

We are now ready to upper bound the competitiveness of SSF-ID when given $(1+\varepsilon)$-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 maximum delay factor $\alpha^{\text{SSF-ID}}$ of SSF-ID. Let k be the class of J_i and let x be the machine J_i was processed on by SSF-ID. We know that $\alpha^{\text{SSF-ID}} = (f_i - a_i)/S_i$ by definition of J_i. We use α^* to denote the delay factor of some fixed optimal 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 $> k$ under SSF-ID's schedule. Note that $t \le a_i$ since x does not process any request of class $> k$ in the interval $[a_i, f_i]$. At time t we know by Corollary 2.9 that $V^*_{\le k}(t) \ge V_{\le k}(t) - m2^{k+3}$. If $f_i \le a_i + 2^{k+4}$ then SSF-ID achieves a competitive ratio of 16 since J_i is in class k. Thus we will assume from now on that $f_i > a_i + 2^{k+4}$.

During the interval $I = [t, f_i)$, SSF-ID completes a total volume of $(1+\varepsilon)(f_i - t)$ work on machine x. Using Lemma 2.7, any other machine y also processes a volume of $(1+\varepsilon)(f_i - t) - 2^{k+3}$ work during I of requests in classes at most k . Thus the total volume processed by SSF-ID during I of requests in classes at most k is at least $m(1+\varepsilon)(f_i - t) - m2^{k+3}$. During I, the optimal algorithm schedules at most a $m(f_i - t)$ volume of work for requests in classes at most k. Combining this with Corollary 2.9, we see that

$$\begin{aligned} V^*_{\le k}(f_i) &\ge V_{\le k}(t) - m2^{k+3} + m(1+\varepsilon)(f_i - t) - m2^{k+3} \\ &\ge V_{\le k}(t) + m(1+\varepsilon)(f_i - t) - m2^{k+4} \ge \varepsilon m(f_i - t)\,. \end{aligned}$$

In the last inequality we use the fact that $f_i - t \ge f_i - a_i \ge 2^{k+4}$. Without loss of generality assume that no requests arrives exactly at f_i. Therefore $V^*_{\le k}(f_i)$ 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 $f_i + \varepsilon(f_i - t)$ and therefore it follows that $\alpha^* \ge \varepsilon(f_i - t)/2^{k+1}$. Since $\alpha^{\text{SSF-ID}} \le (f_i - a_i)/2^k$ and $t \le a_i$, it follows that $\alpha^{\text{SSF-ID}} \le 2\alpha^*/\varepsilon$.

Thus $\alpha^{\text{SSF-ID}} \le \max\{16, 2\alpha^*/\varepsilon\}$ which completes the proof of Theorem 2.5.

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 uniform sizes which is assumed to be unit. Here we consider both the case where pages have uniform and non-uniform sizes. We start by focusing on minimizing the maximum response time of a schedule and then shift our focus to minimizing the maximum delay factor.

3.1 Response time

In this section we analyze FIFO for minimizing maximum response time when page sizes are non-uniform. As mentioned previously, it is known that FIFO is 2-competitive when pages have uniform sizes [17]. We first describe the algorithm FIFO. FIFO broadcasts pages *non-preemptively*; the optimal solution is allowed to use preemption. Consider a time t when FIFO finished broadcasting a page or a request arrives when FIFO has no unsatisfied requests just before time t. Let $J_{p,i}$ be the request in FIFO's queue with earliest arrival time breaking ties arbitrarily. FIFO begins broadcasting page p at time t. At any time during this broadcast, we will say that $J_{p,i}$ *forced* FIFO to broadcast page p at this time. When broadcasting a page p all requests for page p that arrived before or at the start of the broadcast are simultaneously satisfied when the broadcast completes. Any request for page p that arrives during the broadcast are not satisfied until the next full transmission of p. Recall that we are assuming the sequential model where the client does not buffer. The rest of this section will be devoted to proving the following theorem.

Theorem 3.1. *FIFO is a* 2*-competitive online algorithm for minimizing the maximum response time in broadcast scheduling when pages have non-uniform sizes.*

We do not assume speed augmentation when analyzing FIFO. Let σ be an arbitrary sequence of requests. Let OPT denote some fixed optimum schedule and let ρ^* denote the optimum maximum response time and ρ^{FIFO} denote FIFO's maximum response time. We will show that $\rho^{\text{FIFO}} \leq 2\rho^*$. For the sake of contradiction, assume that FIFO witnesses a response time $c\rho^*$ by some request $J_{q,k}$ for some $c > 2$. Let t^* be the time $J_{q,k}$ is satisfied, that is $t^* = f_{q,k}$. Let t_1 be the smallest time less than t^* such that at any time t during the interval $[t_1, t^*]$ the request which forces FIFO to broadcast a page at time t has response time at least ρ^* when satisfied. Note that $t_1 \leq t^* - \ell_q$ where ℓ_q is the length of page q. We let I denote the interval $[t_1, t^*]$. Let $\mathfrak{J}_I$ denote the requests which forced FIFO to broadcast during I. Notice that during the interval I, all requests in $\mathfrak{J}_I$ are completely satisfied during this interval. In other words, any request in $\mathfrak{J}_I$ starts being satisfied during I and is finished during I.

We say that OPT *merges* two distinct requests for a page p if they are satisfied by the same broadcast.

Lemma 3.2. OPT *cannot merge any two requests in $\mathfrak{J}_I$ into a single broadcast.*

Proof. Let $J_{p,i}, J_{p,j} \in \mathfrak{J}_I$ such that $i < j$. Note that $J_{p,i}$ is satisfied before $J_{p,j}$. Let t' be the time that FIFO *starts* satisfying request $J_{p,i}$. By the definition of I, request $J_{p,i}$ has response time at least ρ^*. The request $J_{p,j}$ must arrive after time t', that is $a_{p,j} > t'$, otherwise request $J_{p,j}$ is satisfied by the same broadcast of page p that satisfied $J_{p,i}$. Therefore, it follows that if OPT merges $J_{p,i}$ and $J_{p,j}$ then the finish time of $J_{p,i}$

in OPT is strictly greater than its finish time in FIFO which is already at least ρ^*; this is a contradiction to the definition of ρ^*. □

Lemma 3.3. *All requests in $\mathfrak{J}_I$ arrived no earlier than time $t_1 - \rho^*$.*

Proof. For the sake of contradiction, suppose some request $J_{p,i} \in \mathfrak{J}_I$ arrived at time $a_{p,i} < t_1 - \rho^*$. During the interval $[a_{p,i} + \rho^*, t_1]$ the request $J_{p,i}$ must have wait time at least ρ^*. However, then any request which forces FIFO to broadcast during $[a_{p,i} + \rho^*, t_1]$ must have response time at least ρ^*, contradicting the definition of t_1. □

We are now ready to prove Theorem 3.1, stating that FIFO is 2-competitive.

Proof. Recall that all requests in $\mathfrak{J}_I$ are completely satisfied during I. Thus we have that the total size of requests in $\mathfrak{J}_I$ is $|I|$. By definition $J_{q,k}$ witnesses a response time greater than $2\rho^*$ and therefore $t^* - a_{q,k} > 2\rho^*$. Since $J_{q,k} \in \mathfrak{J}_I$ is the last request done by FIFO during I, all requests in $\mathfrak{J}_I$ must arrive no later than $a_{q,k}$. Therefore, these requests must be finished by time $a_{q,k} + \rho^*$ by the optimal solution. From Lemma 3.3, all the requests $\mathfrak{J}_I$ arrived no earlier than $t_1 - \rho^*$. Thus OPT must finish all requests in $\mathfrak{J}_I$, whose total volume is $|I|$, during $I_{\text{OPT}} = [t_1 - \rho^*, a_{q,k} + \rho^*]$. Thus it follows that $|I| \le |[t_1 - \rho^*, a_{q,k} + \rho^*]|$, which simplifies to $t^* \le a_{q,k} + 2\rho^*$. This is a contradiction to the assumption that $t^* - a_{q,k} > 2\rho^*$. □

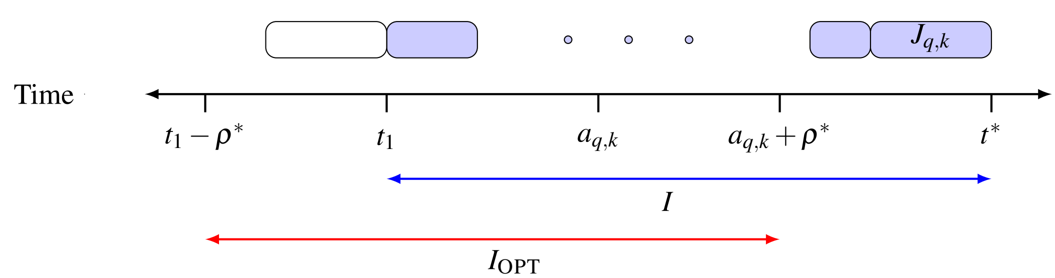

Figure 1: Broadcasts by FIFO satisfying requests in $\mathfrak{J}_I$ are shown in blue. Note that $a_{q,k}$ and $a_{q,k} + \rho^*$ are not necessarily contained in I.

