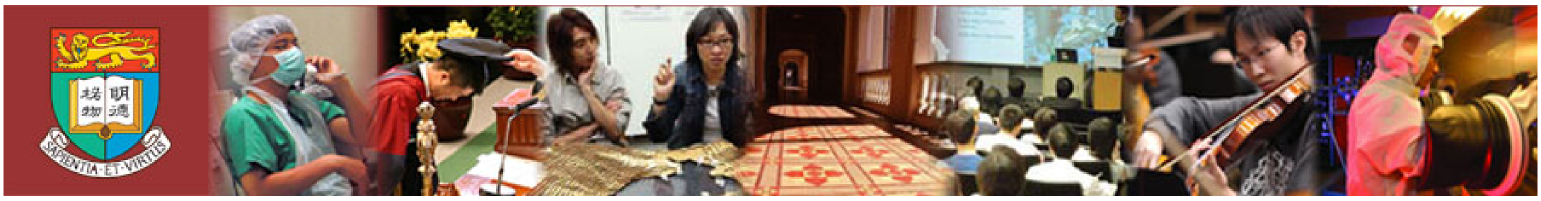

SIAM J. Comput.
Vol. 37, No. 5, pp. 1595–1612

EXTRA UNIT-SPEED MACHINES ARE ALMOST AS POWERFUL AS SPEEDY MACHINES FOR FLOW TIME SCHEDULING[*]

HO-LEUNG CHAN[†], TAK-WAH LAM[†], AND KIN-SHING LIU[†]

Abstract. We study online scheduling of jobs to minimize the flow time and stretch on parallel machines. We consider algorithms that are given extra resources so as to compensate for the lack of future information. Recent results show that a modest increase in machine speed can provide very competitive performance; in particular, using $O(1)$ times faster machines, the algorithm SRPT (shortest remaining processing time) is 1-competitive for both flow time [C. A. Phillips et al., in *Proceedings of STOC*, ACM, New York, 1997, pp. 140–149] and stretch [W. T. Chan et al., in *Proceedings of MFCS*, Springer-Verlag, Berlin, 2005, pp. 236–247] and HDF (highest density first) is $O(1)$-competitive for weighted flow time [L. Becchetti et al., in *Proceedings of RANDOM-APPROX*, Springer-Verlag, Berlin, 2001, pp. 36–47]. Using extra unit-speed machines instead of faster machines to achieve competitive performance is more challenging, as a faster machine can speed up a job but extra unit-speed machines cannot. This paper gives a nontrivial relationship between the extra-speed and extra-machine analyses. It shows that competitive results via faster machines can be transformed to similar results via extra machines, hence giving the first algorithms that, using $O(1)$ times unit-speed machines, are 1-competitive for flow time and stretch and $O(1)$-competitive for weighted flow time.

Key words. online scheduling, flow time, stretch, competitive analysis, extra-resource augmentation

AMS subject classifications. 68Q25, 68W15, 68W40

DOI. 10.1137/060653445

1. Introduction.

In this paper we revisit the problem of online scheduling of jobs to minimize the flow time and stretch on $m \geq 2$ parallel machines (see [24] for a survey). Each job is released at an unpredictable time and is sequential in nature (i.e., it cannot be executed by more than one machine at a time). We consider the case where the processing time (work) of a job is known when it is released. Preemption is allowed at no cost, i.e., a preempted job can be resumed at the point of preemption on any machine. SRPT (shortest remaining processing time first) is a typical example for scheduling in this setting.

Given a schedule, the flow time of a job is the amount of time between its release time and its completion time, and the stretch is the ratio of the flow time to the processing time. In some applications, each job is given a weight, and the concern is the weighted flow time. Common objectives for job scheduling are to minimize the total (or, equivalently, average) flow time (e.g., [19, 20, 2, 1, 21]), stretch (e.g., [7, 9, 22]), or weighted flow time (e.g., [4, 14, 3]) of all jobs. Minimizing stretch is actually a special case of minimizing weighted flow time if we assign the weight of each job to be the reciprocal of its processing time. An online scheduler is said to be c-competitive for flow time (resp., stretch, weighted flow time) if for any job sequence it guarantees the total flow time (resp., stretch, weighted flow time) to be at most c times that of the optimal offline schedule.

[*]Received by the editors March 2, 2006; accepted for publication (in revised form) August 23, 2007; published electronically February 8, 2008. A preliminary version of this paper appeared in the *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2006, pp. 334–343.
http://www.siam.org/journals/sicomp/37-5/65344.html

[†]Department of Computer Science, University of Hong Kong, Hong Kong (hlchan@cs.hku.hk, twlam@cs.hku.hk, ksliu@cs.hku.hk).

Related work. SRPT is perhaps the most well-studied online algorithm for minimizing flow time. For scheduling a single machine ($m = 1$), SRPT is 1-competitive [19]. For $m \geq 2$ machines, Leonardi and Raz [20] showed that SRPT achieves the best possible competitive ratio, which is $\Theta(\min(\log n/m, \log \Delta))$, where n is the number of jobs and Δ is the maximum to minimum ratio of processing times. In the offline context, minimizing total flow time on parallel machines is NP-hard [15], and no algorithm is known to have a constant approximation ratio.

Resource augmentation, pioneered by Kalyanasundaram and Pruhs [17], is a popular approach to studying better performance guarantee for improving the competitiveness of online scheduling (e.g., [23, 21, 13, 11, 16, 6]). Specifically, this approach allows the online scheduler to have extra resources so as to compensate for the lack of future knowledge. The key concerns include (i) whether extra resources can lead to 1-competitive (or even better) performance against the optimal offline algorithm using no extra resources, and (ii) how competitive an arbitrarily small amount of extra resources can be. Extra resources can be in the form of faster machines or extra (unit-speed) machines. Below we denote a machine that can complete $s \geq 1$ units of work in one unit of time as an s-speed machine. For minimizing flow time on parallel machines, Phillips et al. [23] were the first to show that SRPT when given $(2-1/m)$-speed machines is 1-competitive or, in short, $(2-1/m)$-speed 1-competitive. McCullough and Torng [21] later showed that SRPT is indeed α-speed $\frac{1}{\alpha}$-competitive for any $\alpha \geq 2 - 1/m$.

Let us switch to the results on minimizing stretch and weighted flow time on parallel machines (one can refer to [22, 5, 4, 23] for results on a single machine). For the case of stretch, Muthukrishnan et al. [22] have showed that SRPT is 14-competitive and no online algorithm can be 1-competitive. Chekuri, Khanna, and Zhu [14] proposed a different algorithm that is 9.81-competitive. They also gave a lower bound on the competitive ratio for weighted flow time of $\Omega(\min(\sqrt{\Delta}, \sqrt{W}, (n/m)^{1/4}))$, where W is the maximum to minimum ratio of the weights. With resource augmentation, Becchetti et al. [6] showed that HDF (highest density first) is $(2 + 2\epsilon)$-speed $(1 + \frac{1}{\epsilon})$-competitive for weighted flow time. This implies that SJF (shortest job first) is $(2 + 2\epsilon)$-speed $(1 + \frac{1}{\epsilon})$-competitive for stretch. Recently, more results on stretch have become known. Chekuri et al. [13] proved that the nonmigratory algorithm IMD (proposed in [1]) is $(1 + \epsilon)$-speed $O(1 + \frac{1}{\epsilon})$-competitive, and Chan et al. [12] showed that SRPT is indeed 5-speed 1-competitive.

Improving the competitiveness via extra unit-speed machines is more challenging. While a faster machine can speed up a job, multiple unit-speed machines cannot. In other words, we cannot use x unit-speed machines to simulate an x-speed machine, yet the reverse is possible (using time-sharing). The literature contains only a few results on exploiting extra machines to obtain competitive scheduling (see [17, 23, 18, 13]). For flow time scheduling on parallel machines, Chekuri et al. [13] have shown that the algorithm IMD when given $(1+\epsilon)m$ unit-speed machines is $O(1+\frac{1}{\epsilon})$-competitive for both flow time and stretch. Whether $O(m)$ unit-speed machines can make an algorithm 1-competitive for flow time or stretch has been an open problem. There are also results on exploiting extra machines in other problem settings of flow time scheduling [17, 23]. In particular, Kalyanasundaram and Pruhs [17] studied the nonclairvoyant setting on a single machine, and Phillips et al. [23] considered the nonpreemptive setting for parallel machines. The power of extra machines has also been studied in the context of deadline scheduling by Koo et al. [18] and Phillips et al. [23].

To ease our discussion, we adopt the following notation. Let α and τ be any

TABLE 1
Results on flow time scheduling. Results that are given in this paper are marked with †. Note that $\epsilon > 0$ and s are any real numbers and $h \geq 1$ is any integer.

	Extra speed			Extra machines		
	Speed	Competitive ratio		Machines	Competitive ratio	
Flow time	$(1+\epsilon)$	$O(1+1/\epsilon)$	[13]	$\lceil(1+\epsilon)m\rceil$	$O(1+1/\epsilon)$	[13]
	2	1	[23]	$\lceil(2+\epsilon)m\rceil$	$1+1/\epsilon$	†
	$s \geq 2$	$1/s$	[21]	$34m$	1	†
Stretch	$(1+\epsilon)$	$O(1+1/\epsilon)$	[13]	$\lceil(1+\epsilon)m\rceil$	$O(1+1/\epsilon)$	[13]
	5	1	[12]	$533m$	1	†
Weighted flow time	$2+2\epsilon$	$1+1/\epsilon$	[6]	$\lceil(4+24\epsilon)m\rceil$	$8+1/\epsilon$	†
	$16s$, $s \geq 1$	$1/s$	†			
Waiting time	$s \geq 2$	$1/s$	[21]	$(36h-2)m$, $h \geq 1$	$1/h$	†

positive real constants. An algorithm A is said to be α-speed c-competitive (resp., τ-machine c-competitive) for a certain objective function if, for any job sequence, A using m α-speed machines (resp., $\lceil \tau m \rceil$ unit-speed machines) has a performance at most c times of any optimal offline algorithm using m unit-speed machines. When we consider an algorithm A running on m α-speed machines (resp., $\lceil \tau m \rceil$ unit-speed machines), we refer it as $A(\alpha)$ (resp., $A\langle\tau\rangle$).

