# Algorithms for Minimizing Weighted Flow Time

[Extended Abstract]

Chandra Chekuri[*] Sanjeev Khanna[†] An Zhu[‡]

## ABSTRACT

We study the problem of minimizing *weighted* flow time on a single machine in the preemptive setting. We present an $O(\log^2 P)$-competitive *semi-online* algorithm where $P$ is the ratio of the maximum and minimum processing times of jobs in the system. In the offline setting we show that a $(2+\epsilon)$-approximation is achievable in quasi-polynomial time. These are the first non-trivial results for the weighted versions of minimizing flow time. For multiple machines we show that no competitive randomized online algorithm exists for weighted flow time. We also present an improved online algorithm for minimizing total stretch (a special case of weighted flow time) on multiple machines.

## 1. INTRODUCTION

Scheduling independent jobs that arrive over time is a fundamental problem that arises in a variety of applications. The abstract goal of a scheduler is to optimize one or more metrics of the quality of service delivered to the jobs. Some well-studied metrics of interest include throughput, maximum completion time (makespan), sum of completion times, and total flow-time. In many settings, jobs have varying degrees of importance and this is usually represented by assigning weights to the jobs. Typical metrics of interest in such cases are weighted completion time and weighted flow time. In the online setting, where jobs are independent, the total (weighted) flow time is one of the simplest and natural metrics that measures the average service received by the jobs. The flow time of a job (also known as the response time) is the total time it spends in the system, thus it is the waiting time plus the processing time of the job. In this paper we present the first non-trivial algorithms for the preemptive version of minimizing weighted flow time.

Despite substantial interest, until recently, no provably good non-trivial algorithms were known for either the unweighted or the weighted versions of minimizing flow time. In fact, the non-preemptive problem is intractable in a strong sense in both the online and offline settings. In the online setting, even on a single machine, no algorithm can achieve a competitive ratio better than $\Omega(n)$, where $n$ is the number of jobs. When weights are allowed, no online algorithm can achieve a non-trivial competitive ratio. In the offline setting, Kellerer et al. [8] showed that the problem of minimizing unweighted flow time non-preemtively on a single machine is $n^{\frac{1}{2}-\epsilon}$-hard. Thus preemption seems to be essential to obtaining tractable variants of the flow time measure and we focus on this case for the rest of the paper.

We first discuss the unweighted case. A folklore result states that the online algorithm that schedules the job with the *shortest remaining processing time* (SRPT) gives the optimal total flow time on a single machine. Leonardi and Raz [10] analyzed SRPT for the multiple processor case and showed that it is $O(\min\{\log P, \log \frac{n}{m}\})$ competitive; here $P$ is the ratio of maximum and minimum job processing times, and $n$ and $m$ denote the number of jobs and machines respectively. They also showed that no online algorithm (even with randomization) can achieve a better competitive ratio.

For the weighted case, the only known results are for the special case where the weight of each job is inverse of its processing time. This metric, referred to as *stretch*, was first introduced by Bender et al. [3]. They studied the problems of minimizing the *maximum* stretch and *maximum* flow time. The total stretch metric, which is more appropriate for measuring the average performance of the schedule, was first studied by Muthukrishnan et al. [11]. They analyzed the SRPT algorithm and showed that it achieves a competitive ratio of 2 for the single machine, and a competitive ratio of 13 for the multiple machine case. They also established an $\Omega(1)$ lower bound on the competitiveness of any online algorithm. Thus SRPT is competitive for unweighted flow time as well as total stretch. This is perhaps not too surprising, informally speaking, since both measures favor short jobs over long jobs, and the SRPT algorithm implicitly gives priority to the shorter jobs over longer jobs. However, the assumption that shorter jobs are preferred over longer jobs is not flexible enough to handle priorities on jobs that are independent of the size. The SRPT algorithm can be easily

[*]Bell Labs, 600 Mountain Ave, Murray Hill, NJ 07974. E-mail: chekuri@research.bell-labs.com.
[†]Dept. of CIS, University of Pennsylvania, Philadelphia, PA 19104. E-mail: sanjeev@cis.upenn.edu. Supported in part by an Alfred P. Sloan Research Fellowship and by an NSF Career Award CCR-0093117.
[‡]Dept. of Computer Science, Stanford University, Stanford, CA 94305. E-mail: anzhu@cs.stanford.edu. Supported by a GRPW fellowship from Lucent Technologies.


*STOC'01,* July 6-8, 2001, Hersonissos, Crete, Greece.

shown to have an $\Omega(P)$ competitive ratio on even a restricted family of weighted instances where the weight of each job is proportional to its processing time. More interestingly, the natural generalization of the SRPT to the weighted case, that schedules at each time instant the job with the smallest ratio of remaining processing time to weight, can be shown to have an $\Omega(\sqrt{P})$ competitive ratio on general weighted instances. The algorithm that schedules the job with the largest weight among the alive jobs has a competitive ratio of $\Theta(n)$.

**Results:** In this paper we address the weighted flow time problem. In contrast to the unweighted case, the problem with weights is known to be strongly NP-hard even on a single machine [9]. We give a *semi-online* algorithm for a single machine that is $O(\log^2 P)$-competitive. The algorithm is semi-online in that the parameter $P$, the ratio of the maximum job size to the minimum job size, is known in advance. It uses this information only to round the weights of the jobs in a suitable manner. We also show constant factor lower bounds on the deterministic and randomized competitiveness for weighted flow time on a single machine. Our algorithm, when viewed as an offline algorithm provides an $O(\log^2 P)$ approximation in polynomial time. Prior to our work no online algorithm or offline algorithm with a non-trivial ratio was known for the total weighted flow time measure. In the *offline* setting, we also show a $(2+\epsilon)$ approximation algorithm for the case when the weights and processing times are polynomially bounded, however our algorithm runs in quasi-polynomial time. Weighted flow time seems to be substantially harder in the parallel machine case. We show that no randomized online algorithm can achieve a competitive ratio better than $\Omega(\min\{\sqrt{P}, \sqrt{W}, (\frac{n}{m})^{1/4}\})$ on $m$ machines. Here $W$ is the ratio of max to min weights.

We also provide improved algorithms to minimize the total stretch on multiple machines. Previous results for this measure were based on analyzing algorithms natural for minimizing total flow time. We view the problem from a different perspective, as a special case of weighted flow time. Based on this view we develop a new and simple algorithm and show that it provides improved competitive ratios for the problem. Further our algorithm allows for a much simpler analysis.

**Related Work:** Awerbuch et al. [2] developed an online algorithm that does not migrate jobs between preemptions and showed that for unweighted flow time, the competitive ratio of their algorithm essentially matches that of SRPT. Becchetti et al. [4] showed that the algorithm in [2] provides an $O(1)$ ratio for the total stretch metric without migration. A metric that is closely related to flow time is completion time; they differ in value by an additive term that is independent of the schedule. Thus the optimal value for one metric directly yields the optimal value for the other. However they behave very differently in terms of approximability. For example, the single machine non-preemptive problem has a PTAS in the completion time metric (even with weights) [1] while it is NP-hard to approximate the flow time case to within a factor better than $n^{\frac{1}{2}-\epsilon}$ [8]. Considerable work on minimizing completion time has yielded many algorithms in the last few years, see [6, 1] for further references.

Phillips et al. [12] obtained algorithms for some variants of minimizing flow time in the model of Kalyanasundaram and Pruhs [7]. In this model the online (or offline) algorithm is allowed to use extra resources when compared to the adversary. In [12] several results are obtained for flow time when the online algorithm is allowed to use faster machines or more machines. In particular they give an online algorithm for preemptive weighted flow time that achieves the optimum offline schedule value but uses a machine of twice the speed. In contrast, the results in this paper are not based on resource augmentation.