We now discuss the differences between our proof of FIFO for non-uniform sized pages and the proof given by Chang et al. in [17] showing that FIFO is 2-competitive for uniform sized pages. In [17] it is shown that at anytime t, $F(t)$, the set of *unique* pages in FIFO's queue satisfies the following property: $|F(t) \setminus O(t)| \le |O(t)|$ where $O(t)$ is the set of unique pages in OPT's queue. This easily implies the desired bound. To establish this, they use a slot model in which uniform sized pages arrive only during integer times which allows one to define unique pages. This may appear to be a technicality, however when considering non-uniform sized pages, it is not clear how one defines unique pages since this number varies during the transmission of p as requests accumulate. Our approach avoids this issue in a clean manner by not assuming a slot model or uniform sized pages.

3.2 Delay factor

In this section we consider minimizing the maximum delay factor in the broadcast setting. We start by showing that no 1-speed online algorithm can be $(n/4)$-competitive for minimizing the maximum delay factor where n is the number of pages. We then prove the following theorem

Theorem 3.4. *There is an online algorithm that is $(1+\varepsilon)$-speed $O(1/\varepsilon^2)$-competitive for minimizing the maximum weighted delay factor in the broadcast setting where pages have uniform size. For non-uniform sized pages there is a $(1+\varepsilon)$-speed $O(1/\varepsilon^3)$-competitive online algorithm.*

In Section 3.2.1 we show a $(1+\varepsilon)$-speed $O(1/\varepsilon^2)$-competitive online algorithm for uniform sized pages and uniform weight requests. Finally, we extend our algorithm and analysis to the case of non-uniform page sizes and uniform weight requests to obtain a $(1+\varepsilon)$-speed $O(1/\varepsilon^3)$-competitive online algorithm in Section 3.2.2. In Section 4 we show how to extend these results to the case where requests have non-uniform weights.

Theorem 3.5. *Every* 1*-speed online deterministic algorithm for broadcast scheduling to minimize the maximum delay factor has a competitive ratio of at least $n/4$ where n is the number of pages even when pages have uniform sizes.*

Proof. The lower bound instance we construct is inspired by a lower bound given in [17]. Let $\mathcal{A}$ be any online 1-speed algorithm and let n be a multiple of 4. We consider the following adversary. At time 0, the adversary requests pages $1,\ldots,n/2$, all which have a deadline of $n/2$. Between time 1 and $n/4$ the adversary requests whatever page the online algorithm $\mathcal{A}$ broadcasts immediately after that request is broadcast; this new request also has a deadline of $n/2$. It follows that at time $t = n/2$ the online algorithm $\mathcal{A}$ has $n/4$ requests for distinct pages in its queue. However, the adversary can finish all these requests by time $n/2$. Then starting at time $n/2$ the adversary requests $n/2$ new pages, say $(n/2)+1,\ldots,n$. 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 $(n/2)+i$ is requested at times $j\cdot(n/2)+i-1$ 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 $\mathcal{A}$ has $n/4$ requests in its queue at time $n/2$; each of these has a slack of $n/2$. We now argue that the delay factor of $\mathcal{A}$ is at least $n/4$. 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 $(n^2/2)/(n/2) = n$. □

3.2.1 Uniform sized pages

In this section we develop an online algorithm, for uniform sized pages, that is competitive given extra speed. Simple examples show that natural generalizations of the algorithm SSF to the broadcast setting fail to be constant competitive with any constant resource augmentation. The reason for this is that an algorithm that focuses on requests with the smallest slack can be made to do an arbitrary amount of "extra" work over the optimal schedule by repeatedly requesting the same page and giving the page small slack. The algorithm will repeatedly broadcast the same page while the adversary waits and satisfies multiple requests for this page with a single transmission. Further, we will show in Section 5 that another greedy algorithm modeled after the well-studied algorithm Longest-Wait-First is not constant competitive with any fixed constant resource augmentation.

We begin by developing a variant of SSF that *adaptively* introduces a 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 uniform sized requests, preemption is not very helpful. At each time, the algorithm computes the delay factor of all requests in the algorithm's queue, and considers requests for scheduling only when their delay factor is comparable to the request with maximum delay factor among the unsatisfied requests. Since our algorithm SSF schedules only the requests that have waited sufficiently long, the adversary cannot delay those requests to satisfy them with a less number of transmissions. We formalize this intuition in Lemma 3.6.

Algorithm: SSF-W

- The algorithm is non-preemptive. Let t be a time that the machine is free (either because a request has just finished or there are no requests to process).
- Let $Q(t)$ be the set of alive requests at time t and let
$$\alpha_t = \max_{J_{p,i} \in Q(t)} \max\left\{1, \frac{t - a_{p,i}}{S_{p,i}}\right\}$$
be the maximum current delay factor of requests in $Q(t)$.
- Let
$$Q'(t) = \left\{J_{p,i} \in Q(t) \,\middle|\, \frac{t - a_{p,i}}{S_{p,i}} \geq \frac{1}{c}\alpha_t\right\}$$
be the set of requests in $Q(t)$ with current delay factor at least $\frac{1}{c}\alpha_t$.
- Let $J_{p,i}$ be the request in $Q'(t)$ with the smallest slack. Broadcast page p non-preemptively.

We analyze SSF-W when it is given a $(1+\varepsilon)$-speed machine. Let $c > 1 + 2/\varepsilon$ be the constant which parameterizes SSF-W. Let σ be an arbitrary sequence of requests. We let OPT denote some fixed offline optimum schedule and let α^* and $\alpha^{\text{SSF-W}}$ denote the maximum delay factor achieved by OPT and SSF-W, respectively. We will show that $\alpha^{\text{SSF-W}} \leq c^2\alpha^*$. For the sake of contradiction, suppose that SSF-W witnesses a delay factor greater than $c^2\alpha^*$. We consider the *first* time t^* when SSF-W has some request in its queue with delay factor $c^2\alpha^*$. Let the request $J_{q,k}$ be a request which achieves the delay factor $c^2\alpha^*$ at time t^*. Let t_1 be the smallest time less than t^* such that at each time t during the interval

$(t_1, t^*]$ if SSF-W is forced to broadcast by request $J_{p,i}$ at time t it is the case that $(t - a_{p,i})/S_{p,i} > \alpha^*$ and $S_{p,i} \le S_{q,k}$. We let $I = [t_1, t^*]$.[2]

We let $\mathfrak{J}_I$ denote the requests which forced SSF-W to broadcast during the interval $[t_1, t^*]$. We now show that any two requests in $\mathfrak{J}_I$ cannot be satisfied with a single broadcast by the optimal solution. Intuitively, the most effective way the adversary performs better than SSF-W is to merge requests of the same page into a single broadcast. Here we will show this is not possible for the requests in $\mathfrak{J}_I$.

Lemma 3.6. OPT *cannot merge any two requests in* $\mathfrak{J}_I$ *into a single broadcast.*

Proof. Let $J_{x,i}, J_{x,j} \in \mathfrak{J}_I$ such that $i < j$. Let t' be the time that SSF-W starts satisfying request $J_{x,i}$. By the definition of I, request $J_{x,i}$ must have delay factor greater than α^* at time $f_{x,i}$. We also know that the request $J_{x,j}$ must arrive after time t', otherwise request $J_{x,j}$ must also be satisfied at time t'. If the optimal solution combines these requests into a single broadcast then the request $J_{x,i}$ must wait until the request $J_{x,j}$ arrives to be satisfied. However, this means that the request $J_{x,i}$ must achieve a delay factor greater than α^* by OPT, a contradiction to the definition of α^*. □

To fully exploit the advantage of speed augmentation, we need to ensure that the length of the interval I is sufficiently long.

Lemma 3.7. $|I| = |[t_1, t^*]| \ge (c^2 - c)S_{q,k}\alpha^*$.