Our results. This paper shows a nontrivial relationship between the extra-machine analysis and the extra-speed analysis of flow time scheduling. In particular, two methods are given to transform results on competitiveness via faster machines into similar results via extra unit-speed machines. These transformations give the first algorithms that are $O(1)$-machine 1-competitive for flow time and stretch and $O(1)$-competitive for weighted flow time. See Table 1 for a summary of results. Details are as follows.

Flow time transformation. The first transformation is relatively simple, serving as a warm-up. It aims to preserve the flow time of each individual job. Specifically, given an α-speed algorithm $A(\alpha)$ for some $\alpha > 1$, we want to transform A to an algorithm A' that uses extra unit-speed machines to match the flow time of each job as closely as possible. Specifically, our transformation guarantees that A' when given $O(\alpha)m$ (unit-speed) machines increases the flow time of each job at most $\alpha(1+o(1))$ times. Since SRPT is α-speed $\frac{1}{\alpha}$-competitive for flow time, the transformation gives an algorithm that is $O(\alpha)$-machine $(1+o(1))$-competitive (and more precisely, $(2+\epsilon)$-machine $(1+\frac{1}{\epsilon})$-competitive for any $\epsilon > 0$). Note that A' also preserves the competitiveness on weighted flow time and stretch. Thus, based on HDF [6], the transformation gives an $O(1)$-machine $O(1)$-competitive algorithm for weighted flow time.

Waiting time transformation. The waiting time of a job is the amount of time the job is waiting for processing before it is completed. To obtain an $O(1)$-machine 1-competitive algorithm for flow time and stretch, we need a more complicated transformation based on the total waiting time of jobs. By definition, an algorithm A is $O(1)$-machine 1-competitive for waiting time if and only if A is $O(1)$-machine 1-competitive for on flow time. Note that using extra unit-speed machines can possibly improve the competitive ratio on waiting time to be smaller than one, but it is impossible for flow time.

Consider any algorithm A using α-speed machines. Denote $L_{A(\alpha)}(I)$ the total waiting time incurred for a job sequence I by $A(\alpha)$. The work of McCullough and

Torng [21] implies that SRPT is α-speed $\frac{1}{\alpha}$-competitive for waiting time, where $\alpha \geq 2 - 1/m$. That is, for any I, $L_{SRPT(\alpha)}(I) \leq \frac{1}{\alpha} L_{OPT}(I)$, where OPT denotes the optimal offline algorithm using m unit-speed machines. Using unit-speed machines to simulate SRPT(α) or any $A(\alpha)$ does not necessarily blow up the total waiting time α times. Ideally we want to transform $A(\alpha)$ to an algorithm A' using $\tau m = O(\alpha)m$ unit-speed machines such that $L_{A'\langle\tau\rangle}(I) \leq cL_{A(\alpha)}(I)$, where c is a constant independent of α. Then, substituting $A(\alpha)$ with SRPT(c), we have $L_{A'\langle\tau\rangle}(I) \leq L_{OPT}(I)$. Such a constant c, however, does not exist.[1]

To obtain an $O(1)$-machine 1-competitive algorithm for waiting time, we aim at a less demanding requirement, namely, $L_{A'\langle\tau\rangle}(I) \leq c\,L_{A(\alpha)}(I) + o(L_{OPT}(I))$. In fact, we find that $c = 2$ is already feasible. Then, substituting A with SRPT and α with $O(c)$, we have $L_{A'\langle\tau\rangle}(I) \leq L_{OPT}(I)$, and thus A' is $O(1)$-machine 1-competitive for waiting time, as well as for flow time.

The second transformation can be extended to give a guarantee for normalized waiting time (i.e., the waiting time divided by the processing time). This leads to an algorithm that is $O(1)$-machine 1-competitive for stretch.

Technically speaking, the transformations are based on two concepts called rate control and waiting time allowance. Roughly speaking, we need rate control when jobs are released in a bulk; the idea is to partially process and spread these jobs in a certain way without blowing up the flow time. The other concept is about estimating the maximum waiting time of each job that would not exceed that of the offline optimal algorithm. Both concepts make scheduling easy. To make these two concepts viable, we exploit a simulation of an α-speed competitive algorithm.

This paper also contributes to the extra-speed analysis of SJF and HDF. In particular, we improve the result in [6] to show that HDF can be 16-speed 1-competitive for weighted flow time.

2. Transformation that preserves flow time.

Throughout this paper, we use I to denote a sequence of jobs, and denote the release time and the processing time (i.e., the required work) of a job J as $r(J)$ and $p(J)$, respectively. Note that both $r(J)$ and $p(J)$ are real numbers. Let $A(\alpha)$ be an algorithm using m α-speed machines, where $\alpha \geq 1$ is any real number. This section shows how to transform $A(\alpha)$ to an algorithm, called `Scatter`$(A(\alpha), \tau)$, that uses $\lceil \tau m \rceil$ unit-speed machines for any $\tau > \alpha$ and incurs a flow time comparable to $A(\alpha)$ as follows.

LEMMA 1. *Consider any job sequence I. For each job $J \in I$, the flow time of J in the schedule of* `Scatter`$(A(\alpha), \tau)$ *is at most $\alpha(1 + \frac{\alpha-1}{\tau-\alpha})$ times in the schedule of $A(\alpha)$.*

Details of `Scatter`$(A(\alpha), \tau)$ are as follows. `Scatter`$(A(\alpha), \tau)$ divides the $\lceil \tau m \rceil$ machines into two bands. Band 1 uses m machines and Band 2 $\lceil (\tau - 1)m \rceil$ machines. A newly released job J always goes to Band 1 where it is partially processed. Then J is transferred to Band 2 for completion. Consider any sequence I of jobs. We denote the flow time of a job J in the schedule of $A(\alpha)$ as $F_{A(\alpha)}(J)$. We aim to bound the

[1]We consider a simple example where $\alpha = 2$. Let I be a job sequence such that the ith job is released at time $1 - (1/2)^i$ and the required work is $(1/2)^i$. An algorithm with m 2-speed machines can complete each job before the next one is released, thus incurring zero waiting time. On the other hand, any algorithm using τm unit-speed machines must have some job wait after the $(\tau m + 1)$th job is released, thus incurring nonzero waiting time. Note that even one 2-speed machine can complete all jobs with zero waiting time.

flow time of J in Band 1 and Band 2, denoted as $F_1(J)$ and $F_2(J)$, as follows:

$$\text{(i) } F_1(J) = F_{A(\alpha)}(J); \text{ and } \quad \text{(ii) } F_2(J) \le \frac{(\alpha-1)\tau}{\tau-\alpha} F_1(J).$$

Then it follows that the flow time of J in the schedule of $\texttt{Scatter}(A(\alpha),\tau)$ is $F_1(J) + F_2(J) \le (1 + \frac{(\alpha-1)\tau}{\tau-\alpha})F_{A(\alpha)}(J) = \alpha(1+\frac{\alpha-1}{\tau-\alpha})F_{A(\alpha)}(J)$, which is as stated in Lemma 1.

Simulation. Requirement (i) can be achieved easily by simulating the execution of $A(\alpha)$. Precisely, Band 1 uses m machines and schedules the jobs according to a simulated copy of $A(\alpha)$, which uses m α-speed machines. That is, Band 1 runs a job J if and only if $A(\alpha)$ runs the job J. When $A(\alpha)$ completes J, Band 1 transfers J to Band 2. Thus, $F_1(J) = F_{A(\alpha)}(J)$, and J is processed in Band 1 for exactly $p(J)/\alpha$ units of work.

Rate control. Let $rem(J)$ be the amount of remaining work of a job J when it is transferred to Band 2. Jobs may be released in bulk to Band 1, yet they will each be partially processed before being transferred to Band 2 and will thus spread out eventually. Band 1 controls the rate of work transferred to Band 2 in the sense that jobs released and transferred within any time interval have bounded remaining work (see Lemma 2 for technical details). With rate control, Requirement (ii) can be satisfied easily using a simple strategy, namely, the latest release time first algorithm (LRT), which at any time t processes jobs with latest release time (to Band 1). Ties are broken arbitrarily.

The above discussion of `Scatter` is summarized in Algorithm 1, followed by two lemmas on the work transferred to Band 2 and the flow time in Band 2.

Algorithm 1. $\texttt{Scatter}(A(\alpha),\tau)$, which uses $\lceil \tau m \rceil$ unit-speed machines.

Job Release: A newly released job goes to Band 1.

Band 1: It uses m machines. Jobs are scheduled according to a simulated copy of $A(\alpha)$. When a job J is completed in the simulated $A(\alpha)$, it is transferred to Band 2.

Band 2: It uses $\lceil (\tau-1)m \rceil$ machines and it completes all jobs using LRT.

