SERVER SCHEDULING TO BALANCE PRIORITIES, FAIRNESS, AND AVERAGE QUALITY OF SERVICE∗

NIKHIL BANSAL† AND KIRK R. PRUHS‡

Abstract. Often server systems do not implement the best known algorithms for optimizing average Quality of Service (QoS) out of concern that these algorithms may be insufficiently fair to individual jobs. The standard method for balancing average QoS and fairness is to optimize the \( \ell_p \) norm, \( 1 < p < \infty \). Thus we consider server scheduling strategies to optimize the \( \ell_p \) norms of the standard QoS measures, flow and stretch. We first show that there is no \( n^{o(1)} \)-competitive online algorithm for the \( \ell_p \) norms of either flow or stretch. We then show that the standard clairvoyant algorithms for optimizing average QoS, Shortest Job First (SJF), and Shortest Remaining Processing Time (SRPT), are scalable for the \( \ell_p \) norms of flow and stretch. We then show that the standard nonclairvoyant algorithm for optimizing average QoS, Shortest Elapsed Time First (SETF), is also scalable for the \( \ell_p \) norms of flow. We then show that the online algorithm, Highest Density First (HDF), and the nonclairvoyant algorithm, Weighted Shortest Elapsed Time First (WSETF), are scalable for the weighted \( \ell_p \) norms of flow. These results suggest that the concern that these standard algorithms may unnecessarily starve jobs is unfounded. In contrast, we show that the Round Robin, or Processor Sharing, algorithm, which is sometimes adopted because of its seeming fairness properties, is not \( O(1+\epsilon) \)-speed, \( n^{o(1)} \)-competitive for sufficiently small \( \epsilon \).

Key words. scheduling, resource augmentation, flow time, shortest elapsed time first, shortest remaining processing time, multilevel feedback, shortest job first

AMS subject classifications. 68M20, 90B35

DOI. 10.1137/090772228

1. Introduction.

1.1. Motivation. When designing a scheduling strategy for servers in a client-server setting, there is commonly a conflict between the competing demands of fairness and average-case performance. For concreteness let us initially consider a web server serving static content. Each job \( i \) for the web server can be described by a release time \( r_i \) when a request arrives at the server and a size or volume \( p_i \) of the file requested. An online scheduling algorithm must determine the unique job to run or, equivalently, file to transmit, at each point in time. Generally servers, such as web servers, that must handle jobs of widely varying sizes must allow preemption, which is the suspension of the execution of one job and the later resumption of the execution of that job from the point of suspension, to avoid the possibility of one big job delaying many small jobs. For example, standard web servers schedule preemptively. The most commonly used Quality of Service (QoS) measure for a single job \( J_i \) is clearly the flow/response/waiting time \( F_i = C_i - r_i \), where \( C_i \) is the time when the server completes the job. Then the two most obvious QoS objectives for a schedule are the average, or \( \ell_1 \) norm, of the response times, and the maximum, or \( \ell_\infty \) norm, of the response times. It is well known that the scheduling algorithm Shortest Remain-
ing Processing Time (SRPT) is optimal for minimizing the total response time, and
the scheduling algorithm First-Come-First-Served (FCFS) (or, equivalently, First-In-First-Out (FIFO)) is optimal for minimizing the maximum response time. Yet
standard web servers do not use either SRPT or FCFS. For example, the Apache web
server uses FCFS to determine the request that is allocated a free process slot and
essentially leaves the scheduling of the processes to the underlying operating system,
whose scheduling policy may be designed primarily to optimize average-case perfor-
mance [1]. By reverse engineering standard server scheduling policies, such as the
one used by Apache, the apparent goal of the designers was to try to balance the
competing objectives of optimizing the worst case for fairness reasons and optimizing
the average case for performance reasons.