Proof. The request $J_{q,k}$ has delay factor greater than $c\alpha^*$ at any time during $I' = [t', t^*]$, where $t' = t^* - (c^2 - c)S_{q,k}\alpha^*$. Let $\tau \in I'$. The largest delay factor any request can have at time τ is less than $c^2\alpha^*$ by definition of t^* being the first time SSF-W witnesses delay factor $c^2\alpha^*$. Hence, $\alpha_\tau \le c^2\alpha^*$. Thus, the request $J_{q,k}$ is in the queue $Q(\tau)$ because $c\alpha^* \ge \alpha_\tau/c$. Moreover, this means that any request that forced SSF-W to broadcast during I', must have delay factor greater than α^* and since $J_{q,k} \in Q(\tau)$ for any $\tau \in I'$, the requests scheduled during I' must have slack at most $S_{q,k}$. □

We now explain a high level view of how a contradiction is found. From Lemma 3.6, we know any two requests in $\mathfrak{J}_I$ cannot be merged by OPT. Thus if we show that OPT must finish all these requests during an interval which is not long enough to include all of them, we will have a contradiction. More precisely, we will show that all requests in $\mathfrak{J}_I$ must be finished during I_{OPT} by OPT, where $I_{\text{OPT}} = [t_1 - 2S_{q,k}\alpha^* c, t^*]$. It is easy to see that all these requests already have delay factor α^* by time t^*, thus the optimal solution must finish them by time t^*. We first lower bound the arrival times of the requests in $\mathfrak{J}_I$.

Lemma 3.8. *Any request in* $\mathfrak{J}_I$ *must have arrived no earlier than* $t_1 - 2S_{q,k}\alpha^* c$.

Proof. For the sake of contradiction, suppose that some request $J_{p,i} \in \mathfrak{J}_I$ arrived at time $t' < t_1 - 2S_{q,k}\alpha^* c$. Recall that $J_{p,i}$ has a slack no bigger than $S_{q,k}$ by the definition of I. Therefore at time $t_1 - S_{q,k}\alpha^* c$, $J_{p,i}$ has a delay factor greater than $c\alpha^*$. Thus any request scheduled during the interval $I' = [t_1 - S_{q,k}\alpha^* c, t_1]$ has a delay factor greater than α^*. We observe that $J_{p,i}$ is in $Q(\tau)$ for $\tau \in I'$; otherwise there must be a request with a delay factor bigger than $c^2\alpha^*$ at time τ and this is a contradiction to the assumption that t^*

[2]The algorithm SSF-W was proposed in [23] and was shown to be $(2+\varepsilon)$-speed $O(1/\varepsilon^2)$-competitive. The improved analysis of the same algorithm that we present here first appeared in [22]. The weaker analysis in [23] was based on defining t_1 (implicitly) to be $a_{q,k} + c(f_{q,k} - a_{q,k})$.

is the first time that SSF-W witnessed a delay factor of $c^2\alpha^*$. Therefore any request scheduled during I' has a slack no bigger than $S_{p,i}$. Also we know that $S_{p,i} \le S_{q,k}$. In sum, we showed that any request done during I' had slack no bigger than $S_{q,k}$ and a delay factor greater than α^*, which is a contradiction to the definition of t_1. □

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

Lemma 3.9. *Suppose c is a constant s.t. $c > 1+2/\varepsilon$. If SSF-W has $(1+\varepsilon)$-speed then $\alpha^{\text{SSF-W}} \le c^2\alpha^*$.*

Proof. For the sake of contradiction, suppose that $\alpha^{\text{SSF-W}} > c^2\alpha^*$. During the interval I, the number of broadcasts which SSF-W transmits is $(1+\varepsilon)|I|$. From Lemma 3.8, all the requests processed during I have arrived no earlier than $t_1 - 2c\alpha^* S_{q,k}$. We know that the optimal solution must process these requests before time t^* because these requests have delay factor greater than α^* by t^*. By Lemma 3.6 the optimal solution must make a unique broadcast for each of these requests. Thus, the optimal solution must finish all of these requests in $2c\alpha^* S_{q,k} + |I|$ time steps. Thus, it must hold that $(1+\varepsilon)|I| \le 2c\alpha^* S_{q,k} + |I|$. Using Lemma 3.7, this simplifies to $c \le 1+2/\varepsilon$, which is a contradiction to $c > 1+2/\varepsilon$. □

The previous lemmas proves that SSF-W is a $(1+\varepsilon)$-speed $O(1/\varepsilon^2)$-competitive algorithm for minimizing the maximum delay factor in broadcast scheduling with uniform sized pages when $c = 1+3/\varepsilon$.

3.2.2 Non-uniform sized pages

Here we extend our ideas to the case where pages can have non-uniform sizes for the objective of minimizing the maximum delay factor. For this problem, in [23] we developed a generalization of SSF-W and showed that it is a $(4+\varepsilon)$-speed $O(1/\varepsilon^2)$-competitive algorithm. Later, we gave a $(2+\varepsilon)$-speed $O(1/\varepsilon^2)$-competitive algorithm using a similar proof technique as used in the uniform page size setting given in the paper [22]. In this paper we show a $(1+\varepsilon)$-speed $O(1/\varepsilon^3)$-competitive online algorithm by improving our analyses based on the technique developed by Bansal, Krishnaswamy and Nagarajan [6] which considers *fractional* schedules.

We elaborate on the model for non-uniform page sizes. Each page p has size ℓ_p, which we assume for simplicity is an integer. Page p consists of uniform sized pieces $(1,p),(2,p),\ldots(\ell_p,p)$. In a time slot one piece of a unique page can be broadcast by the server. A request $J_{p,i}$ is satisfied if it receives each of the pieces of page p in *sequential* order; in other words, we assume that the clients do not buffer the pieces if they are sent out of order. We assume preemption is allowed, and pieces of different pages can be interspersed. We call this model the integral broadcast setting. We now describe a relaxed notion of a schedule that we call a fractional schedule. In a fractional schedule the pieces of a page p are indistinguishable and the only relevant information are the time slots during which p is transmitted. Now a request $J_{p,i}$ is satisfied once ℓ_p pieces of page p have been broadcast. That is, $J_{p,i}$ is satisfied once the server devotes ℓ_p time slots to page p after $a_{p,i}$. A reduction in [6] gives a scheme that translates an algorithm with 1 speed for the fractional broadcast setting into a $(1+\varepsilon')$-speed integer schedule where the flow time, $f_i - a_i$, of each request increases by a factor of at most $8/\varepsilon'$ for any fixed $\varepsilon' > 0$. Using this technique, any schedule that is s-speed c-competitive for the fractional setting can be converted online into a schedule that is $s(1+\varepsilon')$-speed $O(c/\varepsilon')$-competitive for the integral setting. The algorithm used in [6] simulates the fractional schedule. The algorithm gives priorities to pages based on the unsatisfied requests for the page. The priority of a request is based on the flow time of the request in the fractional schedule; a smaller flow time corresponds to higher priority. Generally, the algorithm broadcasts the page with the highest priority unsatisfied request. We now restrict our attention to the fractional model.

We outline the details of modifications to SSF-W. As before, at any time t, the algorithm considers the alive requests $Q(t)$ and a subset $Q'(t)$ of requests that have waited sufficiently long; that is those with current delay factor at least α_t/c where α_t is the maximum current delay factor for requests in $Q(t)$. Among all requests in $Q'(t)$ the algorithm picks the one with the smallest slack and broadcasts a unit amount of the page for that request; ties are broken arbitrarily. Recall that since the fractional setting is considered, the algorithm only needs to specify the page being broadcast and not the piece of the page. The algorithm may preempt the broadcast of p that is forced by request $J_{p,i}$ if another request $J_{p',j}$ becomes available for scheduling such that $S_{p',j} < S_{p,i}$. A key issue that differentiates the algorithms in [23, 22] from the one here is that those algorithms directly generate an integral schedule; therefore they have to not only specify the page but also the piece of the page. In particular the algorithms in [23, 22] could preempt the transmission of a page p for a request $J_{p,i}$ and transmit p again from the start if a new request $J_{p,i'}$ for p arrives and has much smaller slack than that of $J_{p,i}$. In the sequential model this means that the work done for $J_{p,i}$ is "wasted" and it would be satisfied at the same time as $J_{p,i'}$. In the fractional setting and the algorithm considered here $J_{p,i}$ would continue to be satisfied as if $J_{p,i'}$ did not arrive and prior transmission would not be wasted.

We now analyze the algorithm assuming that it has a $(1+\varepsilon)$-speed advantage over the optimal offline algorithm. As before, let σ be an arbitrary sequence of requests. We let OPT denote some fixed offline optimum schedule and let α^* denote the optimum delay factor. Let $c > 1+3/\varepsilon$ be the constant that parameterizes SSF-W. We will show that $\alpha^{\text{SSF-W}} \le c^2\alpha^*$. For the sake of contradiction, suppose that SSF-W witnesses a delay factor greater than $c^2\alpha^*$. We consider the *first* time t^* when SSF-W has some request in its queue with delay factor $c^2\alpha^*$. Let the request $J_{q,k}$ be a request which achieves the delay factor $c^2\alpha^*$ at time t^*. Let t_1 be the smallest time less than t^* such that at each time t during the interval $(t_1,t^*]$ if SSF-W is forced to broadcast by request $J_{p,i}$ at time t it is the case that $(t-a_{p,i})/S_{p,i} > \alpha^*$ and $S_{p,i} \le S_{q,k}$. Again, let $I = [t_1,t^*]$. Notice that some requests that force SSF-W to broadcast during I could have started being satisfied before t_1. We now show a lemma analogous to Lemma 3.6. We say that two requests for the same page p are satisfied simultaneously at time t if both requests are unsatisfied prior to t, p is broadcast at time t, and both requests receive ℓ_p units of p after their arrival.

