

Extra Unit-Speed Machines are Almost as Powerful as Speedy Machines for Competitive Flow Time Scheduling

Ho-Leung Chan*

Tak-Wah Lam*

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 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 [23] and stretch [12], and HDF (highest density first) is $O(1)$ -competitive for weighted flow time [6]. Using extra unit-speed machines instead of faster machines is more challenging. This paper gives a non-trivial relationship between the extra-speed and extra-machine analysis. It shows that competitive results via faster machines can be transformed to similar results via extra machines, and 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, respectively.

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 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 length of the time interval 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 of that of the optimal offline schedule.

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.

Pioneered by Kalyanasundaram and Pruhs [17], resource augmentation has become a popular approach to studying better performance guarantee for 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 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 achieve. 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 et al. [14] proposed a different algorithm that is 9.81-competitive. They also gave a lower bound on the competitive ratio

*Department of Computer Science, University of Hong Kong.
Email: {hlchan, twlam, ksliau}@cs.hku.hk

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 are known. Chekuri et al. [13] proved that the non-migratory 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, Chekuri et al. [13] have recently 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.

To ease our discussion, we adopt the following notations. Let α and τ be any 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 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 non-trivial 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, $L_{A'\langle\tau\rangle}(I) \leq cL_{A(\alpha)}(I) + o(L_{OPT}(I))$. 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

¹We consider a simple example where $\alpha = 2$. Let I be a job sequence such that the i -th 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 put some job to wait after the $(\tau m + 1)$ -th job is released, thus incurring non-zero waiting time.

	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]	$(36h - 2)m,$ $h \geq 1$	$1/h$	†

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.

on two interesting concepts called rate control and waiting time allowance, both make scheduling easy. To make these two concepts viable without blowing up the flow time or waiting time, we skillfully exploit a simulation of a α -speed competitive algorithm and a prime period busy algorithm.

This paper also makes contribution 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.

Remarks: 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, our new algorithms incorporate SRPT or HDF, and they are migratory in nature and do not allow immediate dispatch. It is interesting to investigate non-migratory algorithms with a similar performance. 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.

2 Transformation that Preserves Flow Time

Throughout this paper, we use I to denote a sequence of jobs. We denote the release time and processing time of a job J as $r(J)$ and $p(J)$, respectively. 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 $\text{Scatter}(A(\alpha), \tau)$, that uses $\lceil \tau m \rceil$ unit-speed machines for any $\tau > \alpha$ and has performance comparable to $A(\alpha)$ in terms of flow time as follows.

LEMMA 2.1. *Consider any job sequence I . For each*

job $J \in I$, the flow time of J in the schedule of $\text{Scatter}(A(\alpha), \tau)$ is at most $\alpha(1 + \frac{\alpha-1}{\tau-\alpha})$ times in the schedule of $A(\alpha)$.

$\text{Scatter}(A(\alpha), \tau)$ divides the $\lceil \tau m \rceil$ machines into two bands. Band 1 uses m 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 flow time of J in Band 1 and Band 2, denoted as $F_1(J)$ and $F_2(J)$, as follows:

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

Then it follows that the flow time of J in the schedule of $\text{Scatter}(A(\alpha), \tau)$ is $F_1(J) + F_2(J) \leq (1 + \frac{(\alpha-1)\tau}{\tau-\alpha})F_{A(\alpha)}(J) = \alpha(1 + \frac{\alpha-1}{\tau-\alpha})F_{A(\alpha)}(J)$, which is as stated in Lemma 2.1.

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 a bulk to Band 1, yet they will each be partially processed before 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 2.2 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 **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 2.2. (*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 $\sum_{J \in H} \text{rem}(J) \leq (\alpha - 1)(t' - t)m$.