**Model:** We are given a set $J$ of $n$ jobs that are released over time. For a job $x \in J$, the quantities $r(x)$, $p(x)$ and $w(x)$ denote the release time, processing time, and the weight respectively. The flow time of $x$ is $F(x) = C(x) - r(x)$ where $C(x)$ is the completion time of job $x$ in the schedule. Our objective is to minimize $\sum_{x \in J} w(x)F(x)$, the total weighted flow time. We assume that the processing time and weight of the job are known when the job enters the system and we allow preemption. Without loss of generality we make the assumption that all quantities of interest are integers. We denote by $P$ the maximum processing time of the jobs in the system.

**Organization:** The rest of the paper is organized as follows. In Section 2, we present an $O(\log^2 P)$ competitive semi-online algorithm for minimizing weighted flow time on a single machine. In Section 2.3 we show the lower bounds on weighted flow time for single and multiple machines. In Section 3 presents the offline algorithm for the same problem. For any $\epsilon > 0$, we give a $(2+\epsilon)$-approximation algorithm that runs in quasi-polynomial time and assumes that all processing times are bounded by some polynomial function of the total number of jobs. Finally, in Section 4, we present online algorithms for minimizing total stretch on multiple machines. We obtain better competitive ratios for this metric. In particular, when migration is allowed, we achieve a competitive ratio of 9.82 (improving upon the earlier ratio of 13), and in the absence of migration, we achieve a competitive ratio of 17.32 (improving upon the earlier ratio of 37). The analysis of our algorithm is simpler than that of SRPT and the one in [2]. Our algorithm simultaneously provides a logarithmic ratio for minimizing total flow time, thus retaining the advantages of earlier algorithms.

## 2. THE ONLINE ALGORITHM

We now present an $O(\log^2 P)$-competitive online algorithm for minimizing weighted flow time. We assume that the weights of the jobs are drawn from the set $\{w_1, w_2, ..., w_{\max}\}$, where $1 = w_1 \leq w_2 ... \leq w_{\max}$. As a first step we round up the weight of any incoming job to an integral power of $4(\log P + 1)$ and hence we can assume that $w_i = (4(\log P + 1))^{i-1}$. This will affect the competitive ratio by a factor of $4(\log P + 1)$. This is the only place where the knowledge of $P$ is used. All logarithms in the paper are to base 2 and for ease of notation we will assume that $\log P$ and $\log w_i$ are integers for all $i$. The *remaining processing time* of a job $x$ at time $t$ is denoted by $p_t(x)$. At any time $t$, we define the *class* of a job to be $\lfloor \log(p_t(x)/w(x)) \rfloor$. The class of a job is essentially the geometric interval in which the ratio of its remaining processing time to weight lies. Notice that the class is an integer in $[-\log w_{\max}, \log(P)]$.

In our algorithm and its analysis we are interested in sets of jobs that are alive at a specific time and the partition induced on them by the weights and classes. We specify a subset of jobs alive at time $t$ by $Q_{p,q}(t)$ where $p$ and $q$ are predicates on integers. A job $x$ belongs to the set $Q_{p,q}(t)$ if the weight index of $x$ satisfies $p$ and the class of $x$ at time $t$ satisfies $q$. Thus $Q_{=i,=j}(t)$ is the set of all jobs that are alive at time $t$ and have weight $w_i$ and belong to class $j$ at time $t$. For brevity, $Q_{i,j}$ is the same as $Q_{=i,=j}$. Similarly $Q_{\leq i,>j}$ is the set of jobs alive at $t$ with weight at most $w_i$ and class greater than $j$ at time $t$. The absence of subscripts, as in $Q(t)$, indicates that all jobs alive at $t$ are included.

Two quantities of interest for a subset of jobs $Q_{p,q}(t)$ are the total weight, denoted by $W(Q_{p,q}(t))$, and the *volume* or the sum of remaining processing times, denoted by $V(Q_{p,q}(t))$. We use $W_{p,q}(t)$ and $V_{p,q}(t)$ as short hand for $W(Q_{p,q}(t))$ and $V(Q_{p,q}(t))$ respectively.

## 2.1 Algorithm Description

Without loss of generality we assume that time is discrete and all release dates are integral. The online algorithm is specified by the rules that choose a job in $Q(t)$ to execute at each discrete time step $t$. In the following

1. $i \leftarrow$ largest weight index in $Q(t)$.
2. $k \leftarrow$ smallest class in jobs of weight $w_i$ in $Q(t)$.
3. If $W_{<i,<k}(t) \leq w_i$, schedule a job of weight $w_i$ and class $k$ (a job from $Q_{i,k}(t)$).
4. Else $i \leftarrow$ largest weight index in $Q(t)$ strictly less than $i$. Go to Step 2.

Informally speaking, the algorithm greedily tries to schedule a job of lowest class at any time $t$ with a "bias" towards jobs of larger weight. We will refer to this algorithm as the *biased greedy* algorithm.

**Intuition:** As we mentioned earlier, the algorithm that schedules the job from the lowest class has an $\Omega(\sqrt{P})$ competitive ratio. We motivate the design of our biased greedy algorithm by giving a bad example for the naive greedy algorithm. At $t_1 = 0$, $L$ large jobs, of processing time $L$ and weight 1, and a single huge job, of processing time $L^2$ and weight $L$, are released. Notice that the large jobs, and the huge job, have the same processing time to weight ratio. Starting at $t_2 = 2L^2 - L$, $K \gg L$ small jobs, of processing time 1 and weight 1, are released, one each at each integral time unit. Breaking ties adversarially, the algorithm can be made to process all the large jobs first, and hence, at time $t_2$, the remaining processing time of the huge job is $L$. At $t_2$, the algorithm switches to the small jobs, and hence, throughout the processing of the small jobs, the queue weight is $L$. On the other hand, the optimal algorithm finishes the huge job first, and hence has a weight of 1 in the queue while executing the small jobs. If we make $K$ large enough, the competitive ratio of the algorithm approaches $L$.

The naive greedy algorithm is stuck with a job of huge weight when many small jobs of much better ratio appear. Thus, it pays to finish the larger weight jobs first. On the other hand, the algorithm that always schedules the largest weight job, without regard to its processing time to weight ratio, also performs badly (has an $\Omega(P)$ competitive ratio). Consider the example of a job of weight 2 and processing time $P$, delaying $P$ jobs of weight 1 and processing time 1. Our algorithm strikes a balance in this regard. We keep processing a large weight job, without regard to its ratio, only as long as the total weight of jobs of strictly better ratio does not get too large. And we apply this principle recursively. Within the same weight we prefer the smaller jobs, as SRPT does. Though the algorithm is stated simply, its analysis is quite tricky.

## 2.2 Analysis

We denote by $t_{i,k}$ the last time before $t$ when a job of weight $w_i$ and of class *greater* than $k$ is executed. If such an event does not exist we set $t_{i,k}$ to 0. We assume w.l.o.g. that the first job is released at $t = 0$ and that the algorithm is busy throughout the sequence.

For the rest of the paper, a superscript of $^*$ indicates that the quantities in question refer to some fixed optimal schedule. The objective function is $\sum_x w(x)F(x)$ where $F(x)$ is the flow time of job $x$. However, for the analysis, it is easier to work with a reformulation of the objective function as $\sum_t W(t)$ where $W(t)$ is the sum of weight of the jobs that are alive at $t$. Our goal will be to show that $W(t) \leq \alpha W^*(t)$ for all $t$, where $\alpha$ will be our competitive ratio. Let $\mathrm{cl}(t)$ and $\mathrm{wg}(t)$ denote the class and weight index of the job executing at time $t$. The following propositions relate the volume of jobs in various classes relative to $\mathrm{cl}(t)$ and $\mathrm{wg}(t)$.