Lemma 3.10. *Consider any two distinct requests $J_{x,j}$ and $J_{x,i}$ for some page x. If $J_{x,j}$ forces SSF-W to broadcast during I before $a_{x,i}$, then* OPT *cannot satisfy $J_{x,j}$ and $J_{x,i}$ simultaneously at any time.*

Proof. Let t' be the time that SSF-W is forced to broadcast page x by $J_{x,j}$ where $t' < a_{x,i}$. By the definition of I, request $J_{x,j}$ must have delay factor greater than α^* at time t'. Hence, if OPT satisfies $J_{x,i}$ and $J_{x,j}$ simultaneously then $J_{x,j}$ will have delay factor strictly larger than α^* in OPT, a contradiction. □

The following Lemmas 3.11 and 3.12 can be proved in the same way that Lemmas 3.7 and 3.8 are proved. This is because the algorithm SSF-W is designed to be oblivious to page sizes. Indeed, the key definitions in the proofs including the first time t^* that SSF-W witnesses delay factor $c^2\alpha^*$, the request $J_{q,k}$ that has delay factor $c^2\alpha^*$ at t^*, and $t' = t^* - (c^2-c)S_{q,k}\alpha^*$ stay the same regardless of whether pages have a uniform size or not. The advantage of thinking about a fractional schedule is that the work conservation argument can be applied once we have Lemma 3.10.

Lemma 3.11. $|I| = |[t_1, t^*]| \geq (c^2 - c)S_{q,k}\alpha^*$.

Lemma 3.12. *Any request which forced SSF-W to schedule a page during I must have arrived after time $t_1 - 2S_{q,k}\alpha^* c$.*

Using the previous lemmas we can bound the competitiveness of SSF-W.

Lemma 3.13. *Suppose c is a constant s.t. $c > 1 + 2/\varepsilon$. If SSF-W has $(1+\varepsilon)$-speed then $\alpha^{\text{SSF-W}} \leq c^2\alpha^*$.*

Proof. For the sake of contradiction, suppose that $\alpha^{\text{SSF-W}} > c^2\alpha^*$. During the interval I, the number of broadcasts which SSF-W transmits is $(1+\varepsilon)|I|$. From Lemma 3.12, all the requests that forced SSF-W to broadcast during I have arrived no earlier than $t_1 - 2c\alpha^* S_{q,k}$. We know that the optimal solution must process these requests before time t^* because these requests have delay factor greater than α^* by t^*. By Lemma 3.10 the optimal solution cannot simultaneously satisfy two requests $J_{x,i}$ and $J_{x,j}$ that forced SSF-W to broadcast during I if $a_{x,i}$ is later than when $J_{x,i}$ forced SSF-W to broadcast. This implies the optimal solution must broadcast at least $(1+\varepsilon)|I|$ units in $2c\alpha^* S_{q,k} + |I|$ time steps. Thus, it must hold that $(1+\varepsilon)|I| \leq 2c\alpha^* S_{q,k} + |I|$. Using Lemma 3.11, this simplifies to $c \leq 1 + 2/\varepsilon$, which is a contradiction to $c > 1 + 2/\varepsilon$. □

By setting $c = 1 + 3/\varepsilon$ we have the following theorem.

Theorem 3.14. *The algorithm SSF-W is $(1+\varepsilon)$-speed $O(1/\varepsilon^2)$-competitive for minimizing the maximum delay factor in a fractional schedule when pages have non-uniform sizes.*

Using the reduction of [6] previously discussed, we have shown the second part of Theorem 3.4.

4 Weighted response time and weighed delay factor

We show the connection of our analysis of SSF and SSF-W to the problem of minimizing the maximum *weighted* response time. In the unicast setting for the problem of minimizing the maximum weighted response time, requests do not have deadlines, but have weights. Each request J_i has a positive weight w_i. The goal is to minimize $\max_i w_i(f_i - a_i)$. Similar to the algorithm SSF, we give the following algorithm called BWF (Biggest-Weight-First). The algorithm BWF always schedules the request with the largest weight.

Algorithm: BWF

- Let $Q(t)$ be the set of alive requests at t.
- Let J_i be the request with the largest weight among requests in $Q(t)$, ties broken by arrival time.
- Preempt the current request and schedule J_i if it is not being processed.

Similarly we can think of the problem of minimizing the maximum weighted delay factor. Here a request J_i has a deadline d_i and a weight w_i. The objective now is to minimize $\max_i w_i \max\{1,(f_i-a_i)/S_i\}$. Let the modified weight of a request J_i be defined as $w'_i = w_i/S_i$. The algorithm BWF above when implemented with modified weights is the algorithm SRF (Smallest Ratio First). It is not difficult to adapt the analysis for SSF in Section 2 to show that BWF is $(1+\varepsilon)$-speed $O(1/\varepsilon)$-competitive for the problem of minimizing the maximum weighted response time, and that SRF is $(1+\varepsilon)$-speed $O(1/\varepsilon)$-competitive for the problem of minimizing the maximum weighted delay factor.

Now consider the problem of minimizing the maximum weighted response time in broadcast scheduling when pages have uniform sizes. A request $J_{p,i}$ has a weight $w_{p,i}$. The goal is to minimize the maximum weighted response time $\max_{p,i} w_{p,i}(f_{p,i}-a_{p,i})$. We extend the algorithm BWF to get an algorithm called BWF-W for Biggest-Wait-First with Waiting. The algorithm is parameterized by a constant $c>1$. At any time t before broadcasting a page, BWF-W determines the largest weighted wait time of any request which has yet to be satisfied. Let this value be ρ_t. The algorithm then chooses to broadcast a page corresponding to the request with largest weight amongst the requests whose current weighted wait time at time t is larger than ρ_t/c.

Algorithm: BWF-W

- The algorithm is non-preemptive. Let t be a time that the machine is free (either because a request has just finished or there are no requests to process).
- Let $Q(t)$ be the set of alive requests at time t and let $\rho_t = \max_{J_{p,i}\in Q(t)} w_{p,i}(t-a_{p,i})$.
- Let $Q'(t) = \{J_{p,i} \in Q(t) \mid w_{p,i}(t-a_{p,i}) \geq \frac{1}{c}\rho_t\}$.
- Let $J_{p,i}$ be the request in $Q'(t)$ with the largest weight. Broadcast page p non-preemptively.

Although minimizing the maximum delay factor and minimizing the maximum weighted flow time are very similar metrics, the problems are not equivalent. It may also be of interest to minimize the maximum *weighted* delay factor. In this setting each request has a deadline and a weight. The goal is to minimize $\max_{p,i} w_{p,i} \max\{1,(f_{p,i}-a_{p,i})/S_{p,i}\}$. For this setting we can simply alter BWF-W where we use modified weights for requests: $w'_{p,i}$ for request $J_{p,i}$ is defined as $w_{p,i}/S_{p,i}$. We call the resulting algorithm SRF-W (Smallest-Ratio-First with Waiting).

For the problems of minimizing the maximum weighted response time and weighted delay factor, the upper bounds shown for SSF-W in this paper also hold for BWF and SRF-W, respectively. The analysis of BWF and SRF-W is very similar to that of SSF-W. To illustrate how the proofs extend, we prove that SRF-W is $(1+\varepsilon)$-speed $O(1/\varepsilon^2)$-competitive for minimizing the maximum weighted delay factor in broadcast scheduling where all pages are uniform sized and there is a single machine. This is the most general problem discussed as it is a generalization of weighted response time and the delay factor. We analyze SRF-W when it is given a $(1+\varepsilon)$-speed machine. Let $c > 1+2/\varepsilon$ be the constant which parameterizes SRF-W. Let σ be an arbitrary sequence of requests. We let OPT denote some fixed offline optimum schedule and let α^* and $\alpha^{\text{SRF-W}}$ denote the maximum weighted delay factor achieved by OPT and SRF-W, respectively. We will show that $\alpha^{\text{SRF-W}} \leq c^2\alpha^*$. For the sake of contradiction, suppose that

SRF-W witnesses a weighted delay factor greater than $c^2\alpha^*$. We consider the *first* time t^* when SRF-W has some request in its queue with weighted delay factor $c^2\alpha^*$. Let the request $J_{q,k}$ be a request which achieves the weighed delay factor $c^2\alpha^*$ at time t^*. Let t_1 be the smallest time less than t^* such that at each time t during the interval $(t_1,t^*]$ if SRF-W is forced to broadcast by request $J_{p,i}$ at time t it is the case that

$$\frac{w_{p,i}(f_{p,i}-a_{p,i})}{S_{p,i}} > \alpha^* \quad \text{and} \quad \frac{S_{p,i}}{w_{p,i}} \le \frac{S_{q,k}}{w_{q,k}}\,.$$

Throughout this section we let $I=[t_1,t^*]$.