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, $\sum_{J \in H} \frac{1}{\alpha}p(J) \leq m(t' - t)$, and $\sum_{J \in H} \text{rem}(J) = \sum_{J \in H} (1 - \frac{1}{\alpha})p(J) \leq (\alpha - 1)(t' - t)m$. \square

LEMMA 2.3. (*LRT*) For any job J , $F_2(J) \leq (\alpha - 1) \times \frac{\tau}{\tau - \alpha} F_1(J)$.

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 $F_1(J) = t_1 - t_0 \geq p(J)/\alpha$, 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., $r(J') \geq r(J)$), and J' is transferred to Band 2 no later than t_2 . Applying Lemma 2.2 to the interval $[t_0, t_2]$, we have $\sum_{J' \in S} \text{rem}(J') \leq (\alpha - 1)(t_2 - t_0)m$.

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 $\frac{1}{\lceil (\tau - 1)m \rceil} \sum_{J' \in S} \text{rem}(J') \leq \frac{1}{\lceil (\tau - 1)m \rceil} (\alpha - 1)(t_2 - t_0)m$. Therefore, $F_2(J) =$

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

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

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 2.1 to obtain the following extra-machine competitive results.

COROLLARY 2.1. 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

Let $A(\alpha)$ be any 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 **Scatter_&Squash**($A(\alpha), \tau$) that uses $\lceil \tau m \rceil$ unit-speed machines and incurs a total waiting time comparable to that of $A(\alpha)$ as follows.

LEMMA 3.1. Let $\tau = 7\alpha + 5k - 2$ for any integer $k \geq 1$. 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 3.1 in Section 3.1. Let us consider its implication first. Suppose that A is α -speed $(1/x)$ -competitive for waiting time for some $x \geq 1$. Let $k = \lceil x \rceil$ and $\tau = 7\alpha + 5\lceil x \rceil - 2$. By Lemma 3.1, **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 3.1. (i) **Scatter_&Squash** can give an algorithm that is 34 -machine 1-competitive for flow time. (ii) For any integer $h \geq 1$, **Scatter_&Squash** (based on SRPT($3h$)) can give an algorithm that is $(36h - 2)$ -machine $(1/h)$ -competitive for waiting time.

3.1 The algorithm As to be 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 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 $L_i(I) = \sum_{J \in I} L_i(J)$. 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 3.1 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 has 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 2.2, Band 1a provides the rate control property.

Define the *prime 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 prime period. That is, a job transferred out of Band 1a within its prime period is retained in Band 1b until the end of its prime period.

LEMMA 3.2. *If a job J is transferred from Band 1 to Band 2 at the end of or after J 's prime 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, $L_1(J) \geq p(J) - w(J) = rem(J)$. \square

Band 1 as a whole still satisfies the rate control property because jobs released and transferred to Band 2 within an interval T is 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 any prime period busy algorithm on $2k + 1$ machines, denoted $PPBUSY\langle 2k + 1 \rangle$, incurs a total 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 and it runs the algorithm $PPBUSY\langle 2k + 1 \rangle$. At the end of the prime period of a job J , J is transferred to Band 2 if it is not completed (with $AWT(J) = L_{1a}(J) + L_{1b}(J)$).

Band 2: It uses $\lceil(7\alpha + 3k - 4)m\rceil$ machines. At any time, it runs the MIN-AWT algorithm to complete the jobs.

at most $\frac{1}{2k}$ times of OPT . Formally speaking, at any time, $PPBUSY$ only considers jobs that are still in their prime periods, and it selects arbitrarily one job for each machine. Notice that $PPBUSY$ may not complete a job J and does not incur waiting time beyond the prime period of J , yet OPT does both. In Section 3.2, we will give a careful charging scheme to relate the waiting time of $PPBUSY$ and OPT . In summary, Band 1 has the following upper bound on waiting time.

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

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 $rem(J)$ (i.e., the remaining work of J at $tsf(J)$) 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 .