LEMMA 2 (rate control). *Consider any time interval $[t,t']$, let H be the set of jobs released within $[t,t']$ and transferred to Band 2 within $[t,t']$. Then $\sum_{J\in H} rem(J) \le (\alpha-1)(t'-t)m$.*

Proof. Each job $J \in H$ has been processed by Band 1 for $\frac{1}{\alpha}p(J)$ units of work during the time interval $[t,t']$. Band 1 can perform at most $m(t'-t)$ units of work during $[t,t']$. Thus, $\sum_{J\in H} \frac{1}{\alpha}p(J) \le m(t'-t)$, and $\sum_{J\in H} rem(J) = \sum_{J\in H}(1-\frac{1}{\alpha})p(J) \le (\alpha-1)(t'-t)m.$ ∎

LEMMA 3 (LRT). *For any job J, $F_2(J) \le (\alpha-1) \times \frac{\tau}{\tau-\alpha}F_1(J)$.*

Proof. For any job J, let $t_0 = r(J)$, let t_1 be the time J is transferred from Band 1 to Band 2, and let t_2 be the time J is completed by Band 2. Note that $F_1(J) = t_1 - t_0 \ge p(J)/\alpha$, and $F_2(J) = t_2 - t_1$.

Assume that J waits for a number of time periods in Band 2 before it is completed. Let S be the set of jobs that have ever received processing in Band 2 while J is waiting. For each job $J' \in S$, J' is released no earlier than t_0 (i.e., $r(J') \ge r(J)$), and J' is transferred to Band 2 no later than t_2. Applying Lemma 2 to the interval $[t_0,t_2]$, we have $\sum_{J'\in S} rem(J') \le (\alpha-1)(t_2-t_0)m$.

Whenever J waits in Band 2, all the $\lceil(\tau-1)m\rceil$ machines are processing jobs in

S. The waiting time of J in Band 2 is at most

$$\frac{1}{\lceil(\tau-1)m\rceil}\sum_{J'\in S} rem(J') \le \frac{1}{\lceil(\tau-1)m\rceil}(\alpha-1)(t_2-t_0)m.$$

Therefore,

$$\begin{aligned} F_2(J) &= t_2-t_1 \\ &\le \frac{\alpha-1}{\alpha}p(J) + \frac{\alpha-1}{\lceil(\tau-1)m\rceil}(t_2-t_0)m \\ &\le (\alpha-1)F_1(J) + \frac{\alpha-1}{\tau-1}(F_1(J)+F_2(J)). \end{aligned}$$

Rearranging the last inequality, we have $F_2(J) \le \frac{(\alpha-1)\tau}{\tau-\alpha}F_1(J)$. □

Based on the results that SRPT is 2-speed $\frac{1}{2}$-competitive for flow time [21], and HDF is 4-speed 2-competitive for weighted flow time [6], we can apply Lemma 1 to obtain the following extra-machine competitive results.

COROLLARY 4. *Consider any* $\epsilon > 0$. (i) *The algorithm* `Scatter`(SRPT(2), $2+\epsilon$) *is* $(2+\epsilon)$*-machine* $(1+1/\epsilon)$*-competitive for flow time.* (ii) *The algorithm* `Scatter`(HDF(4), $4+24\epsilon$) *is* $(4+24\epsilon)$*-machine* $(8+1/\epsilon)$*-competitive for weighted flow time.*

3. Transformation that preserves waiting time.

The waiting time of a job is the amount of time the job is waiting for processing before it is completed. Recall that SRPT is α-speed $(1/\alpha)$-competitive for flow time, where $\alpha \ge 2-1/m$ [21]. In the schedule of SRPT(α), the flow time of a job J is exactly $p(J)/\alpha$ plus the waiting time. Thus, SRPT is also α-speed $(1/\alpha)$-competitive for waiting time.

In this section we show how to transform an algorithm A that uses m α-speed machines, where $\alpha \ge 1$ is any real number, to an algorithm `Scatter_&_Squash`($A(\alpha), \tau$) that uses $\lceil \tau m\rceil$ unit-speed machines and incurs a total waiting time comparable to that of $A(\alpha)$. Details are as follows.

LEMMA 5. *Let* $\tau = 7\alpha+5k-2$ *for any integer* $k \ge 1$. *Then, for any job sequence* I, *the total waiting time incurred by* `Scatter_&_Squash`($A(\alpha), \tau$) *is at most* $2L_{A(\alpha)}(I) + \frac{1}{k}L_{OPT}(I)$, *where* $L_{A(\alpha)}(I)$ *and* $L_{OPT}(I)$ *denote the total waiting time incurred by* $A(\alpha)$ *and the optimal algorithm* OPT *using* m *unit-speed machines, respectively.*

We will prove Lemma 5 in section 3.1. Let us consider its implication first. Suppose that A is α-speed $(1/x)$-competitive for waiting time for some $x \ge 1$. Let $k = \lceil x\rceil$ and $\tau = 7\alpha+5\lceil x\rceil-2$. By Lemma 5, `Scatter_&_Squash` gives an $O(\alpha+x)$-machine $(3/x)$-competitive algorithm for waiting time. In other words, based on the result that SRPT is 3-speed (1/3)-competitive for waiting time [21], we immediately obtain a 34-machine 1-competitive algorithm for waiting time. Notice that an algorithm using unit-speed machines is 1-competitive for waiting time if and only if it is 1-competitive for flow time. The competitive ratio of `Scatter_&_Squash` for waiting time can be further reduced to less than one using a more competitive result of SRPT. However, for flow time, the competitive ratio of an algorithm using unit-speed machines is lower bounded by one. The following corollary summarizes these results.

COROLLARY 6. (i) `Scatter_&_Squash` *gives an algorithm that is* 34*-machine* 1*-competitive for flow time.* (ii) *For any integer* $h \ge 1$, `Scatter_&_Squash` *(based on* SRPT($3h$)*) gives an algorithm that is* $(36h-2)$*-machine* $(1/h)$*-competitive for waiting time.*

Algorithm 2. `Scatter_&_Squash`$(A(\alpha), \tau)$, where $\tau = 7\alpha + 5k - 2$ for any integer $k \geq 1$.

Job Release: A newly released job goes to Band 1a.

Band 1a: It uses m machines. Jobs are scheduled according to a simulated copy of $A(\alpha)$. At any time t, if a job J is completed in the simulated $A(\alpha)$, J is transferred to Band 1b if $t \leq r(J) + p(J)$; otherwise, J is transferred to Band 2 (with $AWT(J) = L_{1a}(J)$).

Band 1b: It uses $(2k+1)m$ machines. It runs the algorithm EPPBUSY$\langle 2k+1 \rangle$; i.e., at any time, it arbitrarily selects up to $(2k+1)m$ jobs that are still within their earliest processing periods for execution. At the end of the earliest processing period of a job J, if J is not completed, then J is transferred to Band 2 (with $AWT(J) = L_{1a}(J) + L_{1b}(J)$).

Band 2: It uses $\lceil (7\alpha + 3k - 4)m \rceil$ machines. It runs the MIN-AWT algorithm; i.e., at any time, it greedily schedules the jobs with smallest AWT. A job remains in Band 2 until it is completed.

3.1. The algorithm.

As shown in Algorithm 2, `Scatter_&_Squash` divides the machines into 3 bands called Band 1a, Band 1b, and Band 2, using, respectively, m, $(2k+1)m$, and $\lceil (7\alpha + 3k - 4)m \rceil$ machines, where k is any integer ≥ 1. Similar to the algorithm `Scatter`, Band 1 (comprising Band 1a and Band 1b) only partially processes the jobs, and Band 2 ensures that all jobs get completed. For $i =$ 1a, 1b, 1, or 2, we denote $L_i(J)$ the waiting time of a job J in Band i, and let $L_i(I) = \sum_{J \in I} L_i(J)$. Note that $L_1(J) = L_{1a}(J) + L_{1b}(J)$. Consider any job sequence I and any job J in I. Given an algorithm $A(\alpha)$, `Scatter_&_Squash` aims to guarantee that $L_{1a}(J) = L_{A(\alpha)}(J)$; $L_{1b}(I) \leq \frac{1}{2k} L_{OPT}(I)$; and $L_2(J) \leq L_1(J)$. Then Lemma 5 follows. To achieve $L_2(J) \leq L_1(J)$, we ensure that jobs transferred from Band 1 to Band 2 are easy to schedule in the following sense. Let $rem(J)$ be the remaining work of a job J when J is transferred to Band 2.

(a) *Rate control.* For any time interval T, the sum of $rem(J)$ over all jobs released during T and transferred from Band 1 to Band 2 during T is at most $(\alpha - 1)m|T|$.

(b) *Bounded remaining work.* $rem(J) \leq L_1(J)$.

Band 1a uses the simulation technique presented in the last section. It uses m machines and schedules jobs according to a simulated copy of $A(\alpha)$. When a job J is transferred out of Band 1a, $p(J)/\alpha$ units of its work have been processed, and Band 1a incurs exactly the same waiting time as $A(\alpha)$; i.e., $L_{1a}(J) = L_{A(\alpha)}(J)$. By Lemma 2, Band 1a provides the rate control property.

Define the *earliest processing period* of a job J to be the time interval $[r(J), r(J) + p(J)]$. To achieve the bounded remaining work property, we simply ensure that each job is transferred to Band 2 after its earliest processing period. That is, a job transferred out of Band 1a within its earliest processing period is retained in Band 1b until the end of its earliest processing period.