We let $\mathfrak{J}_I$ denote the requests which forced SRF-W to schedule broadcasts during the interval $[t_1,t^*]$. We now show that any two request in $\mathfrak{J}_I$ cannot be satisfied with a single broadcast by the optimal solution.

Lemma 4.1. OPT *cannot merge any two requests in $\mathfrak{J}_I$ into a single broadcast.*

Proof. Let $J_{x,i},J_{x,j}\in\mathfrak{J}_I$ such that $i<j$. Let t' be the time that SRF-W satisfies request $J_{x,i}$. By the definition of I, request $J_{x,i}$ must have weighted delay factor greater than α^* at this time. We also know that the request $J_{x,j}$ must arrive after time t', otherwise request $J_{x,j}$ must also be satisfied at time t'. If the optimal solution combines these requests into a single broadcast then the request $J_{x,i}$ must wait until the request $J_{x,j}$ arrives to be satisfied. However, this means that the request $J_{x,i}$ must achieve a weighted delay factor greater than α^* by OPT, a contradiction. □

As in the analysis of SSF-W we show that interval I is sufficiently long.

Lemma 4.2. $|I|=|[t_1,t^*]|\ge(c^2-c)\frac{S_{q,k}}{w_{q,k}}\alpha^*$.

Proof. The request $J_{q,k}$ has weighted delay factor $c\alpha^*$ at time

$$t'=t^*-(c^2-c)\frac{S_{q,k}}{w_{q,k}}\alpha^*\,.$$

The largest weighted delay factor any request can have at time t' is less than $c^2\alpha^*$ by definition of t^* being the first time SRF-W witnesses weighted delay factor $c^2\alpha^*$. Hence, $\alpha_{t'}\le c^2\alpha^*$. Thus, the request $J_{q,k}$ is in $Q(t')$ because $c\alpha^*\ge\alpha_{t'}/c$. Moreover, this means that any request that forced SRF-W to broadcast during $[t',t^*]$, must have weighted delay factor greater than α^* and since $J_{q,k}\in Q(t')$, the requests scheduled during $[t',t^*]$ must have a ratio of slack over weight of at most $S_{q,k}/w_{q,k}$. □

Lemma 4.3. *Any request in $\mathfrak{J}_I$ must have arrived after time $t_1-2\frac{S_{q,k}}{w_{q,k}}\alpha^*c$.*

Proof. For the sake of contradiction, suppose that some request $J_{p,i}\in\mathfrak{J}_I$ arrived at time

$$t'<t_1-2\frac{S_{q,k}}{w_{q,k}}\alpha^*c\,.$$

Recall that $S_{p,i}/w_{p,i}\le S_{q,k}/w_{q,k}$ by the definition of I. Therefore at time

$$t_1-\frac{S_{q,k}}{w_{q,k}}\alpha^*c\,,$$

$J_{p,i}$ has a weighted delay factor greater than $c\alpha^*$. Thus any request scheduled during the interval

$$I' = \left[t_1 - \frac{S_{q,k}}{w_{q,k}}\alpha^* c, t_1\right]$$

has a weighted delay factor greater than α^*. We observe that $J_{p,i}$ is in $Q(\tau)$ for $\tau \in I'$; otherwise there must be a request with weighted delay factor bigger than $c^2\alpha^*$ at time τ and this is a contradiction to the assumption that t^* is the first time that SRF-W witnessed a weighted delay factor of $c^2\alpha^*$. Therefore any request scheduled during I' has a slack over weight no bigger than $S_{p,i}/w_{p,i}$. Also we know that $S_{p,i}/w_{p,i} \le S_{q,k}/w_{q,k}$. In sum, we showed that any request done during I' had slack over weight no bigger than $S_{q,k}/w_{q,k}$ and a delay factor greater than α^*, which is a contradiction to the definition of t_1. □

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

Lemma 4.4. *Suppose c is a constant such that $c > 1+2/\varepsilon$. If SRF-W has $(1+\varepsilon)$-speed then $\alpha^{\text{SRF-W}} \le c^2\alpha^*$.*

Proof. For the sake of contradiction, suppose that $\alpha^{\text{SRF-W}} > c^2\alpha^*$. During the interval I, the number of broadcasts which SRF-W transmits is $(1+\varepsilon)|I|$. From Lemma 4.3, all the requests processed during I have arrived no earlier than $t_1 - 2c\alpha^* S_{q,k}/w_{q,k}$. We know that the optimal solution must process these requests before time t^* because these requests have weighted delay factor greater than α^* by t^*. By Lemma 4.1 the optimal solution must make a unique broadcast for each of these requests. Thus, the optimal solution must finish all of these requests in $2c\alpha^* S_{q,k}/w_{q,k} + |I|$ time steps. Thus it must hold that

$$(1+\varepsilon)|I| \le 2c\alpha^* \frac{S_{q,k}}{w_{q,k}} + |I|.$$

Using Lemma 4.2, this simplifies to $c \le 1+2/\varepsilon$, which is a contradiction to $c > 1+2/\varepsilon$. □

5 Lower bound for a natural greedy algorithm LF

In this section, we consider a natural algorithm for minimizing the maximum delay factor which is similar to SSF-W. This algorithm, which we will call LF for Longest Delay First, always schedules the page which has the largest delay factor. We will consider the algorithm LF in the unicast scheduling setting when all requests have uniform sizes. Recall that the non-uniform requests setting can be reduced to the uniform request setting when preemption is permissible.

Algorithm: LF

- The algorithm is non-preemptive. Let t be a time that the machine is free (either because a request has just finished or there are no requests to process).
- Let $Q(t)$ be the set of alive requests at t.
- Let J_i be the request in $Q(t)$ that maximizes $\frac{t-a_i}{S_i}$. Schedule J_i non-preemptively.

Notice that LF is the same as SSF-W when $c = 1$. However, we are able to show a negative result on this algorithm for minimizing the maximum delay factor. This demonstrates the importance of the trade-off between scheduling a request with smallest slack and scheduling the request with a large delay factor. The algorithm LF was suggested and analyzed in our work [21] and is inspired by LWF [21, 29]. In [21] LF is shown to be $O(k)$-competitive with $O(k)$-speed for L_k norms of flow time and delay factor in broadcast scheduling when pages have uniform sizes. It was suggested in [21] that LF may be competitive for minimizing the maximum delay factor (the L_∞-norm). The rest of this section will be devoted to showing the following theorem.

Theorem 5.1. *For any constant $s > 1$, LF is not constant competitive with s-speed for minimizing the maximum delay factor (or weighted response time) in the unicast scheduling setting when requests have uniform sizes.*

Since LF processes the request J_i such that $(t-a_i)/S_i$ is maximized, it would be helpful to formally define the quantity. Let us define the wait ratio of J_i at time $t \geq a_i$ as $(t-a_i)/S_i$; recall that a_i and S_i are the arrival time and slack size of J_i respectively. Note that J_i's wait ratio at time C_i is the same as its delay factor if J_i has delay factor no smaller than 1. Further note that J_i's delay factor is equal to $\max\{1, (f_i - a_i)/S_i\}$. The algorithm LF schedules the request with the largest wait ratio at each time. LF can be seen as a natural generalization of FIFO. This is because FIFO schedules the request with largest wait time at each time. Recall that SSF-W makes requests to wait to be scheduled to help merge potential requests in a single broadcast. The algorithm LF behaves similarly since it implicitly delays each request until it is the request with the largest wait ratio, potentially merging many requests into a single broadcast. Hence, this algorithm is a natural candidate for the problem of minimizing the maximum delay factor and it does not need any parameters as the algorithm SSF-W does. We show however that this algorithm does not have a constant competitive ratio with any constant speed.