We use an inductive proof and 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 3.3. $\sum_{J' \in S} rem(J') \leq \delta m AWT(J)$, 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 $t_1 - t_0 \geq xAWT(J)$. 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 3.3. \square

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 PPBUSY Scatter & Squash uses the algorithm PPBUSY in Band 1b. To upper bound the waiting time incurred in Band 1b, we first study in this section the waiting time incurred by PPBUSY when it is used alone to process a sequence of jobs. The latter result may have its own interest in other scheduling problem.

PPBUSY(h) uses hm machines, for any integer $h \geq 2$. It schedules a job only within its prime 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 PPBUSY(h) on I , and similarly $OPT(I)$ for OPT . Note that a job remains in $P(I)$ only during its prime 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 $\lambda_1 = [t_1, t'_1], \lambda_2 = [t_2, t'_2], \lambda_3 = [t_3, t'_3], \dots$, where $t_1 < t'_1 < t_2 < t'_2 < t_3 < t'_3 < \dots$. Let $|\lambda_i| = t'_i - t_i$. Note the $P(I)$ accumulates waiting time only during the waiting periods.

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

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 3.1. For any $u \leq i \leq v$, $t_S - t_i \geq h(|\lambda_i| + |\lambda_{i+1}| + \dots + |\lambda_v|)$.

We partition the waiting periods in $P(I)$ into maximal h -close collections $S_1 = \{\lambda_1, \lambda_2, \dots, \lambda_{k_1}\}$, $S_2 =$

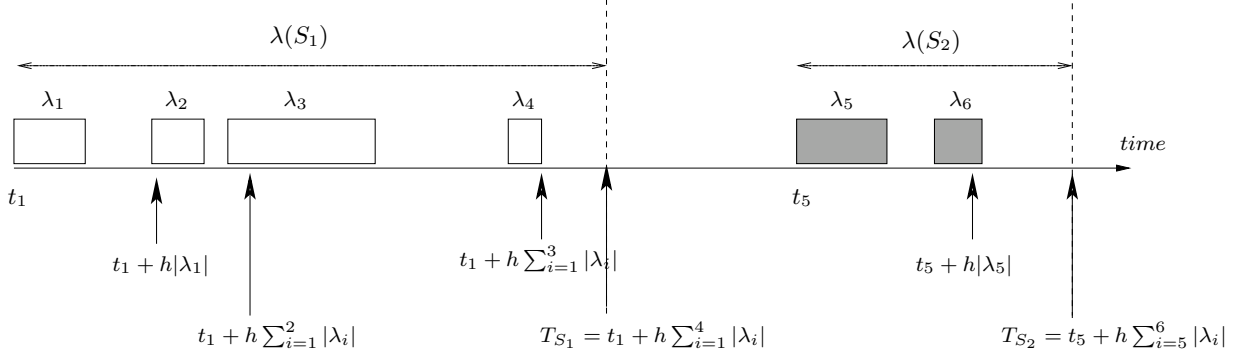


Figure 1: Two h -close collections of waiting periods ($h = 2$ in this example).

$\{\lambda_{k_1+1}, \lambda_{k_1+2}, \dots, \lambda_{k_2}\}, \dots$. 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 2. Consider any interval $\lambda = [t, t']$ and any job J . If λ is enclosed in the prime period of J , the work of J can be partitioned into three chunks of size $t - r(J)$, $t' - t$, and $r(J) + p(J) - t'$, respectively. The middle chunk is referred to as the λ -work of J . In general, for any arbitrary λ , let $\lambda' = \lambda \cap [r(J), r(J) + p(J)]$ and define the λ -work of J to be the λ' -work of J . The amount of work in the λ -work of J is denoted 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 a job J is $\max\{t, r(J)\}$.