Among jobs of a given class the algorithm always schedules the job of the largest weight, hence the following.

PROPOSITION 1. *At any time $t$, $Q_{>wg(t),\leq cl(t)}(t)$ is empty and therefore $V_{>wg(t),\leq cl(t)}(t) = 0$.*

We bound the volume of jobs that are of a better class than the class of the job running at time $t$.

PROPOSITION 2. *Let $\ell = wg(t)$. Then, for all $k < cl(t)$,*

$$V_{<\ell,\leq k}(t) \leq w_\ell \cdot 2^{k+1}.$$

PROOF. From the algorithm description it follows that $W_{<\ell,\leq k}(t) \leq w_\ell$ for all $k < \mathrm{cl}(t)$. From the definition of class it is easy to see that $V_{<\ell,\leq k}(t) \leq 2^{k+1} \cdot W_{<\ell,\leq k}(t)$. □

The following basic lemma, which we apply repeatedly, upper bounds the weight of jobs in the algorithm's queue in specific ranges of weights and classes. The idea is to relate the weight to the differences in the volume of jobs between the algorithm and the optimal. We use the notation $\Delta V_{i,j}(t)$ to denote the quantity $V_{i,j}(t) - V^*_{i,j}(t)$. Lemmas in the same spirit, but for the unweighted case, can be found in [10, 2, 11].

LEMMA 2.1.

$$\sum_{k_1 \leq k \leq k_2} W_{\leq \ell,k}(t) \leq 2 \sum_{k \leq k_2} W^*_{\leq \ell,k}(t) + \frac{\Delta V_{\leq \ell,\leq k_2}(t)}{2^{k_2}} + \sum_{k_1 \leq k < k_2} \frac{\Delta V_{\leq \ell,\leq k}(t)}{2^{k+1}}.$$

PROOF. We drop $t$ in the following.

$$\begin{aligned}\sum_{k_1 \leq k \leq k_2} W_{\leq \ell,k} &\leq \sum_{k_1 \leq k \leq k_2} \frac{V_{\leq \ell,k}}{2^k} \\ &\leq \sum_{k_1 \leq k \leq k_2} \frac{\Delta V_{\leq \ell,k} + V^*_{\leq \ell,k}}{2^k} \\ &\leq \sum_{k_1 \leq k \leq k_2} \frac{V^*_{\leq \ell,k}}{2^k} + \sum_{k_1 \leq k \leq k_2} \frac{\Delta V_{\leq \ell,k}}{2^k}\end{aligned}$$

$$
\begin{aligned}
&\leq 2 \sum_{k_1 \leq k \leq k_2} W^*_{\leq \ell, k} + \\
&\qquad \sum_{k_1 \leq k \leq k_2} \frac{\Delta V_{\leq \ell, \leq k} - \Delta V_{\leq \ell, \leq k-1}}{2^k} \\
&\leq 2 \sum_{k_1 \leq k \leq k_2} W^*_{\leq \ell, k} - \frac{\Delta V_{\leq \ell, \leq k_1 - 1}}{2^{k_1 - 1}} + \\
&\qquad \frac{\Delta V_{\leq \ell, \leq k_2}}{2^{k_2}} + \sum_{k_1 \leq k < k_2} \frac{\Delta V_{\leq \ell, \leq k}}{2^{k+1}} \\
&\leq 2 \sum_{k_1 \leq k \leq k_2} W^*_{\leq \ell, k} + \frac{V^*_{\leq \ell, \leq k_1 - 1}}{2^{k_1 - 1}} + \\
&\qquad \frac{\Delta V_{\leq \ell, \leq k_2}}{2^{k_2}} + \sum_{k_1 \leq k < k_2} \frac{\Delta V_{\leq \ell, \leq k}}{2^{k+1}} \\
&\leq 2 \sum_{k \leq k_2} W^*_{\leq \ell, k} + \frac{\Delta V_{\leq \ell, \leq k_2}}{2^{k_2}} + \\
&\qquad \sum_{k_1 \leq k < k_2} \frac{\Delta V_{\leq \ell, \leq k}}{2^{k+1}}.
\end{aligned}
$$

In the penultimate step we upper bound $-\Delta V_{\leq \ell, \leq k_1 - 1}$ by $V^*_{\leq \ell, \leq k_1 - 1}$, easy to see since $V_{\leq \ell, \leq k_1 - 1} \geq 0$. $\square$

Crucial to our analysis is the definition of the weight *regime* at time $t$, denoted by $r(t)$. Let $\mathrm{lw}(t)$ denote the largest weight index in the algorithm's queue at time $t$. If $\mathrm{wg}(t) < \mathrm{lw}(t)$ a job of lower weight is scheduled at $t$ because of the rule in Step 3 of the algorithm. This establishes a lower bound of $w_{\mathrm{lw}(t)}$ on the weight of jobs in $Q_{\leq \mathrm{lw}(t)-1}$. This lower bound is particularly useful if it is established at a critical time: the last time at which a certain class is executed. We denote by $t_{\leq \ell, k}$, the last time before $t$ when the algorithm executed a job of class greater than $k$ in weight indices at most $\ell$. We define the regime at $t$ to be the following.

$$
r(t) = \min\{j | \max\{\mathrm{lw}(t), \mathrm{lw}^*(t)\} \leq j \text{ and } \forall k, \mathrm{lw}(t_{\leq j, k}) \leq j\}
$$

Our first observation is that the regime is well defined since the index of the largest weight job released by time $t$ satisfies the requirements. However in general it could be much smaller.

Informally speaking, the notion of the regime is used in our analysis in the following way. Let $\ell = r(t)$. We will be able to prove that $\Delta W(t) = O(\log P) w_\ell$ and that $W(t) = \Omega(w_\ell)$. However, at $t$, the queue of the optimal algorithm might not have any weight $w_\ell$ job, or even jobs of comparable weight. To be competitive, we will have to prove a lower bound of $\Omega(w_\ell)$ on $W^*(t)$. In order to accomplish this, we will make use of the following: if the regime at $t$ is $\ell$, then the algorithm has a weight of at least $\Omega(w_\ell)$ in jobs of weights $w_1$ to $w_{\ell-1}$! This is where we crucially use the algorithm's property: if a weight $w_j$ job is scheduled at time $t$ while there is a job of weight $w_\ell$ in the queue, $j < \ell$, then the algorithm's queue has a weight of at least $w_\ell$ in jobs of weight 1 to $\ell - 1$, for otherwise we would have scheduled a job of weight $w_\ell$. We will also be able to argue, if $Q^*(t)$ is empty in $w_\ell$ job, that $W_{\leq \ell - 1}(t) - W^*_{\leq \ell - 1}(t)$ is $O(w_{\ell-1} \log P)$. These claims allow us to lower bound $W^*(t)$.

The following lemma establishes an upper bound on the weight of the algorithm's queue in terms of the weight regime at $t$.