LEMMA 7. *If a job J is transferred from Band 1 to Band 2 at the end of or after J's earliest processing period, then $rem(J) \leq L_1(J)$.*

Proof. Let $w(J) \geq 0$ be the amount of work done on J in Band 1. J is transferred to Band 2 at $r(J) + w(J) + L_1(J)$, which is at least $r(J) + p(J)$. Thus, $L_1(J) \geq p(J) - w(J) = rem(J)$. ∎

Band 1 as a whole still satisfies the rate control property because jobs released and transferred to Band 2 within an interval T are a subset of jobs released and transferred out of Band 1a within an interval T. The nontrivial part is how to ensure that the waiting time incurred in Band 1b is comparable to $A(\alpha)$ or OPT. To our surprise, we find that Band 1b, using an arbitrary algorithm with $(2k+1)m$ machines to process jobs during their earliest processing periods, would incur a total waiting time of at most $\frac{1}{2k}$ times of OPT. We denote such an algorithm by EPPBUSY$\langle 2k+1\rangle$. Formally speaking, at any time, EPPBUSY considers only jobs that are still in their earliest processing periods, and it arbitrarily selects one such job for each machine. Notice that EPPBUSY may not complete a job J and does not incur waiting time beyond the earliest processing period of J, yet OPT does both. In section 3.2, we will give a careful charging scheme to relate the waiting times of EPPBUSY and OPT. In summary, Band 1 has the following upper bound on waiting time:

$$L_1(I) = L_{1a}(I) + L_{1b}(I) \leq L_{A(\alpha)}(I) + \tfrac{1}{2k} L_{OPT}(I).$$

For Band 2, we want to complete the remaining work of each job J such that $L_2(J)$ is at most $L_1(J)$. In other words, J is allowed to wait in Band 2 up to $L_1(J)$ units of time. To ease our discussion, we assume that each job transferred to Band 2 is associated with an extra parameter $AWT(J)$ representing the allowed waiting time of J, and $AWT(J)$ is set to $L_1(J)$. Based on the properties of rate control and bounded remaining work, we find that MIN-AWT, a greedy strategy that schedules jobs with smallest AWT (ties are broken arbitrarily), can complete each job within its allowed waiting time if Band 2 is given $\lceil (7\alpha + 3k - 4)m \rceil$ machines. The above description of `Scatter_&_Squash` is summarized in Algorithm 2. The rest of this subsection is devoted to proving that for each job J in I, MIN-AWT incurs a waiting time at most $L_1(J)$.

MIN-AWT. Consider a job J that is transferred from Band 1 to Band 2, say, at time $tsf(J)$. Recall that $AWT(J)$ is set to $L_1(J)$, and the remaining work of J at $tsf(J)$, denoted $rem(J)$, is at most $AWT(J)$. We want to show that if Band 2 uses MIN-AWT on $O(\alpha + k)$ machines, then J waits no more than $AWT(J)$ units of time in Band 2, or, equivalently, J is completed by the time $d(J) = tsf(J) + rem(J) + AWT(J)$. We call $d(J)$ the deadline of J in Band 2.

We use induction to show that every job is completed by its deadline. Consider jobs in increasing order of deadlines. Let J be a job. Assume that all jobs with deadline earlier than J are completed by their deadlines. We focus on the total waiting time of J up to $d(J)$. During $[tsf(J), d(J)]$, whenever J is waiting, all machines in Band 2 are processing jobs J' with the following properties:

1. $AWT(J') \leq AWT(J)$;
2. J' is transferred to Band 2 no earlier than $tsf(J) - 2AWT(J)$ (otherwise, $d(J') = tsf(J') + rem(J') + AWT(J') < tsf(J) < d(J)$ and J' is completed before $tsf(J)$); and
3. J' is transferred to Band 2 no later than $d(J)$.

Let S be the set of all jobs J' satisfying the above properties. Below we upper bound the sum of $rem(J')$ over all J' in S.

LEMMA 8. $\sum_{J' \in S} rem(J') \leq \delta m \, AWT(J)$, *where* $\delta = 7\alpha + 3k - 4$.

Proof. Let $t_1 = tsf(J) - 2AWT(J)$. By definition, jobs in S are transferred to Band 2 within $[t_1, d(J)]$. Let $t_0 = t_1 - xAWT(J)$ for some $x > 1$. We divide the jobs in S according to their release time (to Band 1a). Let $S_1 = \{J' \in S \mid r(J') \geq t_0\}$ and $S_2 = S - S_1$.

Jobs in S_1. We use the rate control property to bound the sum of $rem(J')$ over all J' in S_1. For each job J' in S_1, J' is released during the time interval $[t_0, d(J)]$ and is transferred to Band 2 during $[t_1, d(J)]$. Recall that $t_0 < t_1$. By the rate control property, $\sum_{J' \in S_1} rem(J')$ is at most $(\alpha - 1)m(d(J) - t_0) \leq (\alpha - 1)m(x + 4)AWT(J)$.

Jobs in S_2. In this case, we exploit the bounded remaining work property and the fact that $AWT(J)$ is set to $L_1(J)$. Each job J' in S_2 is released before t_0 and transferred to Band 2 on or after t_1. Thus, J' is kept in Band 1 for a period of length at least $t_1 - t_0 \geq x\, AWT(J)$. Note that $L_1(J')$ (i.e., the waiting time of J' in Band 1) $= AWT(J') \leq AWT(J)$. Thus, J' is processed by Band 1 for at least $(x-1)\, AWT(J)$ units of work from t_0 to t_1. Band 1 has only $(2k+2)m$ machines and it performs at most $(2k+2)m(xAWT(J))$ units of work from t_0 to t_1. Thus,

$$\begin{aligned}(2k+2)mx\, AWT(J) &\geq \sum_{J' \in S_2} (x-1)AWT(J)\\ &\geq \sum_{J' \in S_2} (x-1)AWT(J')\\ &\geq \sum_{J' \in S_2} (x-1)rem(J').\end{aligned}$$

So, $\sum_{J' \in S_2} rem(J')$ is at most $(2k+2)m\frac{x}{x-1}AWT(J)$. In conclusion, $\sum_{J' \in S} rem(J') \leq [(\alpha - 1)(x + 4) + (2k + 2)\frac{x}{x-1}]m\, AWT(J)$. Putting $x = 3$, we obtain Lemma 8. □

We are ready to prove that J can be completed by $d(J)$. Whenever J is waiting in Band 2 during $[tsf(J), d(J)]$, all machines of Band 2 are processing jobs belonging to the set S, and the sum of $rem(J')$ over all jobs $J' \in S$ is at most $(7\alpha + 3k - 4)m\, AWT(J)$. Band 2 uses $\lceil(7\alpha + 3k - 4)m\rceil$ machines, and the work due to S can keep J waiting in Band 2 for at most $AWT(J)$ units of time. Thus, J is completed by $d(J)$.

3.2. Analysis of EPPBUSY.

`Scatter_&_Squash` uses the algorithm EPPBUSY in Band 1b. To upper bound the waiting time incurred in Band 1b, we first study in this section the waiting time incurred by EPPBUSY when it is used to process a sequence of jobs. This result may be of independent interest.

EPPBUSY$\langle h \rangle$ uses hm machines for any integer $h \geq 2$. It schedules a job only within its earliest processing period and it may not be able to finish each job. Let OPT be the optimal scheduler, which uses m machines to process all jobs to completion and minimizes the total waiting time.

For any job sequence I, let $P(I)$ be the schedule produced by EPPBUSY$\langle h \rangle$ on I, and, similarly, $OPT(I)$ for OPT. Note that a job remains in $P(I)$ only during its earliest processing period, while a job remains in $OPT(I)$ until it is completed. We want to show that the total waiting time of jobs in $P(I)$ is at most $\frac{1}{h-1}$ of that of $OPT(I)$.

We first focus on the schedule $P(I)$. $P(I)$ may contain one or more waiting periods (a waiting period is a period in which at least one job is waiting at any time). Denote these waiting periods as $\lambda_1 = [t_1, t_1'], \lambda_2 = [t_2, t_2'], \lambda_3 = [t_3, t_3'], \ldots$, where $t_1 < t_1' < t_2 < t_2' < t_3 < t_3' < \cdots$. Let $|\lambda_i| = t_i' - t_i$. Note that $P(I)$ accumulates waiting time only during the waiting periods.

DEFINITION 9. *Let* $S = \{\lambda_u, \lambda_{u+1}, \ldots, \lambda_v\}$ *be a collection of consecutive waiting periods. Recall that* h *is the parameter required by EPPBUSY*$\langle h \rangle$. S *is said to be* h*-close if* $t_{u+1} \leq t_u + h|\lambda_u|$, $t_{u+2} \leq t_u + h(|\lambda_u| + |\lambda_{u+1}|), \ldots,$ *and* $t_v \leq t_u + h\sum_{i=u}^{v-1} |\lambda_i|$.