Let $S = \{\lambda_u, \lambda_{u+1}, \dots, \lambda_v\}$ 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 $\lambda_\ell = [t_i, t'_i]$ be the first waiting period overlapping with the prime period in S of J (i.e., $W(J, \lambda_\ell) > 0$). Note that OPT cannot process the λ_ℓ -work of J earlier than t_i or $r(J)$. We mark the first $W(J, \lambda_\ell)$ units of work starting from the time $\max\{t_i, r(J)\}$ in the schedule of J in $OPT(I)$. For each subsequent $j > \ell$, if the λ_j -work of J is non-null, we identify, in the schedule of J in $OPT(I)$, the first time $t \geq t_j$ when no work has been marked, and we mark another $W(J, \lambda_j)$ units of work starting from t . We have

the following observation 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 3.4. *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, $y \leq |\lambda_i| + |\lambda_{i+1}| + \dots + |\lambda_v|$. 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 3.1, we conclude that $L_{OPT}(J)|_{\lambda(S)} \geq h(|\lambda_i| + |\lambda_{i+1}| + \dots + |\lambda_v|) - |\lambda_i| \geq (h-1)y$. \square

LEMMA 3.5.
$$\sum_{J \in I} L_P(J)|_{\lambda(S)} \leq \frac{1}{h-1} \sum_{J \in I} L_{OPT}(J)|_{\lambda(S)}.$$

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

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 3.4, $L_{OPT}(J)|_{\lambda(S)}$ 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 $(h - 1) \sum_{J \in I} L_P(J)|_{\lambda(S)}$. \square

COROLLARY 3.2. *For any job sequence I , let $L_P(I)$ be the total waiting time incurred by $PPBUSY\langle h \rangle$ and $L_{OPT}(I)$ be that for OPT . Then, $L_P(I) \leq \frac{1}{h-1} L_{OPT}(I)$.*

PPBUSY 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 $PPBUSY\langle 2k+1 \rangle$. 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 prime period.

Let us compare the schedule of Band 1b with the schedule when I is scheduled by a stand-alone copy of $PPBUSY\langle 2k+1 \rangle$, which considers each job throughout J 's prime period. At any time t , if a job J remains in Band 1b, then t is still within J 's prime period, and J also remains in the stand-alone $PPBUSY\langle 2k+1 \rangle$. Thus, jobs remaining in Band 1b is a subset of jobs remaining in the stand-alone $PPBUSY\langle 2k+1 \rangle$. As both Band 1b and the stand-alone $PPBUSY\langle 2k+1 \rangle$ 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 the stand-alone $PPBUSY\langle 2k+1 \rangle$, denoted $\#_P(I, t)$.

Let $L_{1b}(J)$ and $L_P(J)$ be the waiting time of J in the schedule of Band 1b and the stand-alone $PPBUSY\langle 2k+1 \rangle$, respectively. We have $\sum_{J \in I} L_{1b}(J) = \int \#_{1b}(I, t) dt \leq \int \#_P(I, t) dt = \sum_{J \in I} L_P(J) \leq \frac{1}{2k} \sum_{J \in I} L_{OPT}(J)$.

4 Extension to Weighted Waiting Time and Stretch

An $O(1)$ -machine algorithm is 1-competitive algorithm for stretch if and only if it is 1-competitive for normalized waiting time (recall that the normalized waiting time of a job refers to the waiting time divided by the processing time). Thus, it is desirable that **Scatter_&_Squash** can preserve the normalized waiting time. In fact, we can show that **Scatter_&_Squash** can be extended to preserve the weighted waiting time, i.e.,

each job is given an arbitrary weight. Notice that regarding weighted waiting time, Band 1a incurs the same amount as the given α -speed algorithm $A(\alpha)$ does, and Band 2 incurs no more than Band 1 does. Only Band 1b requires modification to cater for the weighted setting.

In **Scatter_&_Squash**, Band 1b uses an arbitrary prime period busy algorithm on $(2k+1)m$ machines. We enhance Band 1b by requiring it to select jobs with highest weights. We call this new busy algorithm PPHWF (prime period, highest weight first). Intuitively, jobs with big weights will wait less. Let OPT be the optimal algorithm (using m unit-speed machines) for minimizing weighted flow time. The key observation is that for any job weight w_i , we can bound together the waiting time of all jobs with weight at least w_i in PPHWF $\langle 2k+1 \rangle$ to be at most $1/(2k)$ times that of OPT . Then, by induction from the largest to the smallest job weight, we show that PPHWF $\langle 2k+1 \rangle$ incurs a total weighted waiting time at most $1/(2k)$ times of OPT .