LEMMA 2.2. *Let $\ell = r(t)$. At any time $t$, $W(t) \leq 2(\log P + 4) \cdot w_\ell + 2W^*(t)$.*

PROOF. Consider the time $t_{\leq \ell, k}$, the last time before $t$ when the algorithm executed a job of class greater than $k$ in weight indices at most $\ell$. We claim that $\mathrm{lw}(t_{\leq \ell, k}) \leq \ell$ for all $k$. This follows directly from the definition of the regime. Let $j_k = \mathrm{wg}(t_{\leq \ell, k})$. Since $\ell \geq \max\{\mathrm{lw}(t), \mathrm{lw}^*(t)\}$ neither $Q(t)$ nor $Q^*(t)$ has a job of weight index greater than $\ell$. Hence, in the time interval $[t_{\leq \ell, k}, t)$, the biased greedy algorithm spent no more time than the optimal algorithm on jobs of weight greater than $\ell$. Therefore it follows that

$$
\Delta V_{\leq l, \leq k}(t) \leq V_{\leq \ell, \leq k}(t_{\leq \ell, k}).
$$

We now proceed to upper bound the rhs of the above inequality. Let $t' = t_{\leq \ell, k}$.

$$
\begin{aligned}
V_{\leq \ell, \leq k}(t') &= V_{< j_k, \leq k}(t') + V_{j_k, \leq k}(t') + V_{> j_k, \leq k}(t') \\
&= V_{< j_k, \leq k}(t') + V_{j_k, \leq k}(t') + 0 \text{ (Prop 1)} \\
&\leq w_{j_k} 2^{k+1} + V_{j_k, \leq k}(t') \qquad \text{(Prop 2)} \\
&\leq w_{j_k} 2^{k+1} + w_{j_k} 2^{k+1} = w_{j_k} 2^{k+2}.
\end{aligned}
$$

In the last inequality above we bound $V_{j_k, \leq k}(t_{\leq \ell, k})$ by $w_{j_k} 2^{k+1}$ since the job of weight $w_{j_k}$ can enter class $k$ just at $t_{\leq \ell, k}$. The volume of that job can be at most $w_{j_k} 2^{k+1}$.

We observe that the class range of a job of weight $w_j$ is $[-\log w_j, \log P - \log w_j]$. Hence for any class $k$ we can assume that $k \leq \log P - \log w_{j_k}$.

Applying Lemma 2.1 we have that

$$
\begin{aligned}
W(t) &= \sum_{-\log w_\ell \leq k \leq \log P} W_{\leq \ell, k}(t) \\
&\leq 2W^*(t) + \frac{\Delta V_{\leq \ell, \log P}(t)}{2^{\log P}} + \sum_{-\log w_\ell \leq k < \log P} \frac{\Delta V_{\leq \ell, \leq k}(t)}{2^{k+1}} \\
&\leq 2W^*(t) + 4w_1 + \sum_{-\log w_\ell \leq k < \log P} \frac{w_{j_k} 2^{k+2}}{2^{k+1}}.
\end{aligned}
$$

We next upper bound the sum $\sum_{-\log w_\ell \leq k < \log P} w_{j_k}$. Observe that the class range of a job of weight $w_j$ is $[-\log w_j, \log P - \log w_j]$. Hence the sum is maximized when we assign the weight $w_\ell$ to the lowest $\log P$ classes, weight $w_{\ell-1}$ to the next $(\log P - \log w_{\ell-1}) - (\log P - \log w_\ell)$ classes, and so on. Thus

$$
\sum_{-\log w_\ell \leq k < \log P} w_{j_k} \leq w_\ell \log P + \sum_{\ell \geq j > 1} w_{j-1} (\log w_j - \log w_{j-1}).
$$

Substituting this bound in the above, we get

$$
\begin{aligned}
W(t) &\leq 2W^*(t) + 4w_1 + 2w_\ell \log P \\
&\qquad + 2 \sum_{\ell \geq j > 1} w_{j-1} (\log w_j - \log w_{j-1}) \\
&\leq 2W^*(t) + 2w_\ell \log P + 8(\log P + 1) \sum_{\ell \geq j \geq 1} w_{j-1} \\
&\leq 2W^*(t) + 2w_\ell \log P + 8w_\ell \\
&\leq 2W^*(t) + 2w_\ell (\log P + 4).
\end{aligned}
$$

This finishes the proof. $\square$

Now we prove the lower bound on $W^*(t)$.

LEMMA 2.3. *Let $\ell = r(t)$. At any time $t$, $W^*(t) \geq w_\ell/4$.*

PROOF. If $\mathrm{lw}^*(t) = \ell$ we are done. Hence assume $\mathrm{lw}^*(t) < \ell$. Consider the sequence $t_{\leq \ell-1,k}$ as $k$ decreases. We claim that there exists a $k'$ such that $\mathrm{lw}(t_{\leq \ell-1,k'}) \geq \ell$. To prove this claim we consider two cases based on the relationship between $\ell$ and $\mathrm{lw}(t)$.

- $\ell > \mathrm{lw}(t)$. If $\mathrm{lw}(t_{\leq \ell-1,k}) \leq \ell - 1$ for all $k$, it would contradict the choice of $\ell$ as the minimum $j$ for which this condition is true.
- $\ell = \mathrm{lw}(t)$. We are under the assumption that $\mathrm{lw}^*(t) < \ell$. Let $t'$ be the last time before $t$ when the biased greedy algorithm scheduled a job of weight $w_{\ell-1}$ or less. We claim that $\mathrm{lw}(t') \geq \ell$. Suppose not. In the interval $[t', t)$ the biased greedy algorithm did not process any jobs of weight less than $w_\ell$ and it had no jobs of weight $w_\ell$ or greater at $t'$. Hence it cannot have a deficit in the total amount of remaining processing time in jobs of weight $w_\ell$ or greater at time $t$. However this contradicts the fact that $\mathrm{lw}^*(t) < \ell = \mathrm{lw}(t)$. Therefore at $t'$, $\mathrm{lw}(t') \geq \ell$. Let $k'$ be the class of the job processed at $t'$. Since we did not process any job of weight $w_{\ell-1}$ or less after $t'$ it follows that $t_{\leq \ell-1,k'-1} = t'$.

Let $h$ be the largest $k$ such that $\mathrm{lw}(t_{\leq \ell-1,k}) \geq \ell$. From the definition of $h$ it follows that $\mathrm{cl}(t_{\leq \ell-1,h}) = h + 1$. We claim that $\Delta V_{\leq \ell-1,\leq k}(t) \leq w_{\ell-1} 2^{k+2}$ for all $k > h$. This is because $\mathrm{lw}(t_{\leq \ell-1,k}) \leq \ell - 1$ for $k > h$, and in the interval $[t_{\leq \ell-1,k}, t]$, the amount of time spent by the algorithm on weights $\leq \ell - 1$ is no less than that of the optimal algorithm. Let $j = \log(P/w_\ell)$ -1 – one less than the largest possible class for a job of weight $w_\ell$. We now derive a lower bound on the weight in the queue of the optimal algorithm in terms of the quantity $\sum_{h<k\leq j} W_{\leq \ell-1,k}(t)$ by using the above claim and Lemma 2.1.

$$\begin{aligned}
\sum_{h<k\leq j} W_{\leq \ell-1,k}(t) &\leq 2W^*(t) + \frac{\Delta V_{\ell-1,\leq j}}{2^j} + \sum_{h<k<j} \frac{\Delta V_{\leq \ell-1,\leq k}}{2^{k+1}} \\
&\leq 2W^*(t) + \frac{w_{\ell-1}2^{j+2}}{2^j} + \sum_{h<k<j} \frac{w_{\ell-1}2^{k+2}}{2^{k+1}} \\
&\leq 2W^*(t) + 4w_{\ell-1} + 2w_{\ell-1}(j - h - 1) \\
&\leq 2W^*(t) + 2w_{\ell-1}\log P.
\end{aligned}$$

The last inequality uses the fact that $h \geq -\log w_{\ell-1}$.