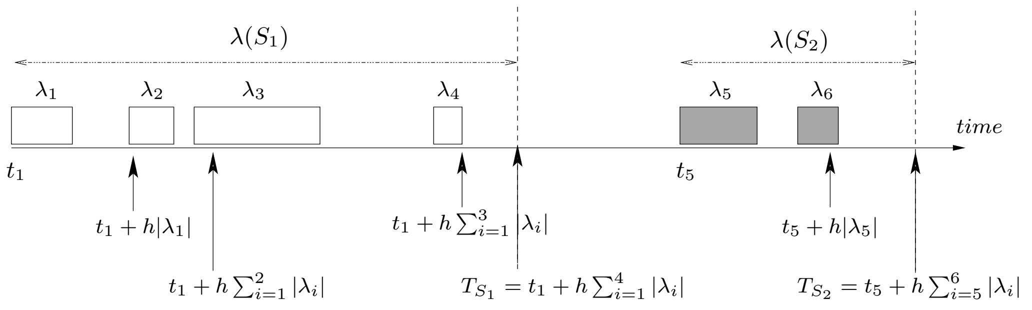
$\lambda(S_1)$
$\lambda(S_2)$
λ_1
λ_2
λ_3
λ_4
λ_5
λ_6
time
t_1
t_5
$t_1 + h|\lambda_1|$
$t_1 + h\sum_{i=1}^{2}|\lambda_i|$
$t_1 + h\sum_{i=1}^{3}|\lambda_i|$
$T_{S_1} = t_1 + h\sum_{i=1}^{4}|\lambda_i|$
$t_5 + h|\lambda_5|$
$T_{S_2} = t_5 + h\sum_{i=5}^{6}|\lambda_i|$

FIG. 1. *Two h-close collections of waiting periods ($h = 2$ in this example).*

Furthermore, define t_S to be the time $(t_u + h\sum_{i=u}^{v}|\lambda_j|)$, and define $\lambda(S)$ to be the interval $[t_u, t_S]$. Note that $|\lambda(S)| = h\sum_{j=u}^{v}|\lambda_j|$. See Figure 1 for an example.

FACT 10. *For any $u \le i \le v$, $t_S - t_i \ge h(|\lambda_i| + |\lambda_{i+1}| + \cdots + |\lambda_v|)$.*

We partition the waiting periods in $P(I)$ into *maximal* h-close collections $S_1 = \{\lambda_1, \lambda_2, \ldots, \lambda_{k_1}\}$, $S_2 = \{\lambda_{k_1+1}, \lambda_{k_1+2}, \ldots, \lambda_{k_2}\}$, ... That is, the next waiting period beyond each S_i has a starting time greater than t_{S_i}. The notion of a maximal h-close collection of waiting periods defines a framework for our analysis. In the following, we show that for each maximal h-close collection S of waiting periods, the waiting time incurred by $P(I)$ within the interval $\lambda(S)$ is at most a factor of $1/(h-1)$ of the waiting time incurred by $OPT(I)$ within $\lambda(S)$.

The following notion further provides a tool for lower bounding the waiting time of $OPT(I)$.

DEFINITION 11. *Consider any interval $\lambda = [t, t']$. Let J be a job with required work $p(J)$. If λ is enclosed in the earliest processing period of J, the work required by J can be partitioned into three chunks of size $t - r(J)$, $t' - t$, and $p(J) - (t' - r(J))$, respectively. The middle chunk is referred to as the λ-work of J. In general, when λ is not enclosed in the earliest processing period of J, we let $\lambda' = \lambda \cap [r(J), r(J)+p(J)]$ and define the λ-work of J to be its λ'-work.*

We denote the size of the λ-work of J as $W(J,\lambda)$, i.e., $W(J,\lambda) = |\lambda \cap [r(J), r(J) + p(J)]|$. A fact useful to our analysis is that if $W(J,\lambda) > 0$, the earliest time $OPT(I)$ (or any schedule using unit-speed machines) can start processing the λ-work of J is $\max\{t, r(J)\}$.

Let $S = \{\lambda_u, \lambda_{u+1}, \ldots, \lambda_v\}$ be a maximal h-close collection of waiting periods. Let J be any job. Consider the λ_i-work of J for all $\lambda_i \in S$. Below, we give a way to mark the earliest possible schedule of the λ_i-work of J in $OPT(I)$. Let $\lambda_i = [t_i, t_i']$ be the first waiting period of S that overlaps with the earliest processing period of J (i.e., $W(J,\lambda_i) > 0$). Note that OPT cannot process the λ_i-work of J earlier than t_i or $r(J)$. We mark the first $W(J,\lambda_i)$ units of work starting from the time $\max\{t_i, r(J)\}$ in the schedule of J in $OPT(I)$. For each subsequent $j > i$, if the λ_j-work of J is nonnull, we identify, in the schedule of J in $OPT(I)$, the first time $t \ge t_j$ when no work has been marked, and we mark another $W(J,\lambda_j)$ units of work starting from t. We have the following lemma on the work marked on the schedule of J in $OPT(I)$.

Within the time interval $\lambda(S)$, we denote the waiting time of J incurred by $P(I)$ as $L_P(J)|_{\lambda(S)}$, and similarly $L_{OPT}(J)|_{\lambda(S)}$ for $OPT(I)$.

LEMMA 12. *Suppose that in the course of marking all the λ_i-work of a job J in $OPT(I)$, a total of y units of work are marked beyond t_S. Then $L_{OPT}(J)|_{\lambda(S)}$ is at least $(h-1)y$.*

Proof. Assume that $\lambda_i = [t_i, t'_i]$ is the first waiting period in S such that part of the λ_i-work of J is marked beyond t_S. Then, $y \le |\lambda_i| + |\lambda_{i+1}| + \cdots + |\lambda_v|$. In $OPT(I)$, the λ_i-work of J is not completed by time t_S. Thus, within $\lambda(S)$, the waiting time of J is at least $t_S - t'_i = t_S - t_i - |\lambda_i|$. By Fact 10, we conclude that $L_{OPT}(J)|_{\lambda(S)} \ge h(|\lambda_i| + |\lambda_{i+1}| + \cdots + |\lambda_v|) - |\lambda_i| \ge (h-1)y$. □

LEMMA 13. $\sum_{J\in I} L_P(J)|_{\lambda(S)} \le \frac{1}{h-1} \sum_{J\in I} L_{OPT}(J)|_{\lambda(S)}$.

Proof. With respect to $P(I)$, the total waiting time of all jobs during a waiting period λ_i is exactly the total length of the λ_i-work of all jobs minus the amount of work that EPPBUSY$\langle h\rangle$ processes during λ_i. That is, $\sum_{J\in I} L_P(J)|_{\lambda_i} = \sum_{J\in I} W(J,\lambda_i) - hm|\lambda_i|$. Summing over all waiting periods in S, we have

$$\begin{aligned}\sum_{J\in I} L_P(J)|_{\lambda(S)} &= \sum_{i=u}^{v}\sum_{J\in I} W(J,\lambda_i) - \sum_{i=u}^{v} hm|\lambda_i| \\ &= \sum_{i=u}^{v}\sum_{J\in I} W(J,\lambda_i) - m|\lambda(S)|.\end{aligned}$$

Note that $\sum_{J\in I} L_P(J)|_{\lambda(S)} \ge 0$, and hence $\sum_{i=u}^{v}\sum_{J\in I} W(J,\lambda_i) \ge m|\lambda(S)|$.

Since OPT has only m machines, during $\lambda(S)$, OPT can process at most $m|\lambda(S)|$ units of work. Consider the λ_i-work of all jobs over all λ_i in S. Their total size is $\sum_{i=u}^{v}\sum_{J\in I} W(J,\lambda_i)$, which exceeds $m|\lambda(S)|$. Thus, not all λ_i-work can be marked within $\lambda(S)$ in $OPT(I)$. The total amount of work marked beyond t_S in $OPT(I)$ is at least $\sum_{i=u}^{v}\sum_{J\in I} W(J,\lambda_i) - m|\lambda(S)| = \sum_{J\in I} L_P(J)|_{\lambda(S)}$. By Lemma 12, $L_{OPT}(J)|_{\lambda(S)}$ is at least the total amount of λ_i-work marked beyond t_S for J. Thus, $\sum_{J\in I} L_{OPT}(J)|_{\lambda(S)}$ is at least $(h-1)\sum_{J\in I} L_P(J)|_{\lambda(S)}$. □

COROLLARY 14. *For any job sequence I, let $L_P(I)$ be the total waiting time incurred by EPPBUSY$\langle h\rangle$ and $L_{OPT}(I)$ be that for OPT. Then, $L_P(I) \le \frac{1}{h-1} L_{OPT}(I)$.*

EPPBUSY in Scatter_&_Squash. We are now ready to analyze the waiting time incurred by Band 1b of `Scatter_&_Squash`. Recall that Band 1b uses $(2k+1)m$ machines to run EPPBUSY$\langle 2k+1\rangle$. We want to prove that for any job sequence I, the total waiting time incurred in Band 1b of `Scatter_&_Squash` is at most $1/(2k)L_{OPT}(I)$.

By definition of `Scatter_&_Squash`, a job J in I is transferred to Band 1b only after it is partially scheduled in Band 1a. Thus, J remains in Band 1b only during a subinterval of its earliest processing period.

Let us compare the schedule of Band 1b with the schedule of I when using a stand-alone copy of EPPBUSY$\langle 2k+1\rangle$. Denote the latter schedule Φ. At any time t, if a job J remains in Band 1b, then t is within J's earliest processing period, and J remains in Φ for possible processing. Thus, jobs remaining in Band 1b are a subset of jobs remaining in Φ. As both Band 1b and the stand-alone EPPBUSY$\langle 2k+1\rangle$ are using $(2k+1)m$ machines, the number of jobs waiting in Band 1b, denoted $\#_{1b}(I,t)$, is at most the number of jobs waiting in Φ, denoted $\#_\Phi(I,t)$.