Furthermore, we can extend the transformation so that the input algorithm, denoted $A(\ell, \alpha)$, uses ℓm α -speed machines, where $\ell \geq 1$ is an integer. We call the transformation with the above extensions **weighted_SS** $(A(\ell, \alpha), \tau)$, which uses $\lceil \tau m \rceil$ machines to preserve the weighted waiting time as follows.

LEMMA 4.1. *Let $\tau = 7\alpha\ell + 5k - \frac{9}{2}\ell + \frac{5}{2}$. The weighted waiting time incurred by **weighted_SS** $(A(\ell, \alpha), \tau)$ is at most 2 times of $A(\ell, \alpha)$ plus $1/k$ times of OPT .*

The proof of the above lemma will be given in the full paper. For the special case of normalized waiting time (i.e., the weight of each job J is $1/p(J)$), we can show that SJF is $(4, 8s)$ -machine-speed $1/s$ -competitive for normalized waiting time for any $s \geq 1$ (see Section 5). Then by Lemma 4.1, we obtain an algorithm that is 533-machine 1-competitive for normalized waiting time, as well as the following result.

COROLLARY 4.1. *Based on SJF, **weighted_SS** gives a 533-machine 1-competitive algorithm for stretch.*

5 Improved Analysis of HDF and SJF on Faster Machines

Recall that the competitive ratio of SRPT for flow time and waiting time can be made arbitrarily small with increased speed. [21]. This section presents a similar result for HDF on weighted flow time and normalized waiting time. We use the notation $A(\ell, \alpha)$ to denote an algorithm running on ℓm α -speed machines, and we say that $A(\ell, \alpha)$ is (ℓ, α) -machine-speed c -competitive if it has performance at most c times of any offline algorithm using m unit-speed machines. It is known that HDF is $(1, 4)$ -machine-speed 2-competitive for weighted flow time [6]. Our key results are as follows.

LEMMA 5.1. *Let $s \geq 1$ be any real number. HDF is $(2, 8s)$ -machine-speed $(1/s)$ -competitive for weighted flow time, and SJF is $(4, 8s)$ -machine-speed $(1/s)$ -competitive for normalized waiting time.*

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

Let $w(J)$ denote the weight of a job J . The density of J is $w(J)/p(J)$. HDF(ℓ, α) always schedules up to ℓm jobs with the highest density and we assume that ties are broken by job ID. Lemma 5.1 stems from an observation that the performance of HDF can be scaled with machine speed. Precisely, we compare HDF(ℓ, α) and HDF($2\ell, c\alpha$) where $c \geq 1$ is a real. We show that the waiting time of each job decreases by c times (see the lemma below).

LEMMA 5.2. *For each job J , $L_{(2,c)}(J) \leq \frac{1}{c}L_{(1,1)}(J)$, where $L_{(2,c)}(J)$ and $L_{(1,1)}(J)$ are the waiting time of J incurred by HDF($2\ell, c\alpha$) and HDF(ℓ, α), respectively.*

Proof. Denote $S_1(I)$ and $S_2(I)$ as the schedule of a job sequence I using HDF(ℓ, α) and HDF($2\ell, c\alpha$), respectively. Consider a job J . Assume J is completed at time $z(J)$ in $S_1(I)$. During $[r(J), z(J)]$, the waiting time of J in $S_1(I)$ can be calculated as follows. Let I' be the job sequence formed by removing J and all jobs with lower priority than J . At any time, J waits in $S_1(I)$ if and only if $S_1(I')$ is busy (i.e., all machines are running some jobs). So the waiting time of J is the total length of the busy periods in $S_1(I')$ during $[r(J), z(J)]$. Let $\lambda_1, \lambda_2, \dots$ be these busy periods.