The standard way to compromise between optimizing for the average and optimiz-
ing for the worst case is to optimize the ℓ_p norm, generally for something like
p = 2 or p = 3. For example, the standard way to fit a line to a collection of points is
to pick the line with minimum least squares (equivalently, ℓ_2) distance to the points,
and Knuth’s TEx typesetting system uses the ℓ_3 norm to determine line breaks [20,
page 97]. This suggests optimizing the objective of the ℓ_p norm of the response times,
(∑_{i=1}^{n} F_i^p / n)^{1/p}, as a way to balance the competing demands of balancing worst-case
and average performance. The ℓ_p norm of response times considers the average in
the sense that it takes into account the response times of all jobs, but because
x^p is
strictly a convex function of x, the ℓ_p norm increases more significantly due to jobs
with unusually high response times.

The following passage from the standard Silberschatz and Galvin text Operating Systems Concepts [27] also argues for optimizing essentially the ℓ_2 norm in time
sharing systems as a way of reducing variability:

It is desirable to maximize CPU utilization and throughput, and
to minimize turnaround time, waiting time and response time. In
most cases, we optimize for the average measure. However, there are
circumstances when it is desirable to optimize for the maximum and
minimum values, rather than the average. For example, to guarantee
that all users get good service. It has also been suggested, that
for interactive systems (such as time sharing systems), it is more
important to minimize the variance in the response time than it is to
minimize the average response time. A system with reasonable and
predictable response time may be considered more desirable than a
system that is faster on the average, but is highly variable. However,
little work has been done on CPU scheduling algorithms to minimize variance.

The variance is the expected squared distance from the mean, ∑_{i=1}^{n} (F_i - Z)^2,
where Z = ∑_{i=1}^{n} F_i / n. The objective of minimizing the variance is not a good formal
criterion since one can achieve zero variance by scheduling the jobs so that all the
jobs have the same exact horrible quality of service. So probably the most reasonable
alternative to variance would be to set Z = 0 and consider the objective of ∑_{i=1}^{n} F_i^2,
which is essentially equivalent to the ℓ_2 norm of response times.

Thus our goal here is to report on a theoretical investigation into server scheduling
with the objective of the ℓ_p norms of standard job QoS measures. To give an indication
of the type of results that we obtain, let us return to our discussions of scheduling
policies for web servers serving static content with the objective being the ℓ_2 norm of
response times.
We first give a negative result that shows that optimizing for the $\ell_2$ norm of response times is much harder than optimizing for the $\ell_1$ and $\ell_\infty$ norms. More precisely, we show that every scheduling algorithm has competitive ratio $\omega(1)$. The competitive ratio of an algorithm $A$ and an objective $F$ is

$$\max_I \frac{F(A(I))}{F(Opt(I))} \leq c,$$

where $A(I)$ denotes the schedule that algorithm $A$ produces on input $I$, and similarly $Opt(I)$ denotes the optimal schedule for $I$. So for every scheduling algorithm $A$, there are instances $I$ on which $A$ produces a schedule where the $\ell_2$ norm of response times can be unboundedly worse than in the optimal schedule.

However, as is often the case in such lower bound constructions, the lower bound instances have a special adversarial structure. The load that these instances place on the server is near the capacity of the server, so that the scheduling algorithm does not have any spare time to recover from even minor scheduling misjudgments. We attempt to illustrate this phenomenon in Figure 1. QoS curves such as those in Figure 1(a) are ubiquitous in server systems. That is, there is a relatively modest degradation in schedule QoS as the load increases until one nears some threshold—this threshold is essentially the capacity of the system—after which any increase in the load precipitously degrades the QoS provided by the server. The concept of load is not so easy to formally define but generally reflects the number of users of the system. The online algorithm whose performance is pictured in Figure 1(b), while not optimal, would seem to have reasonable performance. However, this online algorithm would have a high competitive ratio. The value of the competitive ratio is at least the ratio of the values of the performance curves for online and optimal at a particular load. So in Figure 1(b) one can establish that the competitive ratio must be high by picking a load a bit less than the capacity of the system, where the performance of the online algorithm has degraded significantly, but the performance of the optimal algorithm has not yet degraded.