We construct the following adversarial instance σ for any integral speed-up $s > 1$ and any integer $c \geq 2$; the assumption of s and c being integral can be easily removed by multiplying the parameters in the instance by a sufficiently large constant. For this problem instance we will show that LF has wait ratio at least c, while OPT has wait ratio at most 1. In the instance σ there is a sequence of groups of requests, $\mathcal{J}_i$ for $0 \leq i \leq k$, where k is an integer to be fixed later. We now explain the requests in each group. For simplicity of notation and readability, we will allow requests to arrive at negative times. We can shift all of the arrival times later to make the arrival times positive. All requests have unit sizes. All requests in each group $\mathcal{J}_i$ have the same arrival time

$$A_i = -(sc)^{k-i+1} - \sum_{j=0}^{k-1-i} (sc)^j$$

and have the same slack size

$$S_i = \frac{s(sc)^{k-i}}{(1-1/sc)^{k-i}}\,.$$

For notational simplicity, we override the notation S_i to refer to the slack size of any request in $\mathcal{J}_i$, rather than to refer to the slack size of an individual request J_i. There are $s(sc)^{k+1}$ requests in group $\mathcal{J}_0$ and $s(sc)^{k-i}$ requests in group $\mathcal{J}_i$ for $1 \leq i \leq k$.

We now explain how LF and OPT behave for the instance σ. Sometimes, we will use $\mathcal{J}_i$ to refer to requests in $\mathcal{J}_i$ rather than explicitly saying "requests in $\mathcal{J}_i$," since all requests in the same group are indistinguishable to the scheduler. For the first group $\mathcal{J}_0$, LF keeps processing $\mathcal{J}_0$ upon its arrival until completing it. On the other hand, we let OPT procrastinate $\mathcal{J}_0$ until OPT finishes all requests in $\mathcal{J}_1$ to $\mathcal{J}_k$. This does not hurt OPT, since the slack size of the requests in $\mathcal{J}_0$ is large relative to other requests. For each group $\mathcal{J}_i$ for $1 \le i \le k$, OPT will start $\mathcal{J}_i$ upon its arrival and complete each request in $\mathcal{J}_i$ without interruption. To the contrary, for each $1 \le i \le k$, LF will not begin scheduling $\mathcal{J}_i$ until finishing all requests in $\mathcal{J}_{i-1}$. In this way substantial delay is accumulated before LF processes $\mathcal{J}_k$ and such a delay is critical for LF, since the slack of $\mathcal{J}_k$ is small. We refer the reader to Figure 2.

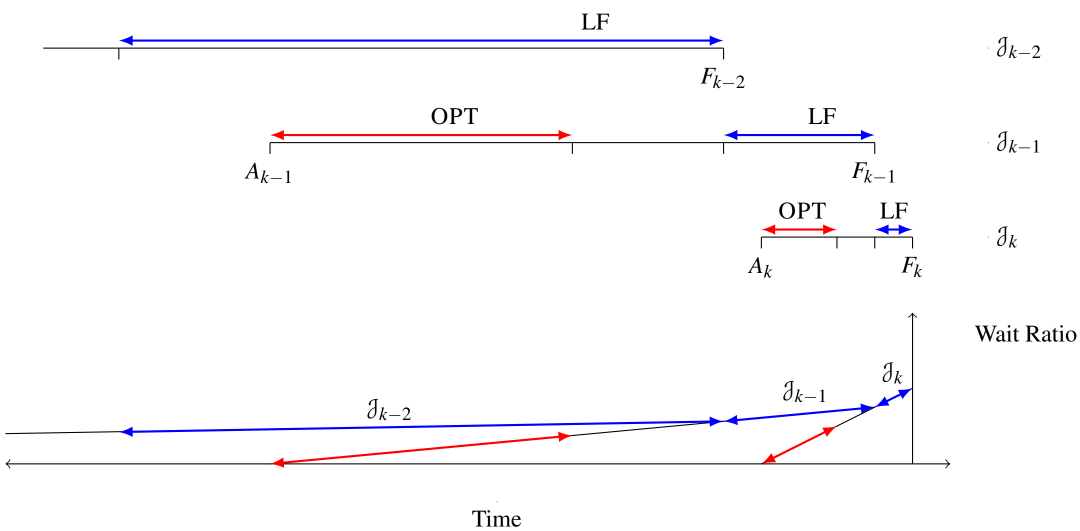

Figure 2: Comparison of scheduling of group $\mathcal{J}_k$, $\mathcal{J}_{k-1}$, and $\mathcal{J}_{k-2}$ by LF and OPT.

We now formally prove that LF has the maximum wait ratio c, while OPT has wait ratio at most 1 for the given problem instance σ. Let

$$F_i = A_i + (sc)^{k-i+1} = -\sum_{j=0}^{k-1-i} (sc)^j, \quad 0 \le i \le k.$$

Let $r_i(t)$ denote the wait ratio of any request in $\mathcal{J}_i$ at time t. We fix k to be an integer such that

$$3sc\left(1 - \frac{1}{sc}\right)^k c \le 1.$$

Lemma 5.2. *LF, given speed s, completely schedules $\mathcal{J}_0$ during $[A_0, F_0]$ and $\mathcal{J}_i$ during $[F_{i-1}, F_i]$, $1 \le i \le k$. Further, the maximum wait ratio of any request in $\mathcal{J}_k$ under LF's schedule is c.*

Proof. Observe that the length of the time interval $[A_0, F_0]$ is exactly the amount of time LF with speed s needs to completely process $\mathcal{J}_0$, since $s|[A_0, F_0]| = s(sc)^{k+1} = |\mathcal{J}_0|$. Similarly we observe that the length of the time interval $[F_{i-1}, F_i], 1 \le i \le k$ is exactly the amount of time LF with speed s requires to completely process $\mathcal{J}_i$, since $s|[F_{i-1}, F_i]| = s(sc)^{k-i} = |\mathcal{J}_i|$.

First we show that $\mathcal{J}_0$ is finished during $[A_0, F_0]$ by LF. Note that before time F_0, no request in $\mathcal{J}_j$, $j \geq 2$ arrives, since

$$F_0 = -\sum_{j=0}^{k-1} (sc)^j \leq -(sc)^{k-1} - \sum_{j=0}^{k-3} (sc)^j = A_2$$

and all requests in $\mathcal{J}_j$, $j \geq 2$ arrive no earlier than time A_2. We will show that $\mathcal{J}_0$ has the same wait ratio as $\mathcal{J}_1$ at time F_0. Then since $\mathcal{J}_0$ has a slack greater than $\mathcal{J}_1$, at any time t during $[A_0, F_0]$, $r_0(t) > r_1(t)$ and hence LF will work on $\mathcal{J}_0$ over $\mathcal{J}_1$. Indeed, the wait ratio of $\mathcal{J}_0$ at time F_0 is

$$r_0(F_0) = \frac{F_0 - A_0}{S_0} = \frac{(sc)^{k+1}}{\frac{s(sc)^k}{(1-1/sc)^k}} = c(1-1/sc)^k,$$

which is equal to the wait ratio of $\mathcal{J}_1$ at time F_0,

$$r_1(F_0) = \frac{F_0 - A_1}{S_1} = \frac{(sc)^k - (sc)^{k-1}}{\frac{s(sc)^{k-1}}{(1-1/sc)^{k-1}}} = c(1-1/sc)^k.$$

To complete the proof, we show that $\mathcal{J}_i$, $i \geq 1$ is finished during $[F_{i-1}, F_i]$ by LF. This proof is very similar to the above. Note that no request in $\mathcal{J}_j$, $j \geq i+2$ arrives before time F_i, since

$$F_i = -\sum_{j=0}^{k-1-i} (sc)^j \leq -(sc)^{k-i-1} - \sum_{j=0}^{k-3-i} (sc)^j = A_{i+2}$$

and all requests in $\mathcal{J}_j$, $j \leq i+2$ arrive no earlier than time A_{i+2}. We will show that $\mathcal{J}_i$ has the same wait ratio as $\mathcal{J}_{i+1}$ at time F_i. Then since the slack of $\mathcal{J}_{i+1}$ is smaller than $\mathcal{J}_i$, at any time t during $[F_{i-1}, F_i]$, $\mathcal{J}_i$ will have wait ratio no smaller than $\mathcal{J}_{i+1}$ and hence LF will work on $\mathcal{J}_i$ over $\mathcal{J}_{i+1}$. Indeed, the wait ratio of $\mathcal{J}_i$ at time F_i is

$$r_i(F_i) = \frac{F_i - A_i}{S_i} = \frac{(sc)^{k-i+1}}{\frac{s(sc)^{k-i}}{(1-1/sc)^{k-i}}} = c(1-1/sc)^{k-i},$$

which is equal to the current delay factor of $\mathcal{J}_{i+1}$ at time F_i,

$$r_{i+1}(F_i) = \frac{F_i - A_{i+1}}{S_{i+1}} = \frac{(sc)^{k-i} - (sc)^{k-i-1}}{\frac{s(sc)^{k-1-i}}{(1-1/sc)^{k-i-1}}} = c(1-1/sc)^{k-i}.$$

Hence LF has wait ratio at least c for a certain request in $\mathcal{J}_k$. □

In the following lemma, we show that there exists a feasible scheduling by OPT that has wait ratio at most 1, which together with Lemma 5.2 will complete the proof of Theorem 5.1.