Let $\lambda = [t_1, t_2]$ be one of the above busy periods. Let $W = \ell m \alpha |\lambda|$ be the work done by $S_1(I')$ during $|\lambda|$. Let R be the 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 λ .

Next, we consider the schedule $S_2(I')$. Within the time interval λ , denote the busy periods in $S_2(I')$ as $\rho_1, \rho_2, \dots, \rho_n$. Let g be the total length of these ρ_j . Note that $g \leq |\lambda|$. During these periods ρ_j 's, $S_2(I')$ can process the work W plus at most $c\alpha g$ units of work for each job in R . Thus, the total amount of work done by $S_2(I')$ during these ρ_i 's is exactly $2\ell m \alpha c g$, which is upper bounded by $W + c\alpha g |R| < \ell m \alpha |\lambda| + c\alpha g \ell m$. Rearranging the terms, we have $g \leq |\lambda|/c$.

In $S_2(I)$, the waiting time of J during λ is at most the total length of busy periods in $S_2(I')$ during λ , which is at most $\frac{1}{c}|\lambda|$. Note that during $[r(J), z(J)]$, $S_2(I')$ is not busy for any time outside the periods λ_i 's. So by considering each λ_i individually, we conclude that the waiting time of J in $S_2(I)$ at most $\frac{1}{c}$ times that of $S_1(I)$. \square

Lemma 5.2 also implies that when comparing HDF($2\ell, c\alpha$) against HDF(ℓ, α), the weighted waiting

time, flow time, and weighted flow time of each job also decrease by c times. Since HDF is $(1, 4)$ -machine-speed 2-competitive for weighted flow time, we conclude that HDF is $(2, 8s)$ -machine-speed $(1/s)$ -competitive for weighted flow time for any $s \geq 1$. In the appendix, we will show that SJF(2, 4) is 2-competitive for normalized waiting time. By Lemma 5.2, HDF(4, 8s) is $1/s$ -competitive for normalized waiting time for any $s \geq 1$.

References

- [1] N. Avrahami and Y. Azar. Minimizing total flow time and total completion time with immediate dispatching. In *SPAA*, pages 11–18, 2003.
- [2] B. Awerbuch, Y. Azar, S. Leonardi and O. Regev. Minimizing the flow time without migration. In *STOC*, pages 198–205, 1999.
- [3] N. Bansal. On minimizing the total flow time on multiple machines. In *SODA*, pages 572–574, 2004.
- [4] N. Bansal and K. Dhamdhare. Minimizing weighted flow time. In *SODA*, pages 508–516, 2003.
- [5] N. Bansal and K. Pruhs. Server scheduling in the L_p norm: a rising tide lifts all boat. In *STOC*, pages 242–250, 2003.
- [6] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs. Online weighted flow time and deadline scheduling. In *RANDOM-APPROX*, pages 36–47, 2001.
- [7] L. Becchetti, S. Leonardi, and S. Muthukrishnan. Scheduling to minimize average stretch without migration. In *SODA*, pages 548–557, 2000.
- [8] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *SODA*, pages 270–279, 1998.
- [9] M. A. Bender, S. Muthukrishnan, and R. Rajaraman. Improved algorithms for stretch scheduling. In *SODA*, pages 762–771, 2002.
- [10] P. Berman and C. Coulston. Speed is more powerful than clairvoyance. In *Proc. 6th SWAT*, pages 255–263, 1998.
- [11] H. L. Chan, T. W. Lam, and K. K. To. Non-migratory online deadline scheduling on multiprocessors. In *SODA*, pages 970–979, 2004.
- [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 *MFCS*, 2005.
- [13] C. Chekuri, A. Goel, S. Khanna, and A. Kumar. Multi-processor scheduling to minimize flow time with ϵ resource augmentation. In *STOC*, pages 363–372, 2004.
- [14] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for minimizing weighted flow time. In *STOC*, pages 84–93, 2001.
- [15] J. Du, J. Y. T. Leung, and G. H. Young. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75(3):347–355, 1990.
- [16] J. Edmonds. Scheduling in the Dark. In *STOC*, pages 179–188, 1999.
- [17] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