To finish the proof, we lower bound $\sum_{h<k\leq j} W_{\leq \ell-1,k}(t)$, the weight that the algorithm has in its queue in classes $h+1$ to $j$. At $t_{\leq \ell-1,h}$ we executed a job of weight class at most $\ell - 1$ while $\mathrm{lw}(t_{\leq \ell-1,h}) \geq \ell$. From the rule in Step 3 of the algorithm, it follows that $\sum_{k\leq j} W_{\leq \ell-1,k}(t_{\leq \ell-1,h}) \geq w_\ell$. This holds whether there exists a job of weight $w_\ell$ at $t_{\leq \ell-1,h}$ or not, the bound only improves if there is no $w_\ell$ job, for in that case $\mathrm{lw}(t_{\leq \ell-1,h}) > \ell$.

We claim that the quantity $\sum_{k\leq h} W_{\leq \ell-1,k}(t_{\leq \ell-1,h})$ is upper bounded by $w_{\ell-1}$, for otherwise we would have executed a job of class at most $h$, contradicting the definition of $t_{\leq \ell-1,h}$. The two claims imply that $\sum_{h<k\leq j} W_{\leq \ell-1,k}(t_{\leq \ell-1,h}) \geq w_\ell - w_{\ell-1}$. By the definition of $t_{\leq \ell-1,h}$, the algorithm did not execute any job of weight less than $\ell$ of a class greater than $h$ in the interval $[t_{\leq \ell-1,h}, t)$. Hence, those jobs are alive at $t$, and we get the following.

$$\sum_{h<k\leq j} W_{\leq \ell-1,k}(t_{\leq \ell-1,h}) = \sum_{h<k\leq j} W_{\leq \ell-1,k}(t) \geq w_\ell - w_{\ell-1}.$$

Comparing the two bounds on $\sum_{h<k\leq j} W_{\leq \ell-1,k}(t)$ we conclude that

$$\begin{aligned}
2W^*(t) &\geq w_\ell - 2w_{\ell-1}(\log P + 1) \\
&\geq w_\ell/2.
\end{aligned}$$

Here we use the fact that the weights have been rounded up to power of $4(\log P + 1)$. $\square$

THEOREM 1. *The biased greedy algorithm is $O(\log^2 P)$-competitive.*

PROOF. Putting together Lemmas 2.2 and 2.3 we obtain that $W(t) \leq O(\log P)W^*(t)$. This yields a competitive ratio of $O(\log P)$, however the weights of the jobs have been rounded up to the nearest power of $\log P$. Hence, for the original instance, we obtain a competitive ratio of $O(\log^2 P)$. $\square$

## 2.3 Lower Bounds for Online Algorithms

### *2.3.1 Single Machine*

We show a simple lower bound of 1.618 on the competitiveness of any deterministic online algorithm for minimizing weighted flow time on a single machine. We also a show a 4/3 lower bound for randomized competitiveness.

Let $\mathcal{A}$ be any deterministic algorithm. Consider the following behaviour of an adversary. At time 0 two jobs are released. Job $J_1$ has weight $L$ and processing time $2L$. Job $J_2$ has weight $aL$ and a processing time $L^2$. No more jobs are released till time $t = L^2$. If $\mathcal{A}$ finishes $J_2$ by $t$ then the adversary gives no more jobs. If $\mathcal{A}$ finishes $J_1$ by $t$ then the adversary gives $n$ more jobs $J_3, J_4, \ldots, J_{n+2}$ each with weight 1 and processing time 1. $J_i$ is released at $t+i-3$ for $3 \leq i \leq n$. We claim that if we choose $a$ to be $(1+\sqrt{5})/2 \simeq 1.618$, then in either case the algorithm loses a factor of $a$ over the optimal offline schedule provided $L$ and $n$ are sufficiently large.

For the randomized lower bound we consider two instances as described above: $I_1$ in which no jobs other than $J_1$ and $J_2$ are released and $I_2$ in which $n$ more jobs are released at $t$. We further set the weight of $J_2$ to be $aL$ where $a = 2$. We consider a probability distribution on instances where $I_1$ is given with probability 2/3 and $I_2$ with probability 1/3. For the analysis purpose it is easy to see that we can restrict attention to two deterministic algorithms – one that schedules $J_1$ before $J_2$ and the other vice versa. The expected competitiveness of both these algorithms on the above distribution is 4/3. This proves the bound.

THEOREM 2. *For minimizing weighted flow time on a single machine no deterministic online algorithm has a competitive ratio better than* 1.618 *and no randomized online algorithm has a competitive ratio better than* 4/3

### 2.3.2 Multiple Machines

Leonardi and Raz [10] establish a lower bound of $\Omega(\min\{\log P, \log \frac{n}{m}\})$ on the competitiveness of any randomized online algorithm for minimizing unweighted flow time on $m > 1$ machines. This is in contrast to the single machine case where SRPT is an optimal online algorithm. The fundamental difficulty with multiple machines is the inability of the online algorithm to load balance in the face of uncertainty. Thus the online algorithm is forced to under utilize machine resources. We show that this phenomenon has a much bigger impact when weights are allowed. In particular, we establish a lower bound of $\Omega(\min\{\sqrt{P}, \sqrt{W}, (\frac{n}{m})^{1/4}\})$ on the competitiveness of any randomized online algorithm for weighted flow time on $m > 1$ machines. We first sketch an argument that shows a deterministic lower bound and then extend it to the randomized case.

Let $\mathcal{A}$ be any deterministic algorithm. In the following $L$ will be a sufficiently large integer to be specified later. The adversary gives $m+1$ jobs at time 0. $J_1, \ldots, J_m$ are identical with weight $L$ and processing time $L$ each. $J_{m+1}$ has weight $\sqrt{L}$ and processing time $2L$. The adversary gives no other jobs till $t_1 = L$. Let $x$ be the remaining processing time of $J_{m+1}$ at $t'$ in the queue of $\mathcal{A}$. If $x < 3L/2$, the adversary releases $m$ identical jobs of weight 2 and processing time 1 each, at unit intervals, starting at $t_1$, for a duration of $L^2$ time units. If $x \geq 3L/2$ the adversary releases the same set of small jobs but releases them starting at $t_2 = 2L$. Let the former instance be $I_1$ and the latter $I_2$. Now we argue that in either case the algorithm's competitive ratio is $\Omega(\sqrt{L})$.

If $x < 3L/2$ then by conservation of volume, collectively $J_1, \ldots, J_m$ have at least $L/2$ processing time left at $t_1$. Let $t' > t_1$ be the time at which the algorithm finishes the last of these remaining jobs. Then during the interval $[t_1, t']$ the algorithm carries a weight of at least $L$ in its queue. If $t' < t_1 + L^2$, then by volume conservation, the algorithm must carry at least $L/2$ small jobs in its queue during the interval $[t', t_1 + L^2]$ - a weight of at least $L$. It follows that the value of the schedule will be $\Omega(L^3 + mL^2)$. On the other hand the optimal offline algorithm will finish $J_1, \ldots, J_m$ and carry $J_{m+1}$ for the duration of the execution of the small jobs. Hence the optimal offline schedule values is $O(L^{5/2} + mL^2)$.

If $x \geq 3L/2$ then $\mathcal{A}$ will have at least $L/2$ remaining processing time for $J_{m+1}$ at $t_2 = 2L$. In this case the best strategy for $\mathcal{A}$ is to finish all the small jobs before finishing $J_{m+1}$. Hence, the schedule value is $\Omega(L^{5/2} + mL^2)$. The offline optimal schedule finishes $J_1, \ldots, J_{m+1}$ by $t_2$. It does this by dedicating a machine to finish $J_{m+1}$ and stacking up $J_1$ and $J_2$ on another machine and finishing $J_3$ to $J_m$ on a machine each. The value of the schedule is $O(mL^2)$.