To address this issue, [18] introduced resource augmentation analysis, which compares the online algorithm against an optimal offline algorithm with a slower processor. More formally, in the context of a scheduling minimization problem with an objective function $F$, an algorithm $A$ is $s$-speed, $c$-competitive if

$$\max_I \frac{F(A_s(I))}{F(Opt(I))} \leq c,$$
where the subscripts denote the speed of the processor that the algorithms use. Increasing the speed of a server by a factor of \( s \) is essentially equivalent to lowering the load on the server by a factor of \( s \). A \((1 + \epsilon)\)-speed, \(O(1)\)-competitive algorithm is said to be scalable \([25, 24]\). More formally, an algorithm \( A \) is scalable if for all \( \epsilon > 0 \), there exists a constant \( c \) (which may depend upon \( \epsilon \)) such that for all inputs \( I \),

\[
\frac{F(A_{1+\epsilon}(I))}{F(\text{Opt}_I)} \leq c.
\]

Such a scalable algorithm is \(O(1)\)-competitive on inputs \( I \) where \( \text{Opt}_I(I) \) is approximately \( \text{Opt}_{1+\epsilon}(I) \). The loads in the usual performance curve shown in Figure 1(a), where \( \text{Opt}_I(I) \) is not approximately \( \text{Opt}_{1+\epsilon}(I) \), are those points near or above the capacity of the system. Thus the performance curve of a scalable scheduling algorithm should be no worse than that shown in Figure 1(b); that is, the scheduling algorithm should scale reasonably well up to quite near the capacity of the system.

Among other results, we will show that the scheduling algorithm \( \text{SRPT} \) is a scalable algorithm for the objective of the \( \ell_p \) norm of response times. This result suggests that perhaps the fear that \( \text{SRPT} \) will unnecessarily starve jobs is unfounded since \( \text{SRPT} \) scales almost as well with load as the optimal algorithm for the \( \ell_p \) norm of response times.

1.2. Our results. We analyze the performance of various standard scheduling algorithms for the objectives of the \( \ell_p \) norms, \( 1 < p < \infty \), of the two most common job QoS measures, response time and stretch. The stretch or slowdown for a job \( i \) is defined to be \( \frac{F_i}{p_i} \). If a job has stretch \( s \), then it appears to the client that it received dedicated service from a speed \( 1/s \) processor. One motivation for considering stretch is that a human user may have some feeling for the size of a job. For example, in the setting of a web server, the user may have some knowledge about the size of the requested document (for example, the user may know that video documents are generally larger than text documents) and may be willing to tolerate a larger response time for larger documents.

We consider the following algorithms:

- **SJF.** Shortest Job First always runs the job \( i \) of minimum volume \( p_i \). SJF may preempt a job when a job of smaller volume arrives.
- **SRPT.** Shortest Remaining Processing Time always runs the job of minimum unfinished volume. So, for example, if a job \( i \) is two-thirds completed, then its unfinished volume is \( p_i/3 \).
- **RR.** Round Robin shares the processor equally among all jobs.
- **SETF.** Shortest Execution Time First always runs the job that has been run the least so far.
- **HDF.** Highest Density First always runs the job of highest density, which is the weight of a job divided by the size of a job.
- **WSETF.** Among all jobs with the smallest norm, Weighted Shortest Execution Time First splits the processor proportionally to the weights of the jobs. The norm of a job is its finished volume divided by its weight.

All the algorithms except for \( \text{WSETF} \), which we introduce here, are standard scheduling algorithms.