- [18] C. Y. Koo, T. W. Lam, T. W. Ngan and K. K. To. Extra processors versus future information in optimal deadline scheduling. In *SPAA*, pages 133–142, 2002.
- [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 *STOC*, pages 110–119, 1997.
- [21] J. McCullough and E. Torng. SRPT optimally utilizes faster machines to minimize flow time. In *SODA*, pages 343–351, 2004.
- [22] S. Muthukrishnan, R. Rajaraman, A. Shaheen, and J. Gehrke. Online scheduling to minimize average stretch. In *FOCS*, pages 433–442, 1999.
- [23] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *STOC*, pages 140–149, 1997.
- [24] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In J. Leung, editor, *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, pages 15–1–15–41. CRC Press, 2004.

Appendix. Normalized Waiting Time of SJF

For the special case that the weight of each job J is $1/p(J)$, we will show that SJF is $(2, 4)$ -machine-speed 2-competitive for normalized waiting time. Since SJF and HDF are equivalent in this special case, we can make use of Lemma 5.2 to show that SJF is $(4, 8s)$ -machine-speed $1/s$ -competitive for normalized waiting time for any $s \geq 1$.

Basically, we will first show that SJF is $(2, 4)$ -machine-speed 2-competitive for (unweighted) waiting time. SJF accumulates less waiting time than other algorithms on jobs with smaller size, so by considering the jobs with increasing job size, we can show that SJF is $(2, 4)$ -machine-speed 2-competitive even for normalized waiting time. Details are as follows. We let ALG be any algorithm using m unit-speed machines.

LEMMA 5.3. *SJF is $(2, 4)$ -machine-speed 2-competitive for (unweighted) waiting time.*

Proof. Consider any job sequence I . Since we are interested in the (unweighted) waiting time only, we can assume that the weight of each job is 1 and the schedule of SJF is unaffected.

We notice the schedule of SJF on I is equivalent to the schedule of HDF on I . It is shown in [6] that HDF(1, 4) is locally 2-competitive in weighted flow time, i.e., at any time t , the total weight of jobs remaining in HDF(1, 4) is at most two times of ALG . Thus, at any time, the number of jobs remaining in SJF(1, 4) is at most two times of ALG .

At any time t , let $U_t(\text{SJF}(2, 4))$ be the number of jobs remaining in SJF(2, 4), and define $U_t(\text{SJF}(1, 4))$ and $U_t(ALG)$ similarly. Then, $U_t(\text{SJF}(2, 4)) \leq U_t(\text{SJF}(1, 4)) \leq 2 \times U_t(ALG)$. Note that SJF(2, 4) has $2m$ machines. If there are more than $2m$ jobs remaining in SJF(2, 4), the number of jobs waiting in SJF(2, 4) =

$U_t(\text{SJF}(2, 4)) - 2m \leq 2(U_t(ALG) - m)$. Thus, at any time, the number of jobs waiting in SJF(2, 4) at most two times that of ALG , and the lemma follows. \square

LEMMA 5.4. *SJF is $(2, 4)$ -machine-speed 2-competitive for normalized waiting time.*