In either case if we choose $L$ to be sufficiently larger than $m$ the competitive ratio of $\mathcal{A}$ is seen to be $\Omega(\sqrt{L})$. In the above instance $P = W = L$. Further, $n = O(mL^2)$. We obtain the claimed lower bound from the discussion above. It is easy to extend the lower bound to randomized algorithms. Consider a distribution on instances where $I_1$ and $I_2$ have probability 1/2 each. The arguments above show that any deterministic algorithm will have a competitive ratio of $\Omega(\sqrt{L})$ on at least one of them.

THEOREM 3. *No online randomized algorithm for minimizing weighted flow time on $m > 1$ parallel machines can be $o(\min\{\sqrt{P}, \sqrt{W}, (\frac{n}{m})^{1/4}\})$ competitive.*

## 3. THE OFFLINE ALGORITHM

Our algorithm in Section 2 provides a $O(\log^2 P)$ approximation in polynomial time. Here we present a quasi-polynomial time $(2+\epsilon)$-approximation algorithm for weighted flow-time, which gives strong evidence that the problem is approximable to within a constant factor. Our result assumes that the processing times are bounded by a polynomial function of the number of jobs. We observe that even this case is NP-hard.

Our starting point is a partition of jobs into sets such that each set contains jobs of identical weight and essentially similar processing times. Let $w_1 < w_2, < \ldots, < w_{\max}$ be the set of distinct weights of the jobs. Fix a constant $\delta \in (0, 1)$. We say a job $x$ is in processing group $j$ if its processing time $p(x)$ satisfies the condition $(1+\delta)^j \leq p(x) < (1+\delta)^{j+1}$. Let $S_{ij}$ denote all jobs of with weight $w_i$ and processing time group $j$.

**Arrival-ordered schedules:** We say a schedule $\sigma$ is *arrival-ordered* with respect to a subset of jobs $S$ if it executes jobs in $S$ in the strict order of their arrival (ties broken arbitrarily). Let $|\sigma|$ denote the flow-time of a schedule $\sigma$. The lemma below gives a useful property of arrival-ordered schedules.

LEMMA 3.1. *For $\delta \in (0, 1)$, any given schedule $\sigma$ can be transformed into a schedule $\sigma'$ such that, $\sigma'$ is arrival ordered with respect to each of the sets $S_{ij}$, and $|\sigma'| \leq 2|\sigma|$.*

PROOF. Starting from the schedule $\sigma$, we construct the schedule $\sigma'$ by simply reordering the execution of jobs in each set $S_{ij}$, in the order of arrival. Let $n_{ij}(t)$ and $n'_{ij}(t)$ denote the number of jobs in $S_{ij}$ that remain at time $t$ in the schedules $\sigma$ and $\sigma'$ respectively. For any set $S_{ij}$ and time $t$, since the total volume in $S_{ij}$ that is processed in $\sigma$ by time $t$ is same as the volume processed in $\sigma'$, we get

$$n'_{ij}(t) \leq \lfloor n_{ij}(t)(1+\delta) \rfloor + 1.$$

Using the fact that $\delta \in (0, 1)$, we can conclude that $n'_{ij}(t) \leq 2n_{ij}(t)$ at all times $t$. The lemma follows. $\square$

**Approximating arrival-ordered schedules:** We will now design a quasi-polynomial time algorithm to find a $(1+\epsilon)$-approximate arrival-ordered schedule. Let $W$ and $P$ denote the largest weight and the largest processing time, respectively, over all jobs. We assume without loss of generality that no two release dates are separated by more than $nP$; otherwise the instance can be partitioned into many disjoint instances that satisfy this property. Thus any non-idling schedule can complete all jobs within $O(n^2P)$ time.

We now use dynamic programming to compute a $(1+\epsilon)$-approximate arrival-ordered schedule. We partition jobs into two sets $X_1$ and $X_2$ such that $X_1 = \{x \mid w(x) > W/(n^4P)\}$. We will first construct a schedule for the jobs in $X_1$, and then fill in an arbitrary schedule for jobs in $X_2$, always giving preference to jobs in $X_1$. Clearly jobs in $X_1$ are not affected. The total contribution of $X_2$ to the schedule value is bounded by $n \cdot n^P \cdot W/(n^4P)$. Since the cost of any schedule for jobs in $X_1$ is at least $W$, the contribution of $X_2$ is no more than an $o(1)$-factor.

To construct a schedule for jobs in $X_1$, we first do a preprocessing step. We scale all weights by $W/(n^4P)$ and round the scaled weight of each job upwards to the nearest power of $(1+\epsilon)$. Let $a = \lceil \log_{1+\delta} P \rceil$ and let $b = \lceil \log_{1+\epsilon}(n^4P) \rceil$.

Thus, $a$ is the number of distinct processing groups, and $b$ is the number of distinct weights.

The idea underlying the dynamic program is to maintain at each time $t$, a pair of state vectors of the form $\vec{n} = \langle n_{11}(t), n_{12}(t), ..., n_{ab}(t)\rangle$, and $\vec{l} = \langle l_{11}(t), l_{12}(t), ..., l_{ab}(t)\rangle$. The component $n_{ij}(t)$ of the vector $\vec{n}$ indicates the number of jobs in $S_{ij}$ that have been completed by time $t$, while the component $l_{ij}(t)$ of the vector $\vec{l}$ denotes the amount of time that has been spent on the $(n_{ij}+1)$th job in the set $S_{ij}$. Notice that since we are considering only arrival-ordered schedules, the vector $\vec{n}$ completely describes the identity of the jobs that have been completed so far, and the vector $\vec{l}$ describes the state of (at most) one partially executed job in each class $S_{ij}$. We iterate over $t$ to compute a table $Z$ such that $Z[t, \vec{n}, \vec{l}]$ is the minimum flow time for the completed jobs with the state specified by $\vec{n}$ and $\vec{l}$ at time $t$. If the state is infeasible, we set the table entry to $\infty$. The running time of this dynamic program can be bounded by $O(n^2 P \cdot n^{ab} \cdot P^a)$. Since $P$ is polynomially bounded in $n$, the running time is bounded by $n^{O(\log^2 n)}$. We omit further details from this version.

THEOREM 4. *Given an instance of weighted flow-time problem on a single machine with polynomially bounded processing times, there is an algorithm that computes a $(2+\epsilon)$-approximate solution in quasi-polynomial time for any fixed $\epsilon > 0$.*

# 4. MINIMIZING TOTAL STRETCH

In this section we give new online algorithms for minimizing total stretch on multiple processors and improve the performance guarantees over earlier algorithms. We consider two models based on whether or not migration of jobs is allowed or not. If migration is allowed, a job that is preempted, is free to be rescheduled on any machine in the system. If migration is not allowed, a job once scheduled on a machine, has to complete its processing only on that machine. When migration is allowed Muthukrishnan et al. [11] show that SRPT achieves a competitive ratio of 13. Becchetti et al. [4] analyzed the non-migratory algorithm of Awerbuch et al. [2] for total stretch and showed that it achieves a competitive ratio of 37. The above results are based on analyzing algorithms that have been originally devised for minimizing total unweighted flow time. As remarked upon earlier, the stretch measure, when viewed from a weighted flow time perspective, favors small jobs more so than total flow time. We give both migratory and non-migratory algorithms using a similar idea that specifically takes advantage of the stretch measure. We show that our algorithms achieve a competitive ratio of 13 and 19 for migratory and non-migratory models respectively. An advantage of our approach is that it lends itself to substantially simpler analysis when compared to those in [11] and [4]. This allows us to parameterize the algorithm and optimize the parameter to obtain improved ratios of 9.82 and 17.32 for the migratory and non-migratory models respectively. We can also show that our algorithms achieve an $O(\min\{\log P, \log \frac{n}{m}\})$ ratio for minimizing total flow time, matching the performance of SRPT. Thus our algorithms retain the advantage of SRPT in being competitive for both stretch and flow time.