We summarize our results in Table 1. We first consider the standard online algorithms aimed at average QoS, that is, \( \text{SJF} \) and \( \text{SRPT} \). We show that the algorithms \( \text{SJF} \) and \( \text{SRPT} \) are scalable with respect to all \( \ell_p \) norms of flow and stretch. Next we consider the class of algorithms that do not require a priori knowledge of the service
Table 1
Resource augmentation results for unweighted and weighted \( \ell_p \) norms, \( 1 < p < \infty \).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Speed</th>
<th>( \ell_p ) Norm of flow</th>
<th>( \ell_p ) Norm of stretch</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>General clairvoyant algorithm</td>
<td>1</td>
<td>( n^{O(1)} )</td>
<td>( n^{\Omega(1)} )</td>
<td>4</td>
</tr>
<tr>
<td>SJF</td>
<td>((1 + \epsilon))</td>
<td>( O(1/\epsilon) )</td>
<td>( O(1/\epsilon) )</td>
<td>5</td>
</tr>
<tr>
<td>SRPT</td>
<td>((1 + \epsilon))</td>
<td>( O(1/\epsilon) )</td>
<td>( O(1/\epsilon) )</td>
<td>6</td>
</tr>
<tr>
<td>SETF</td>
<td>((1 + \epsilon))</td>
<td>( O(1/\epsilon^{2 + 2/p}) )</td>
<td>( O(1/\epsilon^{2 + 2/p}) )</td>
<td>7</td>
</tr>
<tr>
<td>RR</td>
<td>((1 + \epsilon))</td>
<td>( \Omega(n^{1-2/p}) )</td>
<td>( \Omega(n) )</td>
<td>8</td>
</tr>
<tr>
<td>General nonclairvoyant algorithm</td>
<td>((1 + \epsilon))</td>
<td>( \Omega(\min{n, \log B}) )</td>
<td>( \Omega(\min{n, \log B}) )</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Speed</th>
<th>( \ell_p ) Norm of flow</th>
<th>( \ell_p ) Norm of stretch</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDF</td>
<td>((1 + \epsilon))</td>
<td>( O(1/\epsilon^2) )</td>
<td>( O(1/\epsilon^2) )</td>
<td>9</td>
</tr>
<tr>
<td>WSETF</td>
<td>((1 + \epsilon))</td>
<td>( O(1/\epsilon^{2 + 2/p}) )</td>
<td>( O(1/\epsilon^{2 + 2/p}) )</td>
<td>10</td>
</tr>
</tbody>
</table>

requirement of a job, which are often referred to as nonclairvoyant algorithms [22]. We show that the standard nonclairvoyant algorithm SETF is also scalable for all \( \ell_p \) norms of flow. For the \( \ell_p \) norm of stretch, we show that, given \((1 + \epsilon)\) speed-up, SETF is polylogarithmically competitive in \( B \), the ratio of the length of the longest job to the shortest job. We also give an essentially matching lower bound. Note that all of the results assume that \( p \) is constant, so that multiplicative factors, which are a function of \( p \) alone, are absorbed into the constant in the asymptotic notation. Recall that we are primarily interested in \( p = 2 \) and \( p = 3 \).

Some server systems have mechanisms that allow the users or the system to specify that some jobs are more important than other jobs. For example, in the Unix operating system the jobs have priorities which can be set with the nice command. One natural way to formalize a scheduling objective in a setting where the jobs are of varying importance is to associate a weight \( w_i \) with each job \( i \), and then to weight the performance of each job accordingly in the objective. So, for example, the weighted \( \ell_p \) norm of flow would be \( \left( \sum_{i=1}^{n} w_i F_i^p \right)^{1/p} \). We show that both HDF and the nonclairvoyant algorithm WSETF are scalable for the \( \ell_p \) norms of flow. An interesting aspect of our analysis of HDF and WSETF is that we first transform the problem on the weighted instance to a related problem on the unweighted instance.

These resource augmentation results argue that the concern that these standard scheduling algorithms aimed at optimizing average QoS might unnecessarily starve jobs is unfounded when the server is less than fully loaded.

It might be tempting to conclude that all reasonable algorithms should have such scaling guarantees. However, we show that this is not the case. More precisely, the standard Processor Sharing, or, equivalently, Round Robin, algorithm is not \((1 + \epsilon)\)-speed, \( O(1) \)-competitive for any \( \ell_p \) norm of flow, \( 1 < p < \infty \), and for sufficiently small \( \epsilon \). This is perhaps surprising, since fairness is a commonly cited reason for adopting Processor Sharing [28].

2. Related results. The results in the literature that are closest in spirit to those here are found in a series of papers, including [4, 14, 16, 26], that argue that SRPT will not unnecessarily starve jobs any more than Processor Sharing, or, equiv-
ally, $RR$, does under “normal” situations. In these papers, “normal” is defined as there being a Poisson distribution on release times, and processing times being independent samples from a heavily tailed distribution. More precisely, these papers argue that every job should prefer $SRPT$ to Processor Sharing under these circumstances. Experimental results supporting this hypothesis are also given. So informally our paper and these papers reach the same conclusion about the superiority of $SRPT$. But in a formal sense the results are incomparable.