Proof. We consider any job sequence I , and let $s_1 < s_2 < \dots < s_r$ be the distinct values of job size in I . Let $L_{\text{SJF}}(s_i, I)$ be the total waiting time incurred by SJF(2, 4) on jobs in I with size exactly s_i , and define $L_A(s_i, I)$ similarly for ALG . For any $q = 1, \dots, r$, let I' be the job sequence including only jobs in I with size at most s_q . Since SJF does not change the schedule of a job due to other jobs with larger size, the total waiting time incurred by SJF(2, 4) on $I' = \sum_{i=1}^q L_{\text{SJF}}(s_i, I') = \sum_{i=1}^q L_{\text{SJF}}(s_i, I)$. By Lemma 5.3, it is at most two times the total waiting time incurred by any schedule of I' on m unit-speed machines. Thus, for $q = 1, 2, \dots, r$, $\sum_{i=1}^q L_{\text{SJF}}(s_i, I) \leq 2 \times \sum_{i=1}^q L_A(s_i, I)$.

By putting $a_i = L_{\text{SJF}}(s_i, I)$, $b_i = L_A(s_i, I)$, $c_i = 1/s_i$ and $M = 2$ into the next lemma, we conclude that the normalized waiting time of SJF(2, 4) = $\sum_{i=1}^r \frac{1}{s_i} L_{\text{SJF}}(s_i, I) \leq 2 \sum_{i=1}^r \frac{1}{s_i} L_A(s_i, I) =$ the normalized waiting time of ALG . \square

LEMMA 5.5. *Let $r \geq 1$ be an integer and $M > 0$ be a real number. Let a_1, \dots, a_r and b_1, \dots, b_r be two sequences of non-negative real numbers such that for $q = 1, \dots, r$, $\sum_{i=1}^q a_i \leq M \times \sum_{i=1}^q b_i$. Let $c_1 > c_2 > \dots > c_r$ be a sequence of positive real numbers. Then, $\sum_{i=1}^r c_i a_i \leq M \times \sum_{i=1}^r c_i b_i$.*

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 $z \geq 1$. When $r = z + 1$, we have a_1, \dots, a_{z+1} and b_1, \dots, b_{z+1} such that for $q = 1, \dots, z + 1$, $\sum_{i=1}^q a_i \leq M \times \sum_{i=1}^q b_i$. We consider two cases.

(Case 1.) If $a_{z+1} \leq M \times b_{z+1}$, then $\sum_{i=1}^{z+1} c_i a_i = \sum_{i=1}^z c_i a_i + c_{z+1} a_{z+1} \leq M \times \sum_{i=1}^z c_i b_i + M \times c_{z+1} b_{z+1} = M \times \sum_{i=1}^{z+1} c_i b_i$.

(Case 2.) Otherwise, $a_{z+1} > M \times b_{z+1}$. Let $a_{z+1} = M \times b_{z+1} + \delta$ for some real number $\delta > 0$.

$$\begin{aligned} \sum_{i=1}^{z+1} c_i a_i &= \sum_{i=1}^{z-1} c_i a_i + c_z a_z + c_{z+1} \times (M \times b_{z+1} + \delta) \\ &\leq \sum_{i=1}^{z-1} c_i a_i + c_z (a_z + \delta) + M \times c_{z+1} b_{z+1}. \end{aligned}$$

Consider the sequence of real numbers d_1, \dots, d_z where for $i = 1, \dots, z - 1$, $d_i = a_i$ and $d_z = a_z + \delta$. For any $q = 1, \dots, z - 1$, $\sum_{i=1}^q d_i = \sum_{i=1}^q a_i \leq M \times \sum_{i=1}^q b_i$. For $q = z$, $\sum_{i=1}^z d_i = \sum_{i=1}^z a_i + \delta = \sum_{i=1}^{z+1} a_i - M \times b_{z+1} \leq M \times \sum_{i=1}^z b_i$. Thus, by the induction hypothesis, $\sum_{i=1}^z c_i d_i \leq M \times \sum_{i=1}^z c_i b_i$. Thus, we have $\sum_{i=1}^{z+1} c_i a_i \leq M \times \sum_{i=1}^z c_i b_i + M \times c_{z+1} b_{z+1} = M \times \sum_{i=1}^{z+1} c_i b_i$, and it completes the induction. \square