Let $L_{1b}(J)$ and $L_\Phi(J)$ be the waiting times of J in the schedules of Band 1b and Φ, respectively. We have

$$\sum_{J\in I} L_{1b}(J) = \int \#_{1b}(I,t)dt \le \int \#_\Phi(I,t)dt = \sum_{J\in I} L_\Phi(J) \le \frac{1}{2k}\sum_{J\in I} L_{OPT}(J).$$

4. Extension to weighted waiting time and stretch. The normalized waiting time of a job refers to the waiting time divided by the processing time. An algorithm is $O(1)$-machine 1-competitive for stretch if and only if it is $O(1)$-machine 1-competitive for normalized waiting time. To derive an $O(1)$-machine 1-competitive algorithm for stretch, we want `Scatter_&_Squash` to transform a given α-speed algorithm $A(\alpha)$ to an $O(\alpha)$-machine algorithm that preserves the normalized waiting time. In fact, `Scatter_&_Squash` can even be extended to preserve the weighted waiting time when every job is given an arbitrary weight. The idea is quite simple. By the definition of `Scatter_&_Squash` (in section 3.1), the performance guarantee for Band 1a and Band 2 is based on the (unweighted) waiting time of each job, and it remains the same when weighted waiting time is concerned. As a whole, Band 1a still incurs the same amount as $A(\alpha)$ does, and Band 2 incurs no more than Band 1 does. Only Band 1b requires modification to cater to the weighted setting.

Before looking at the details of Band 1b, we prove a lemma that can transform a special relationship of the unweighted waiting times of two schedules into a relationship of their total weighted waiting times. Below, x_i and y_i denote the waiting time of a job in two schedules, and w_i is the weight of the job.

LEMMA 15. *Let $x_1, x_2, \dots, x_r$ and $y_1, y_2, \dots, y_r$ be two sequences of nonnegative reals. Let γ be any positive real. Suppose that $\sum_{i=1}^{q} x_i \le \gamma \sum_{i=1}^{q} y_i$ for all $q = 1, \dots, r$. Then for any nondecreasing sequence of positive reals $w_1 \ge w_2 \ge \cdots \ge w_r$, we have $\sum_{i=1}^{r} w_i x_i \le \gamma \sum_{i=1}^{r} w_i y_i$.*

Proof. We prove the lemma by induction on r. The case for $r = 1$ is obvious. Assume that the lemma is true when $r = z$, for some integer $z \ge 1$. When $r = z+1$ we consider the following two cases.

Case 1. If $x_{z+1} \le \gamma y_{z+1}$, then $\sum_{i=1}^{z+1} w_i x_i = \sum_{i=1}^{z} w_i x_i + w_{z+1} x_{z+1} \le \gamma \sum_{i=1}^{z} w_i y_i + \gamma w_{z+1} y_{z+1} = \gamma \sum_{i=1}^{z+1} w_i y_i$.

Case 2. Otherwise, $x_{z+1} > \gamma y_{k+1}$. Let $\delta = x_{z+1} - \gamma y_{z+1}$.

$$\begin{aligned}\sum_{1\le i\le z+1} w_i x_i &= \sum_{1\le i<z} w_i x_i + w_z x_z + w_{z+1}(\gamma \times y_{z+1} + \delta)\\ &= \sum_{1\le i<z} w_i x_i + w_z x_z + w_{z+1}\delta + w_{z+1}\gamma y_{z+1}\\ &\le \sum_{1\le i<z} w_i x_i + w_z(x_z+\delta) + \gamma w_{z+1} y_{z+1}.\end{aligned}$$

Define the sequence $(d_1, d_2, \dots, d_z)$ such that $d_i = x_i$ for $i = 1, \dots, z-1$ and $d_z = x_z + \delta$. For any $q = 1, \dots, z-1$, $\sum_{i=1}^{q} d_i = \sum_{i=1}^{q} x_i \le \gamma \sum_{i=1}^{q} y_i$. For $q = z$,

$$\sum_{1\le i\le z} d_i = \sum_{1\le i\le z} x_i + \delta = \sum_{1\le i\le z+1} x_i - \gamma y_{z+1} \le \gamma \sum_{1\le i\le z+1} y_i - \gamma y_{z+1} = \gamma \sum_{1\le i\le z} y_i.$$

Applying the induction hypothesis to d_i and y_i, we have $\sum_{i=1}^{z} w_i d_i \le \gamma \sum_{i=1}^{z} w_i y_i$. Thus, we have $\sum_{i=1}^{z+1} w_i x_i \le \gamma \sum_{i=1}^{z} w_i y_i + \gamma w_{z+1} y_{z+1} = \gamma \sum_{i=1}^{z+1} w_i y_i$. The induction is complete. ∎

In `Scatter_&_Squash`, Band 1b uses an arbitrary algorithm with $(2k+1)m$ machines to process jobs during their earliest processing periods. We enhance Band 1b by selecting jobs with largest weights. We call this new algorithm EPPHWF (earliest processing period, highest weight first), and denote it as EPPHWF$\langle h\rangle$ when it is equipped with hm processors, where h is an integer at least 2. Intuitively, jobs with large weights will wait less. Let OPT be the optimal algorithm (using m unit-speed machines) for minimizing weighted flow time. Our key observation is that for any job weight w, we can bound the total waiting time of all jobs with weight at least w in EPPHWF$\langle 2k+1\rangle$ to be at most $1/(2k)$ times that of OPT. Then, we can make use of Lemma 15 inductively to show that EPPHWF$\langle 2k+1\rangle$ incurs a total weighted waiting time at most $1/(2k)$ times that of OPT. Details are as follows.

LEMMA 16. *Let $h \geq 2$ be an integer. For any job sequence I, the total weighted waiting time incurred by* EPPHWF$\langle h\rangle$ *is at most $\frac{1}{h-1}$ times that of OPT.*

Proof. Consider any job sequence I. We compare the schedules of EPPHWF$\langle h\rangle$ and OPT. Let $w_1 > w_2 \cdots > w_r$ be the distinct weights of the jobs in I. Consider any integer $q \in \{1, 2, \dots, r\}$. With respect to the schedule of EPPHWF$\langle h\rangle$, let $L(I)|_{w_q}$ be the total *unweighted* waiting time incurred on jobs with weight exactly w_q. $L_{OPT}(I)|_{w_q}$ is defined similarly for the schedule of OPT. Note that for jobs with weight at least w_q, the total weighted waiting time incurred by EPPHWF$\langle h\rangle$ and OPT is $\sum_{i=1}^{q} w_i L(I)|_{w_i}$ and $\sum_{i=1}^{q} w_i L_{OPT}(I)|_{w_i}$, respectively.

We first focus on the unweighted waiting time. Let $I_q \subseteq I$ be the set of jobs having weight at least w_k, where $q \in \{1, \dots, r\}$. EPPHWF$\langle h\rangle$ does not change the schedule of the jobs in I_q when jobs with less weight are removed, so EPPHWF$\langle h\rangle$, when scheduling I_q alone, incurs a total (unweighted) waiting time of $\sum_{i=1}^{q} L(I)|_{w_i}$. EPPHWF is a special case of EPPBUSY, so by Corollary 3.2, we have $\sum_{i=1}^{q} L(I)|_{w_i} \leq \frac{1}{h-1} L_{OPT(I_q)}$, where $L_{OPT(I_q)}$ is the total waiting time in the optimal (unweighted) schedule for I_q. When scheduling I, the waiting time incurred by OPT on the jobs in I_q is $\sum_{i=1}^{q} L_{OPT}(I)|_{w_i}$, which is at least $L_{OPT(I_q)}$. Therefore, for each $q = 1, \dots, r$,

$$\sum_{i=1}^{q} L(I)|_{w_i} \leq \frac{1}{h-1} \sum_{i=1}^{q} L_{OPT}(I)|_{w_i}.$$

By Lemma 15, we transform the above relation of unweighted waiting times to a weighted version:

$$\sum_{i=1}^{r} w_i L(I)|_{w_i} \leq \frac{1}{h-1} \sum_{i=1}^{r} w_i L_{OPT}(I)|_{w_i}.$$

Note that the former is the weighted waiting time of EPPHWF$\langle h\rangle$, and $\sum_{i=1}^{r} w_i L_{OPT} \cdot (I)|_{w_i}$ is the weighted waiting time of OPT. The lemma follows. ∎

Transformation that preserves weighted waiting time. Let `weighted_SS` be the algorithm `Scatter_&_Squash` with Band 1b using EPPHWF instead of EPPBUSY. Given an α-speed algorithm $A(\alpha)$, `weighted_SS` transforms $A(\alpha)$ to a τ-machine algorithm (recall that $\tau = 7\alpha + 5k - 2$, where $k \geq 1$). Band 1a and Band 2 have the same performance as before. Specifically, for each job, Band 1a still incurs a waiting time (and weighted waiting time) the same as $A(\alpha)$, and Band 2 incurs a waiting time (and weighted waiting time) no more than Band 1 does. By Lemma 16, the total weighted waiting time incurred by Band 1b, which uses $(2k+1)$ machines, is at most $\frac{1}{2k}$ times that of OPT. Thus, the total weighted waiting time has the same bound as before.

LEMMA 17. *Let $\tau = 7\alpha + 5k - 2$ for any integer $k \geq 1$. The weighted waiting time incurred by* `weighted_SS`$(A(\alpha), \tau)$ *is at most* 2 *times that of $A(\alpha)$ plus $1/k$ times that of OPT.*

Suppose there is an algorithm A that is $O(s)$-speed $\frac{1}{s}$-competitive for weighted waiting time, for any $s \geq 1$. Then by Lemma 17, we can derive an algorithm that is

$O(1)$-machine 1-competitive for weighted waiting time or, equivalently, for weighted flow time. However, such an algorithm A is not known to exist. Even if we restrict our attention to normalized waiting time, we do not know any algorithm that is $O(s)$-speed $\frac{1}{s}$-competitive and that can be used to derive an $O(1)$-machine 1-competitive algorithm for stretch.