Lemma 5.3. *Consider a schedule which processes all requests in $\mathcal{J}_0$ during $[F_k, F_k + |\mathcal{J}_0|]$ and all requests in $\mathcal{J}_i$ during $[A_i, A_i + |\mathcal{J}_i|]$ for $1 \leq i \leq k$ with speed 1. This schedule is feasible and, moreover, the maximum wait ratio of any request under this schedule is at most one.*

Proof. We first observe that the time intervals $[F_k, F_k + |\mathcal{J}_0|]$ and $[A_i, A_i + |\mathcal{J}_i|]$ for $1 \leq i \leq k$ do not overlap, since for $i \geq 1$,

$$A_{i+1} - (A_i + |\mathcal{J}_i|) = (sc)^{k+1-i} + (sc)^{k-1-i} - s(sc)^{k-i} - (sc)^{k-i} \geq (sc)^{k-i}(sc - s - 1) > 0\,,$$

and $F_k - (A_k + |\mathcal{J}_k|) = sc - s > 0$. Further, all requests in $\mathcal{J}_0$ can be completed during $[F_k, F_k + |\mathcal{J}_0|]$ by a scheduler with speed 1. Likewise, this shows that all requests in $\mathcal{J}_i$ can be completed during $[A_i, A_i + |\mathcal{J}_i|]$ by a scheduler with speed 1. Hence this is a feasible schedule for a 1 speed algorithm.

It now remains to upper bound the maximum wait ratio of any request under the suggested schedule. Consider any request in $\mathcal{J}_i$, $i \geq 1$. The maximum wait ratio of $\mathcal{J}_i$ under the schedule is

$$\frac{A_i + |\mathcal{J}_i| - A_i}{S_i} = \frac{s(sc)^{k-i}}{\frac{s(sc)^{k-i}}{(1-1/sc)^{k-i}}} = (1 - 1/sc)^{k-i} < 1\,.$$

The maximum wait ratio of any request in $\mathcal{J}_0$ is

$$\begin{aligned} r_0(F_k + |\mathcal{J}_0|) = \frac{F_k + |\mathcal{J}_0| - A_0}{S_0} &= \frac{(s+1)(sc)^{k+1} + \sum_{j=0}^{k-1}(sc)^j}{\frac{s(sc)^k}{(1-1/sc)^k}} \\ &\leq \frac{3s(sc)^{k+1}}{\frac{s(sc)^k}{(1-1/sc)^k}} = 3sc(1 - 1/sc)^k \leq 1\,. \end{aligned}$$

The last inequality holds since k was chosen to satisfy the inequality. □

6 Conclusions

We considered online scheduling to minimize maximum (weighted) response time and to minimize maximum (weighted) delay factor. Delay factor and the weighted response time metrics have not been previously considered. We developed scalable algorithms for these metrics in both the unicast and broadcast scheduling models. Our algorithms demonstrate an interesting trade off on whether to prioritize requests with larger weight or those that have waited longer in the system. Understanding this trade off has led to the first online scalable algorithm for minimizing average response time in broadcast scheduling [34] which has been an open problem for several years.

We close with the following open problems. Our algorithm for the maximum delay factor with uniform sized pages uses a parameter that explicitly depends on the speed given to the algorithm. Is there an algorithm that is scalable without needing this information? FIFO is 2-competitive for minimizing maximum response time in broadcast scheduling. In the offline setting can the 2-approximation implied by FIFO be improved? For the more general problem of minimizing maximum delay factor, no non-trivial offline approximation is known that does not use resource augmentation.

Acknowledgments We thank Samir Khuller for clarifications on previous work and for his encouragement, and Kirk Pruhs for his comments and suggestions. We also thank the anonymous reviewers for their suggestions that helped improve the presentation.

References

[1] SWARUP ACHARYA, MICHAEL FRANKLIN, AND STANLEY ZDONIK: Dissemination-based data delivery using broadcast disks. *IEEE Personal Communications*, 2(6):50–60, Dec 1995. [doi:10.1109/98.475988] 166

[2] DEMET AKSOY AND MICHAEL FRANKLIN: R x W: A scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Transactions on Networking*, 7(6):846–860, 1999. [doi:10.1109/90.811450] 166

[3] NIR AVRAHAMI AND YOSSI AZAR: Minimizing total flow time and total completion time with immediate dispatching. *Algorithmica*, 47:253–268, 2007. Preliminary version in SPAA'03. [doi:10.1007/s00453-006-0193-6] 169, 170, 173, 174

[4] NIKHIL BANSAL, MOSES CHARIKAR, SANJEEV KHANNA, AND JOSEPH (SEFFI) NAOR: Approximating the average response time in broadcast scheduling. In *Proc. 16th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'05)*, pp. 215–221. SIAM, 2005. 169

[5] NIKHIL BANSAL, DON COPPERSMITH, AND MAXIM SVIRIDENKO: Improved approximation algorithms for broadcast scheduling. *SIAM J. Comput.*, 38(3):1157–1174, 2008. Preliminary version in SODA'06. [doi:10.1137/060674417] 169

[6] NIKHIL BANSAL, RAVISHANKAR KRISHNASWAMY, AND VISWANATH NAGARAJAN: Better scalable algorithms for broadcast scheduling. In *Proc. 37th Internat. Colloq. on Automata, Languages and Programming (ICALP'10)*, pp. 324–335. Springer, 2010. [doi:10.1007/978-3-642-14165-2_28] 170, 181, 183

[7] NIKHIL BANSAL AND KIRK PRUHS: Server scheduling in the L_p norm: A rising tide lifts all boat. In *Proc. 35th STOC*, pp. 242–250. ACM Press, 2003. [doi:10.1145/780542.780580] 169, 170, 173

[8] 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. [doi:10.1287/moor.27.3.518.314] 166

[9] AMOTZ BAR-NOY, SUDIPTO GUHA, YOAV KATZ, JOSEPH (SEFFI) NAOR, BARUCH SCHIEBER, AND HADAS SHACHNAI: Throughput maximization of real-time scheduling with batching. *ACM Transactions on Algorithms*, 5(2):18:1–18:17, 2009. [doi:10.1145/1497290.1497294] 166

[10] YAIR BARTAL AND S. MUTHUKRISHNAN: Minimizing maximum response time in scheduling broadcasts. In *Proc. 11th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'00)*, pp. 558–559. SIAM, 2000. [ACM:338605] 166, 167, 169

[11] MICHAEL A. BENDER, SOUMEN CHAKRABARTI, AND S. MUTHUKRISHNAN: Flow and stretch metrics for scheduling continuous job streams. In *Proc. 9th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'98)*, pp. 270–279. SIAM, 1998. [ACM:314613.314715] 167, 170, 171

[12] MICHAEL A. BENDER, RAPHAËL CLIFFORD, AND KOSTAS TSICHLAS: Scheduling algorithms for procrastinators. *J. Scheduling*, 11(2):95–104, 2008. [doi:10.1007/s10951-007-0038-4] 170

[13] MICHAEL A. BENDER, S. MUTHUKRISHNAN, AND RAJMOHAN RAJARAMAN: Approximation algorithms for average stretch scheduling. *J. Scheduling*, 7(3):195–222, 2004. [doi:10.1023/B:JOSH.0000019681.52701.8b] 167, 170

[14] GENNARO BOGGIA, PIETRO CAMARDA, LUIGI MAZZEO, AND MARINA MONGIELLO: Performance of batching schemes for multimedia-on-demand services. *IEEE Transactions on Multimedia*, 7(5):920–931, 2005. [doi:10.1109/TMM.2005.854383] 166

[15] ALAN BURNS AND SANJOY BARUAH: Sustainability in real-time scheduling. *Journal of Computing Science and Engineering*, 2(1):74–97, 2008. `http://jcse.kiise.org/PublishedPaper/topic_abstract.asp?idx=15`. 168

[16] 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 *Proc. 10th Ann. Internat. Computing and Combinatorics Conf. (COCOON'04)*, pp. 210–218. Springer Berlin / Heidelberg, 2004. [doi:10.1007/978-3-540-27798-9_24] 168, 169, 170

[17] JESSICA CHANG, THOMAS ERLEBACH, RENARS GAILIS, AND SAMIR KHULLER: Broadcast scheduling: Algorithms and complexity. *ACM Transactions on Algorithms*, 7(4):47:1–47:14, 2011. Preliminary version in SODA'08. [doi:10.1145/2000807.2000815] 166, 167, 168, 169, 170, 176, 177, 178