## 4.1 Minimizing Stretch without Migration

We first describe the non-migratory version of the algorithm. We partition the jobs in to *groups* based on their processing times. A job $x$ with $p(x) \in [2^i, 2^{i+1})$ is said to be in group $i$. We use $G_i$ to denote the jobs in group $i$. Note that unlike the definition of *class* used in Section 2 (and in [2]), the *group* of a job is defined by its processing time and does not change during the execution of the algorithm. Let $g(x)$ denote the group of job $x$. The algorithm is very similar to the one in [2]. The algorithm maintains a pool of jobs that have arrived but have not been processed at all. In addition it also maintains a stack of jobs for each machine, those that have already been partially processed by that machine. Once a job is assigned to the stack of machine $i$ it is committed to being processed by $i$, and hence the algorithm is non-migratory. The algorithm works as follows:

- Each machine processes the job at the top of its stack.
- When a new job arrives, the algorithm looks for a machine that is either idle or is currently processing a job of a higher *group* than that of the new job. If such a machine exists, the new job is pushed on top of that machine's stack and its processing begins. Otherwise, the job is inserted into the pool.
- If a job is completed on a machine, it is removed from its stack. If the stack becomes empty, or if the top of the stack is of a higher *group* than the minimum group of the jobs in the pool, a job from the pool from the jobs of the minimum group is moved to the machine's stack.

The only difference between the above algorithm, which we refer to as SG (for smallest group), and the one in [2] is that the decisions are based on the group of the job instead of its class. The notion of class ignores the original processing time of the job and retains only the current remaining processing time. This is a reasonable strategy for minimizing total flow time where all jobs have the same weight irrespective of their processing time. When minimizing total stretch it is advantageous to retain information about the weight of the jobs. The SG algorithm, unlike SRPT and the algorithm of [2], can preempt jobs with tiny remaining processing time by jobs of much larger remaining processing time. However, since the preempting job is of a strictly smaller group, we will be able to charge the *weight* of the preempted jobs to that of the preempting job.

Now we proceed with the analysis. Let $m(t)$ denote the number of busy machines at time $t$. Let $S(t)$ be the set of jobs in the stacks at time $t$ and let $P(t)$ be the set of jobs in the pool. Let $n$ be the number of jobs released in the sequence. The following proposition states a simple lower bound on stretch of any schedule (see also [11]).

PROPOSITION 3. OPT $\geq n$.

The following lemma bounds the contribution to the total stretch of all the jobs in the stacks.

LEMMA 4.1. $\sum_t \sum_{x \in S(t)} w(x) \leq 3n$.

PROOF. Let $R(t)$ denote the set of jobs being processed at time $t$. We claim that $\sum_t \sum_{x \in R(t)} w(x) = n$ since each job

$x$ is in the set $R(t)$ for exactly a duration of $p(x)$ time with $w(x) = 1/p(x)$. Consider the jobs in a machine's stack that are not being processed. There is at most one job from each group. Further, each of them is from a strictly larger group than that of the job being processed. We upper bound the weight of jobs in each stack by rounding up the weight of a job in group $i$ to $1/2^i$ and summing up over all groups larger than the group of the job being processed. It is easy to see that this weight is at most twice the weight of the job being processed. Hence we can charge the weight of the stacks to the jobs in $R(t)$, and the lemma follows. □

We now bound the weight of the jobs in the pool at each time $t$. Let $V_i(t)$ denote the volume of the jobs in $P(t) \cap G_i$ — this is the total volume of jobs in the pool from $G_i$ at time $t$. Let $V_{\leq i}(t)$ denote $\sum_{j \leq i} V_i(t)$.

Lemma 4.2. *For $i \geq 1$, $\Delta V_{\leq i}(t) \leq (m-1) \cdot 2^{i+2}$.*

Proof. Let $t_i$ be the last time before $t$ when the algorithm executes a job from $G_{>i}$. It is clear that in the interval $(t_i, t]$, the algorithm is continuously working on jobs in $G_{\leq i}$. Further, at $t_i$, the pool is either empty in jobs from $G_{\leq i}$, or they have all just arrived in which case they do not contribute to the volume difference. Hence $\Delta V_{\leq i}(t)$ can be upper bounded by the volume of jobs that are in the stack of the algorithm at $t_i$ in groups $G_{\leq i}$. Up to $(m-1)$ machines could be processing a job in $G_{\leq i}$ and in each of their stacks there could be jobs in groups 1 to $i$. Therefore the total volume in the stack is at most $(m-1) \cdot 2^{i+2}$. □

Let $\ell(t)$ denote $\max_{x \in R(t)} g(x)$, the largest group of which a job is running at $t$. Since there cannot be a job of group less than $\ell$ in the pool, we have the following proposition.

Proposition 4. *$V_{<\ell(t)}(t) = 0$, hence at time $t$, there are no jobs in the algorithm's queue from groups smaller than $\ell(t)$.*

We now bound the weight of the jobs in the pool of the algorithm at time $t$ in terms of the weight of the queue of the optimal at time $t$ and the weight of the jobs of the algorithm that are running at time $t$.

Lemma 4.3. *For all $t$,*

$$\sum_{x \in P(t)} w(x) \leq 4W^*(t) + 12 \sum_{x \in R(t)} w(x).$$

Proof. Let $h(t)$ be the largest group such that a job from $G_h$ is alive at time $t$. If $m(t) < m$ then the pool is empty and the lemma is trivially true. Hence we assume that $m(t) = m$. We use $\ell$ and $h$ instead of $\ell(t)$ and $h(t)$ for ease of notation.

$$\sum_{x \in P(t)} w(x) \quad = \quad \sum_{\ell \leq i \leq h} \sum_{x \in G_i \cap P(t)} w(x).$$

We bound $\sum_{x \in G_i \cap P(t)} w(x)$ by considering $V_i(t)$. The number of jobs in $G_i \cap P(t)$ is upper bounded by $V_i(t)/2^i$ and each job has a weight at most $1/2^i$. Hence we get that $\sum_{x \in P(t)} w(x)$ can be bounded as follows:

$$\begin{aligned}
&\leq \sum_{\ell \leq i \leq h} \frac{V_i(t)}{2^i \cdot 2^i} \\
&= \sum_{\ell \leq i \leq h} \frac{V_i^*(t) + \Delta V_i(t)}{2^i \cdot 2^i} \\
&= \sum_{\ell \leq i \leq h} \frac{V_i^*(t)}{2^i \cdot 2^i} + \sum_{\ell \leq i \leq h} \frac{\Delta V_i(t)}{2^i \cdot 2^i} \\
&\leq \sum_{\ell \leq i \leq h} 4W_i^*(t) + \sum_{\ell \leq i \leq h} \frac{\Delta V_{\leq i}(t) - \Delta V_{\leq i-1}(t)}{2^i \cdot 2^i} \\
&\leq \sum_{\ell \leq i \leq h} 4W_i^*(t) - \frac{\Delta V_{\leq \ell - 1}}{2^\ell \cdot 2^\ell} + \sum_{\ell \leq i < h} \frac{3\Delta V_{\leq i}(t)}{2^{i+1} \cdot 2^{i+1}} + \frac{\Delta V_{\leq h}(t)}{2^h \cdot 2^h} \\
&\leq \sum_{\ell \leq i \leq h} 4W_i^*(t) + \frac{V^*_{\leq \ell-1}(t)}{2^\ell \cdot 2^\ell} + \sum_{\ell \leq i < h} \frac{6(m-1)}{2^{i+1}} + \frac{4(m-1)}{2^h} \\
&\leq \sum_{\ell \leq i \leq h} 4W_i^*(t) + \sum_{1 \leq i \leq \ell - 1} 4W_i^*(t) + 12(m-1)\frac{1}{2^{\ell+1}} \\
&\leq 4W^*(t) + 12(m-1)\frac{1}{2^{\ell+1}}.
\end{aligned}$$

In the above inequalities we upper bounded $-\Delta V_{\leq \ell-1}(t)$ by $V^*_{\leq \ell - 1}(t)$ and $\frac{V^*_{\leq \ell-1}(t)}{2^\ell \cdot 2^\ell}$ by $\sum_{1 \leq i \leq \ell-1} 4W_i^*(t)$.