Alternative transformation that preserves weighted waiting time. The rest of this paper shows another way to obtain an algorithm that is $O(1)$-machine 1-competitive for stretch. In the next section, we will show a weaker result on using a faster processor to improve the normalized waiting time; specifically, we prove that an algorithm based on SJF is $(1/s)$-competitive when using 4m machines that are $8s$-speed, where $s \geq 1$. To ease our discussion, we say this algorithm is (4-machine, $8s$-speed) $\frac{1}{s}$-competitive. (Note that this result does not imply an algorithm that is $32s$-speed $\frac{1}{s}$-competitive for normalized waiting time.[2]) Furthermore, we can extend the transformation result in Lemma 17 so that the input algorithm to `weighted_SS`, denoted $A[\ell, \alpha]$, uses ℓm machines that are α-speed, where $\ell \geq 1$ is an integer. In this case, Band 1a uses ℓm machines, Band 1b uses $(2k+1)m$ machines, and Band 2 uses $\lceil 7\ell(\alpha-1)m + \frac{3}{2}(\ell+2k+1)m \rceil$ machines. The proof of Lemma 5 can be easily generalized to show that

- for each job, Band 1a still incurs a weighted waiting time the same as $A[\ell, \alpha]$ does, and Band 2 incurs a weighted waiting time no more than Band 1 does; and
- the total weighted waiting time incurred by Band 1b is at most $\frac{1}{2k}$ times that of OPT.

Thus, we have the following result.

LEMMA 18. *Let $\tau = 7\alpha\ell + 5k - \frac{9}{2}\ell + \frac{5}{2}$, where k is any positive integer. The weighted waiting time incurred by* `weighted_SS`$(A[\ell, \alpha], \tau)$ *is at most* 2 *times that of* $A[\ell, \alpha]$ *plus* $1/k$ *times that of* OPT.

In the next section, we show that, for the special case of normalized waiting time, SJF is (4-machine, $8s$-speed) $\frac{1}{s}$-competitive for any $s \geq 1$. Choosing s to be 2.2 and applying Lemma 18 with $k = 11$, $\ell = 4$, and $\alpha = 8 \times 2.2$, we obtain an algorithm that is 533-machine 1-competitive for normalized waiting time, as well as the following result on stretch.

COROLLARY 19. *Based on SJF,* `weighted_SS` *gives a* 533*-machine* 1*-competitive algorithm for stretch.*

Remark. We conjecture that HDF is ($O(1)$-machine, $O(s)$-speed) $\frac{1}{s}$-competitive for weighted waiting time, for any $s \geq 1$. If this can be proven, then Lemma 18 can transform HDF to an algorithm that is $O(1)$-machine 1-competitive for weighted waiting time, as well as for weighted flow time.

5. Improved analysis of SJF with faster machines.

This section proves that SJF is (4-machine, $8s$-speed) $\frac{1}{s}$-competitive for normalized waiting time, where $s \geq 1$. The proof is divided into two parts. First, we show how to make use of a result by Becchetti et al. [6] to show that SJF is (2-machine, 4-speed) 2-competitive for normalized waiting time. Next, we show a scaling lemma for SJF by which the waiting time of each job can be reduced by s times if a double number of s times faster machines are used. That is, for any $s \geq 1$, SJF$[2\ell, s\alpha]$, when compared with SJF$[\ell, \alpha]$, reduces the waiting time of every job by at least s times, where $s \geq 1$.

[2]Roughly speaking, if we use a four-times faster machine to simulate four unit-speed machines by time sharing, the flow time of each job is preserved, but the actual time to process a job is shortened by four times. Thus, the waiting time is longer than before.

Then, the fact that SJF is (2-machine, 4-speed) 2-competitive for normalized waiting time would imply that SJF is also (4-machine, $8s$-speed) $\frac{1}{s}$-competitive for normalized waiting time, for any $s \geq 1$.

5.1. SJF and normalized waiting time. First, we observe the following result by Becchetti et al. [6] on using HDF(4) (i.e., HDF with m 4-speed machines) to schedule jobs with arbitrary weights. Below the notation HDF (or any algorithm) is overloaded to mean the algorithm itself as well as the schedule defined by HDF.

LEMMA 20 (see [6]). *Let I be a job sequence with arbitrary weights. Let A be any schedule of I using m unit-speed machines. Consider the two schedule A and the schedule of I in accordance with HDF(4). At any time t, the total weight of the remaining jobs in HDF(4) is at most two times that of A.*

The above lemma implies that in the unweighted setting, at any time, the number of remaining jobs in SJF(4) is at most two times that of A. Furthermore, we have the following result.

LEMMA 21. SJF *is (2-machine, 4-speed) 2-competitive for (unweighted) waiting time.*

Proof. Consider any job sequence I. Let A be any schedule of I using m unit-speed machines. At any time t, let $U_t(\text{SJF}[2,4])$ be the number of jobs remaining in SJF$[2,4]$, and define $U_t(\text{SJF}(4))$ and $U_t(A)$ similarly. Then, $U_t(\text{SJF}[2,4]) \leq U_t(\text{SJF}(4)) \leq 2 \times U_t(A)$. Note that SJF$[2,4]$ is using $2m$ machines. At time t, if there is a job waiting in SJF$[2,4]$, then $U_t(\text{SJF}[2,4]) > 2m$, and the number of jobs waiting is $U_t(\text{SJF}[2,4]) - 2m \leq 2(U_t(A) - m)$. Thus, at any time, the number of jobs waiting in SJF$[2,4]$ at most two times of that of A, and the lemma follows. ∎

Intuitively, SJF gives priority to smaller jobs, and it is competitive not only for the total waiting time, but also for the waiting time of small jobs only. This allows us to derive inductively a bound of the waiting time. Then, using Lemma 15, we transform this bound to a bound on the total normalized waiting time. Details are as follows.

LEMMA 22. SJF *is (2-machine, 4-speed) 2-competitive for normalized waiting time.*

Proof. Let I be any job sequence, and let $w_1 < w_2 \cdots < w_r$ be the distinct job sizes in I. Consider the schedule of I in accordance with SJF$[2,4]$, and let $L(I)|_{w_i}$ be the total waiting time of jobs with size exactly w_i. Denote OPT be the optimal algorithm using m unit-speed machines, and define $L'(I)|_{w_i}$ similarly for OPT. Consider any $q \in \{1, 2, \dots, r\}$. Let I_q be the job sequence including only jobs in I with size w_1, w_2, $\dots$, or w_q. Since SJF does not change the schedule of a job due to other jobs with larger size, $\sum_{i=1}^{q} L(I_q)|_{w_i}$ (i.e., the total waiting time incurred by SJF$[2,4]$ on I_q) is exactly equal to $\sum_{i=1}^{q} L(I)|_{w_i}$. By Lemma 21, the total waiting time incurred by SJF$[2,4]$ is at most two times the total waiting time incurred by any schedule of I_q on m unit-speed machines. Thus, $\sum_{i=1}^{q} L(I)|_{w_i} = \sum_{i=1}^{q} L(I_q)|_{w_i} \leq 2 \times \sum_{i=1}^{q} L'(I)|_{w_i}$, where $q = 1, 2, \dots, r$.

Again, we make use of Lemma 15 to turn the above result into a weighted one: $\sum_{i=1}^{r} \frac{1}{w_i} L_s(I)|_{w_i} \leq 2 \sum_{i=1}^{r} \frac{1}{w_i} L_o(I)|_{w_i}$, or, equivalently, the normalized waiting time of SJF$(2,4)$ is at most two times that of OPT. ∎

5.2. The scaling lemma. This section shows that the waiting time incurred by SJF can be scaled down with increasing speed. Precisely, we compare SJF$[\ell, \alpha]$ and SJF$[2\ell, c\alpha]$, where $c \geq 1$ is a real, and we show that the waiting time of each job decreases by c times (note that this result is much stronger than bounding the total waiting time). More interestingly, this result is true for the more general algorithm

HDF (see the lemma below). Note that the density of a job, defined as the ratio of its weight to its processing time, is fixed throughout the life span of a job. HDF schedules jobs with highest densities (we assume that ties are broken by job IDs). SJF is identical to HDF when all jobs are assumed to have a unit weight.

LEMMA 23. *Consider any job sequence I with arbitrary weights. For each job J in I, denote the waiting time of J incurred by* HDF$[2\ell, c\alpha]$ *and* HDF$[\ell, \alpha]$ *as $L_{(2,c)}(J)$ and $L_{(1,1)}(J)$, respectively. Then $L_{(2,c)}(J) \leq \frac{1}{c}L_{(1,1)}(J)$.*

Proof. Denote $S_1(I)$ and $S_2(I)$ as the schedule of a job sequence I in accordance with HDF$[\ell, \alpha]$ and HDF$[2\ell, c\alpha]$, respectively. By definition of HDF, at any time, every job has less remaining work in HDF$[2\ell, c\alpha]$ than in HDF$[\ell, \alpha]$, and if a job is waiting in HDF$[2\ell, c\alpha]$, then the job must also be waiting in HDF$[\ell, \alpha]$.