[18] MOSES CHARIKAR AND SAMIR KHULLER: A robust maximum completion time measure for scheduling. In *Proc. 17th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'06)*, pp. 324–333. SIAM, 2006. [doi:10.1145/1109557.1109594] 170

[19] CHANDRA CHEKURI, AVIGDOR GAL, SUNGJIN IM, SAMIR KHULLER, JIAN LI, RICHARD MCCUTCHEN, BENJAMIN MOSELEY, AND LOUIQA RASCHID: New models and algorithms for throughput maximization in broadcast scheduling - (extended abstract). In *Proc. 8th Workshop on Approximation and Online Algorithms (WAOA'10)*, pp. 71–82. Springer, 2010. [doi:10.1007/978-3-642-18318-8_7] 169

[20] CHANDRA CHEKURI, ASHISH GOEL, SANJEEV KHANNA, AND AMIT KUMAR: Multi-processor scheduling to minimize flow time with ε resource augmentation. In *Proc. 36th STOC*, pp. 363–372. ACM Press, 2004. [doi:10.1145/1007352.1007411] 169, 170, 173, 174

[21] CHANDRA CHEKURI, SUNGJIN IM, AND BENJAMIN MOSELEY: Longest wait first for broadcast scheduling [extended abstract]. In *Proc. 7th Workshop on Approximation and Online Algorithms (WAOA'09)*, pp. 62–74. Springer, 2009. [doi:10.1007/978-3-642-12450-1_6] 169, 187

[22] CHANDRA CHEKURI, SUNGJIN IM, AND BENJAMIN MOSELEY: Minimizing maximum response time and delay factor in broadcast scheduling. In *Proc. 17th Ann. European Symp. on Algorithms (ESA'09)*, pp. 444–455. Springer, 2009. [doi:10.1007/978-3-642-04128-0_40] 165, 180, 181, 182

[23] CHANDRA CHEKURI AND BENJAMIN MOSELEY: Online scheduling to minimize the maximum delay factor. In *Proc. 20th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'09)*. SIAM, 2009. [ACM:1496891] 165, 180, 181, 182

[24] MAREK CHROBAK, CHRISTOPH DÜRR, WOJCIECH JAWOR, ŁUKASZ KOWALIK, AND MACIEJ KUROWSKI: A note on scheduling equal-length jobs to maximize throughput. *J. of Scheduling*, 9(1):71–73, 2006. [doi:10.1007/s10951-006-5595-4] 169, 170

[25] ASIT DAN, DINKAR SITARAM, AND PERWEZ SHAHABUDDIN: Dynamic batching policies for an on-demand video server. *Multimedia Syst.*, 4(3):112–121, 1996. [doi:10.1007/s005300050016] 166

[26] RAJAT K. DEB: Optimal control of bulk queues with compound Poisson arrivals and batch service. *Opsearch.*, 21:227–245, 1984. 166

[27] RAJAT K. DEB AND RICHARD F. SERFOZO: Optimal control of batch service queues. *Adv. Appl. Prob.*, 5:340–361, 1973. [doi:10.2307/1426040] 166

[28] JEFF EDMONDS AND KIRK PRUHS: Multicast pull scheduling: When fairness is fine. *Algorithmica*, 36(3):315–330, 2003. [doi:10.1007/s00453-003-1018-5] 167, 169, 170

[29] JEFF EDMONDS AND KIRK PRUHS: A maiden analysis of longest wait first. *ACM Trans. Algorithms*, 1(1):14–32, 2005. [doi:10.1145/1077464.1077467] 169, 187

[30] JEFF EDMONDS AND KIRK PRUHS: Scalably scheduling processes with arbitrary speedup curves. In *Proc. 20th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'09)*, pp. 685–692. SIAM, 2009. [doi:10.1145/1496770.1496845] 169

[31] THOMAS ERLEBACH AND ALEXANDER HALL: NP-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. *Journal of Scheduling*, 7:223–241, 2004. Preliminary version in SODA'02. [doi:10.1023/B:JOSH.0000019682.75022.96] 169

[32] 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. Preliminary version in IPCO'02. [doi:10.1007/s00453-003-1058-x] 169

[33] RAJIV GANDHI, SAMIR KHULLER, SRINIVASAN PARTHASARATHY, AND ARAVIND SRINIVASAN: Dependent rounding and its applications to approximation algorithms. *J. ACM*, 53(3):324–360, 2006. [doi:10.1145/1147954.1147956] 169

[34] SUNGJIN IM AND BENJAMIN MOSELEY: An online scalable algorithm for average flow time in broadcast scheduling. In *Proc. 21st Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'10)*, pp. 1322–1333. SIAM, 2010. [ACM:1873601.1873708] 170, 190

[35] BALA KALYANASUNDARAM AND KIRK PRUHS: Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000. Preliminary version in FOCS'95. [doi:10.1145/347476.347479] 168

[36] BALA KALYANASUNDARAM, KIRK R. PRUHS, AND MAHENDRAN VELAUTHAPILLAI: Scheduling broadcasts in wireless networks. *J. Scheduling*, 4(6):339–354, 2001. Preliminary version in ESA'00. [doi:10.1002/jos.87] 169

[37] DAVID KARGER, CLIFF STEIN, AND JOEL WEIN: Scheduling algorithms. In MIKHAIL J. ATALLAH, editor, *Handbook on Algorithms and Theory of Computation*, chapter 35. CRC Press, 1999. [doi:10.1201/9781420049503-c36] 167

[38] SAMIR KHULLER AND YOO-AH KIM: Equivalence of two linear programming relaxations for broadcast scheduling. *Oper. Res. Lett.*, 32(5):473–478, 2004. [doi:10.1016/j.orl.2003.11.012] 170

[39] JAE-HOON KIM AND KYUNG-YONG CHWA: Scheduling broadcasts with deadlines. *Theoret. Comput. Sci.*, 325(3):479–488, 2004. Preliminary version in COCOON'03. [doi:10.1016/j.tcs.2004.02.047] 168, 169, 170

[40] INSUP LEE, JOSEPH Y-T. LEUNG, AND SANG H. SON, editors. *Handbook of Real-Time and Embedded Systems*. CRC Press, 2007. [doi:10.1201/9781420011746] 170

[41] KIRK PRUHS: Competitive online scheduling for server systems. *SIGMETRICS Perform. Eval. Rev.*, 34(4):52–58, 2007. [doi:10.1145/1243401.1243411] 169, 170

[42] KIRK PRUHS, JIŘÍ SGALL, AND ERIC TORNG: Online scheduling. In JOSEPH Y-T. LEUNG, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, 2004. 169, 170

[43] KIRK R. PRUHS AND PATCHRAWAT UTHAISOMBUT: A comparison of multicast pull models. *Algorithmica*, 42(3-4):289–307, 2005. Preliminary version in ESA'02. [doi:10.1007/s00453-005-1170-1] 167

[44] JIŘÍ SGALL: On-line scheduling. In *Developments from a June 1996 seminar on Online algorithms*, pp. 196–231. Springer-Verlag, 1998. [ACM:723918] 167

[45] REHA UZSOY: Scheduling batch processing machines with incompatible job families. *International Journal of Production Research*, 33(10):2685–2708, 1995. [doi:10.1080/00207549508904839] 166

[46] 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 *Proc. 12th Ann. Internat. Computing and Combinatorics Conf. (COCOON'06)*, pp. 320–329. Springer, 2006. [doi:10.1007/11809678_34] 168, 169, 170

AUTHORS

Chandra Chekuri
Associate professor
Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
chekuri@cs.illinois.edu
http://www.cs.uiuc.edu/~chekuri/

Sungjin Im
Graduate student
Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
im3@cs.illinois.edu
http://www.cs.illinois.edu/homes/im3/

Benjamin Moseley
Graduate student
Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
bmosele2@cs.illinois.edu
http://www.cs.illinois.edu/homes/bmosele2/

ABOUT THE AUTHORS

CHANDRA CHEKURI is an Associate Professor of Computer Science at the University of Illinois at Urbana-Champaign (UIUC). He obtained his Ph. D. in Computer Science at Stanford University under the supervision of Rajeev Motwani in 1998, and then worked at Lucent Bell Labs for eight years. His primary research interests are algorithms, combinatorial optimization, and mathematical programming.

SUNGJIN IM is a Ph. D. student at the University of Illinois at Urbana-Champaign, advised by Chandra Chekuri. He received his undergraduate and master degrees from Seoul National University. His research interests include online algorithms, approximation algorithms and combinatorial optimization.

BENJAMIN MOSELEY is a Ph. D. student at the University of Illinois at Urbana-Champaign, advised by Chandra Chekuri. He received his undergraduate and master degrees from the same university. His research interests include algorithms, scheduling, large data analysis, online algorithms and algorithmic applications.