Every job in $R(t)$ is of group $\ell$ or smaller and $m(t) = m$. Hence $\sum_{x \in R(t)} w(x) \geq m\frac{1}{2^{\ell+1}}$. The lemma follows. □

Theorem 5. *The algorithm* SG *is* 19*-competitive.*

Proof. Putting together Lemmas 4.1 and 4.3 we have the following.

$$\begin{aligned}
\sum_t W(t) &= \sum_t \sum_{x \in S(t)} w(x) + \sum_t \sum_{x \in P(t)} w(x) \\
&\leq 3n + \sum_t (4W^*(t) + 12 \sum_{x \in R(t)} w(x)) \\
&\leq 3n + 4\text{OPT} + 12n \\
&\leq 15n + 4\text{OPT} \\
&\leq 19\text{OPT}.
\end{aligned}$$

□

**Remark on Analysis:** The analysis in both [11] and [4] is more involved than our analysis above. Informally speaking the main reason is as follows. In the proof of Lemma 4.3, we charge the weight of extra jobs in the algorithm's pool at time $t$ to the jobs that are being processed by the algorithm at $t$. This is possible only because our algorithm always runs the jobs from the smallest group (largest weight). In SRPT, and the class based algorithm of [2, 4], the jobs that are being processed at time $t$ are of smaller class than those in the pool. However they could be jobs with much larger processing time (and hence small weight in the stretch measure) but whose remaining processing time has fallen sufficiently low for them to be in a small class. Hence, the weight of the extra jobs in the pool cannot be directly charged to the running jobs at all times. This technical difficulty can be gotten around to get an $O(1)$ ratio, however, it adds to the complexity of the analysis and results in weaker bounds.

## 4.2 Minimizing Stretch with Migration

We modify the SG algorithm in a small way to take advantage of migration. To make the changes minimal we retain the overall structure of the algorithm including the stacks and the pool. The only difference in the behaviour is when a job completes. The algorithm compares the top of the stack with not just the pool but also all the waiting jobs in all of the stacks of the other machines and picks the lowest group job. Ties between jobs in the pool and stacks are broken in favor of those in stacks. Thus jobs can migrate from one stack to another. The analysis is exactly the same except for Lemma 4.2. We claim an improved bound below.

LEMMA 4.4. *For* $i \geq 1$, $\Delta V_{\leq i}(t) \leq (m-1) \cdot 2^{i+1}$.

PROOF. Let $t_i$ be the last time before $t$ when the algorithm executes a job from $G_{>i}$. The reasoning is similar to that of Lemma 4.2, the difference is in estimating the volume of jobs in the stack at $t_i$. Since *migration is allowed* from other stacks, executing a job of group larger than $i$ at $t_i$ implies that the number of jobs in groups $i$ and less is at most $(m-1)$. The maximum volume of $(m-1)$ jobs from $G_{\leq i}$ is bounded by $(m-1) \cdot 2^{i+1}$. □

The rest of the analysis is the same and we obtain the following.

THEOREM 6. *The algorithm* SG 13-*competitive when migration is allowed.*

## 4.3 Improved Bounds

The algorithm as we described assigns two jobs to the same group if they are within a factor of 2 of each other. We can parameterize the algorithm by a real $\alpha > 1$ and assign a job to group $i$ if $p(x) \in [\alpha^i, \alpha^{i+1})$. The analysis can be carried out in a very similar way and we can optimize the constant $\alpha$ to obtain the best ratio. We omit details in this version.

THEOREM 7. *For appropriate choices of* $\alpha$*, the algorithm* SG *is* 9.82-*competitive with migration allowed and is* 17.32-*competitive if migration is not allowed.*

**Flow time:** Finally, the analysis of SG for stretch can extended to show the following result for minimizing total flow time.

THEOREM 8. SG *is* $O(\min\{\log P, \log \frac{n}{m}\})$-*competitive for total flow time.*

Lemmas 4.3 and 4.2 can be adapted without much difficulty to show a bound of $O(\log P)$ for total flow time. Proving the $O(\log \frac{n}{m}\})$ bound is more technically involved. However, the analysis is very similar to that in [2] which is itself based on the analysis of SRPT by Leonardi and Raz [10]. We omit details in this version.

# 5. CONCLUSIONS

We can further improve the ratios in Section 4 by using randomization in the grouping. We choose a number $r$ according to some probability distribution and put all jobs into a group $G_i$ if their procesing times lie in the interval $[r\alpha^i, r\alpha^{i+1})$. Thus we are perturbing the starting point for the geometric partitioning. This is a natural idea that has been used in several contexts earlier. We can optimize $\alpha$ and the distribution for $r$ to obtain improved constants for stretch, we defer the details to the full version.

Many open problems remain regarding weighted flow time. We believe there is a true online algorithm with a polylogarithmic competitive ratio that does not assume knowledge of $P$. It might even be possible to obtain a constant competitive ratio for we do not know any lower bound stronger than 1.618. For the offline problem we have recently obtained a quasi-polynomial time approximation scheme and a PTAS for some restricted cases [5]. These results might perhaps lead to a true PTAS thus resolving the complexity of the problem. The approximability of this problem remains wide open in the multi-processor case. We do not know any non-trivial offline algorithm even for two machines! We believe that the problem is APX-hard if the number of machines is part of the input. Finally, the online algorithm of Leonardi and Raz [10] implies an $O(\min\{\log P, \log \frac{n}{m}\})$-approximation for minimizing unweighted flow time on multiple machines; no better results are known for the offline case. On the other hand, not even APX-hardness is known for this version of the problem. It would be interesting to close the gap on the approximability of unweighted flow time on multiple machines.

# 6. REFERENCES


[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, 1999.

[2] B. Awerbuch, Y. Azar, S. Leonardi, and O. Regev. Minimizing flow time without migration. In *Proceedings of the 31st ACM Symposium on the Theory of Computing*, pages 198–205, 1999.

[3] M. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[4] L. Becchetti, S. Leonardi, and S. Muthukrishnan. Scheduling to minimize average stretch without migration. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 548–57, 2000.

[5] C. Chekuri and S. Khanna. Approximation schemes for preemptive weighted flow time. Manuscript, 2001.

[6] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Offline and online algorithms. *Math. of Operations Research*, 22:513–544, 1997.

[7] B. Kalyanasundaram and K. Pruhs. Speed is equivalent to clairvoyance. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 115–124, 1995.

[8] H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 418–426, May 1996.

[9] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[10] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 110–119, 1997.

[11] S. Muthukrishnan, R. Rajaraman, R. Shaheen, and J. Gehrke. Online scheduling to minimize average stretch. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 433–43, 1999.

[12] C. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the 29th Annual Symposium on Theory of Computing*, 1997.