Consider a job J in I. Assume that J is completed at time $z(J)$ in $S_1(I)$, and hence no later than $z(J)$ in $S_2(I)$. We call jobs in I that have higher densities than J the *higher-priority* jobs. At any time, $S_1(I)$ (as well as $S_2(I)$) is said to be *busy* if all available machines are running some higher-priority jobs. Note that at any time t in $[r(J), z(J)]$, J is waiting in $S_1(I)$ if and only if $S_1(I)$ is busy. So the waiting time of J in $S_1(I)$ is the total length of the busy periods of $S_1(I)$ during $[r(J), z(J)]$. Denote these busy periods $\lambda_1, \lambda_2, \ldots, \lambda_h$.

Next, we consider the schedule $S_2(I)$. Suppose that J is waiting in $S_2(I)$ during a time interval ρ. Note that ρ is a busy period in $S_2(I)$; furthermore, J is also waiting in $S_1(I)$ during ρ, and ρ is a subinterval of some busy period λ of $S_1(I)$. Therefore, to upper bound the waiting time of J in $S_2(I)$, it suffices to consider each busy period λ of $S_1(I)$ separately. Below we show that the busy period of $S_2(I)$ within λ has a total length at most a fraction $\frac{1}{c}$ of λ (see the lemma below). Then we can conclude that the waiting time of J in $S_2(I)$, which is the total length of the busy periods within $\lambda_1, \lambda_2, \ldots, \lambda_h$, is at most $\frac{1}{c}(|\lambda_1| + |\lambda_2| + \cdots + |\lambda_h|)$ or, equivalently, $\frac{1}{c}$ times the waiting time of J in $S_1(I)$. Lemma 23 follows. ◻

It remains to prove the following lemma.

LEMMA 24. *Let $\lambda = [t_1, t_2]$ be a busy period in $S_1(I)$. Denote the busy periods of $S_2(I)$ that are within $[t_1, t_2]$ as $\rho_1, \rho_2, \ldots, \rho_g$, and let $y = \sum_{1 \leq i \leq g} |\rho_i|$ be their total length. Then $y \leq \frac{1}{c}|\lambda|$.*

Proof. Let $\lambda = [t_1, t_2]$. The work scheduled by $S_1(I)$ during λ, denoted by W, has a total size exactly $\ell m \alpha |\lambda|$. Let R be the high-priority jobs remaining in $S_1(I)$ immediately after t_2. Note that R contains at most $\ell m - 1$ jobs as $S_1(I)$ is not busy immediately after t_2.

During the busy periods $\rho_1, \rho_2, \ldots, \rho_g$, the total amount of work processed in $S_2(I)$ is $2\ell m c \alpha y$; on the other hand, the high-priority work available to $S_2(I)$ is limited. In particular, $S_2(I)$ can process at most all the work W and some of the work of R (that are processed beyond t_2 in $S_1(I)$). The former has a total amount of $\ell m \alpha |\lambda|$, and the latter is bounded by $c\alpha y |R| < c\alpha y \ell m$. Thus, we have

$$2\ell m c \alpha y < \ell m \alpha |\lambda| + c \alpha y \ell m$$

or, equivalently, $y \leq |\lambda|/c$. ◻

Lemma 23 implies that when comparing HDF$[2\ell, c\alpha]$ against HDF$[\ell, \alpha]$, the weighted waiting time, flow time, and weighted flow time of each job decrease by c times. In particular, together with Lemma 22, Lemma 23 gives the main result of this section.

COROLLARY 25. SJF *is* (4*-machine,* $8s$*-speed)* $\frac{1}{s}$*-competitive for normalized waiting time, where $s \geq 1$ is any real.*

Proof. By Lemma 22, SJF is (2-machine, 4-speed) 2-competitive for normalized waiting time. By Lemma 23, if we double the number of machines and increase the speed $2s$ times, then the competitive ratio is reduced to $1/s$, where $s \geq 1$. □

Lemma 23 has another implication. It is known that HDF is 4-speed 2-competitive for weighted flow time [6]. Again, if we double the number of machines and increase the speed $2s$ times for any $s \geq 1$, then the competitive ratio is reduced to $1/s$.

LEMMA 26. *Let $s \geq 1$ be any real number.* HDF *is $(2, 8s)$-machine-speed $(1/s)$-competitive for weighted flow time.*

The above result also implies an algorithm that is $16s$-speed $(1/s)$-competitive for weighted flow time (by simulating HDF$(2, 8s)$ using time-sharing).

6. Conclusion.

This paper serves as the first step in understanding how extra-machine analysis is related to extra-speed analysis, and how extra machines can provide 1-competitive scheduling for minimizing flow time and stretch. There are several interesting problems to be addressed. We do not have a similar result for weighted flow time. Unlike the algorithm IMD [13], our new algorithms incorporate SRPT or HDF, and they are migratory in nature and do not allow immediate dispatch. It is interesting to investigate nonmigratory algorithms with similar performances. Another important direction is to minimize the L_p norm of flow time and stretch [5, 13]. Note that Chekuri et al. [13] have extended $(1+\epsilon)$-speed (or $(1+\epsilon)$-machine) $O(1+1/\epsilon)$-competitive results for flow time and stretch to the L_p norm.

REFERENCES

[1] N. Avrahami and Y. Azar, *Minimizing total flow time and total completion time with immediate dispatching*, in Proceedings of SPAA, ACM, New York, 2003, pp. 11–18.

[2] B. Awerbuch, Y. Azar, S. Leonardi, and O. Regev, *Minimizing the flow time without migration*, in Proceedings of STOC, ACM, New York, 1999, pp. 198–205.

[3] N. Bansal, *On minimizing the total flow time on multiple machines*, in Proceedings of SODA, ACM, New York, SIAM, Philadelphia, 2004, pp. 572–574.

[4] N. Bansal and K. Dhamdhere, *Minimizing weighted flow time*, in Proceedings of SODA, ACM, New York, SIAM, Philadelphia, 2003, pp. 508–516.

[5] N. Bansal and K. Pruhs, *Server scheduling in the L_p norm: A rising tide lifts all boat*, in Proceedings of STOC, ACM, New York, 2003, pp. 242–250.

[6] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs, *Online weighted flow time and deadline scheduling*, in Proceedings of RANDOM-APPROX, Springer-Verlag, Berlin, 2001, pp. 36–47.

[7] L. Becchetti, S. Leonardi, and S. Muthukrishnan, *Scheduling to minimize average stretch without migration*, in Proceedings of SODA, ACM, New York, SIAM, Philadelphia, 2000, pp. 548–557.

[8] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan, *Flow and stretch metrics for scheduling continuous job streams*, in Proceedings of SODA, ACM, New York, SIAM, Philadelphia, 1998, pp. 270–279.

[9] M. A. Bender, S. Muthukrishnan, and R. Rajaraman, *Improved algorithms for stretch scheduling*, in Proceedings of SODA, ACM, New York, SIAM, Philadelphia, 2002, pp. 762–771.

[10] P. Berman and C. Coulston, *Speed is more powerful than clairvoyance*, in Proceedings of the 6th Annual SWAT, Stockholm, Sweden, 1998, pp. 255–263.

[11] H. L. Chan, T. W. Lam, and K. K. To, *Non-migratory online deadline scheduling on multiprocessors*, in Proceedings of SODA, ACM, New York, SIAM, Philadelphia, 2004, pp. 970–979.

[12] W. T. Chan, T. W. Lam, K. S. Liu, and P. Wong, *New resource augmentation analysis of the total stretch of SRPT and SJF in multiprocessor scheduling*, in Proceedings of MFCS, Springer-Verlag, Berlin, 2005, pp. 236–247.

[13] C. Chekuri, A. Goel, S. Khanna, and A. Kumar, *Multi-processor scheduling to minimize flow time with ϵ resource augmentation*, in Proceedings of STOC, ACM, New York, 2004, pp. 363–372.

[14] C. Chekuri, S. Khanna, and A. Zhu, *Algorithms for minimizing weighted flow time*, in Proceedings of STOC, ACM, New York, 2001, pp. 84–93.
[15] J. Du, J. Y. T. Leung, and G. H. Young, *Minimizing mean flow time with release time constraint*, Theoret. Comput. Sci., 75 (1990), pp. 347–355.
[16] J. Edmonds, *Scheduling in the dark*, in Proceedings of STOC, ACM, New York, 1999, pp. 179–188.
[17] B. Kalyanasundaram and K. Pruhs, *Speed is as powerful as clairvoyance*, J. ACM, 47 (2000), pp. 617–643.
[18] C. Y. Koo, T. W. Lam, T. W. Ngan, and K. K. To, *Extra processors versus future information in optimal deadline scheduling*, in Proceedings of SPAA, ACM, New York, 2002, pp. 133–142.
[19] K. R. Baker, *Introduction to Sequencing and Scheduling*, Wiley, New York, 1974.
[20] S. Leonardi and D. Raz, *Approximating total flow time on parallel machines*, in Proceedings of STOC, ACM, New York, 1997, pp. 110–119.
[21] J. McCullough and E. Torng, *SRPT optimally utilizes faster machines to minimize flow time*, in Proceedings of SODA, ACM, New York, SIAM, Philadelphia, 2004, pp. 343–351.
[22] S. Muthukrishnan, R. Rajaraman, A. Shaheen, and J. Gehrke, *Online scheduling to minimize average stretch*, in Proceedings of FOCS, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 433–442.
[23] C. A. Phillips, C. Stein, E. Torng, and J. Wein, *Optimal time-critical scheduling via resource augmentation*, in Proceedings of STOC, ACM, New York, 1997, pp. 140–149.
[24] K. Pruhs, J. Sgall, and E. Torng, *Online scheduling*, in Handbook of Scheduling: Algorithms, Models and Performance Analysis, J. Leung, ed., CRC Press, Boca Raton, FL, 2004, pp. 15-1–15-41.