The following results are known about online algorithms when the objective function is average flow time. The competitive ratio of every deterministic nonclairvoyant algorithm is $O(n^{1/3})$, and the competitive ratio of every randomized nonclairvoyant algorithm against an oblivious adversary is $O(\log n)$ [22]. Here $n$ is the number of jobs in the instance. The randomized nonclairvoyant algorithm Randomized Multi-Level Feedback (RMLF), proposed in [19], is $O(\log n)$-competitive against an oblivious adversary [5]. The online clairvoyant algorithm $SRPT$ is optimal. The online clairvoyant algorithm $SJF$ is scalable [6]. The nonclairvoyant algorithm $SETF$ is scalable [18]. $RR$ is also known to be $O(1)$-speed, $O(1)$-competitive for average flow time [15].

For minimizing average stretch, [23] shows that $SRPT$ is 2-competitive. In the offline case, there is a polynomial-time approximation scheme for average stretch [7, 10]. In the nonclairvoyant case, the authors of [3] showed that any algorithm is $\Omega(B)$-competitive (without speed-up), where $B$ is the ratio of the maximum to minimum size. They also showed that even with an $O(1)$ times speed-up, any algorithm is $\Omega(\min(n, \log B))$-competitive. They also gave a nonclairvoyant algorithm that is $(1 + \epsilon)$-speed, $O(\text{poly}(\epsilon^{-1} \cdot \log B))$-competitive.

$HDF$ was shown in [6] to be $(1 + \epsilon)$-speed, $O(1/\epsilon)$-competitive for weighted flow time. This immediately implies a similar result for $SJF$ for average flow time and for average stretch. Reference [2] gives an $O(\log W)$-competitive algorithm for weighted flow time. Other results for weighted flow time can be found in [11, 10, 2].

Subsequent to this research, two papers [9, 12] extend our results to the multiprocessor setting. The paper [9] shows that $HDF$ is scalable in the multiprocessor setting. The paper [12] shows that scalable nonclairvoyant randomized algorithms exist that have the immediate dispatch property; that is, they assign a job to a processor as soon as the job arrives.

3. Definitions and preliminaries. We assume a collection of jobs $J = J_1, \ldots, J_n$. For $J_i$, the release time is denoted by $r_i$, the volume/size by $p_i$, and the weight by $w_i$. The density of a job $J_i$ is $w_i/p_i$. Our analyses will require the technical restriction that the weights are positive integers.

A schedule specifies for each time a job run at that time. If a speed $s$ processor works on a job $J_i$ for an amount of time $t$, then the unfinished volume of $J_i$ is reduced by $s \cdot t$. Note that we allow preemption; that is, a job can be interrupted arbitrarily in time and resumed later from the point of interruption. We use $p_i(t)$ to denote the remaining/unfinished volume on job $J_i$ at time $t$, and $x_i(t) = p_i - p_i(t)$ to denote the work performed on $J_i$ by time $t$. The completion time $C_i(S)$ of a job $J_i$ in a schedule $S$ is the first time $t$ after $r_i$ when $p_i(t) = 0$. The flow time of $J_i$ in $S$ is $F_i(S) = C_i(S) - r_i$. The stretch $S_i$ of $J_i$ in $S$ is $F_i(S)/p_i$. If the schedule $S$ is understood, it may be dropped from the notation. An online algorithm does not know about job $J_i$ until time $r_i$, at which time it learns the weight of $J_i$. A clairvoyant algorithm $A$ learns $p_i$ at time $r_i$. A nonclairvoyant algorithm $A$ learns about the size of a job only when it meets its service requirement and leaves the system. In particular, at any time $t$ the algorithm only knows a lower bound on $p_i$ equal to $x_i(t)$.
We use $A_s(\mathcal{I})$ to denote the schedule output by the algorithm $A$, with a speed $s$ processor, on the input $\mathcal{I}$. For an objective function $X$, we use $X(A_s(\mathcal{I}))$ to denote the value of the objective function $X$ on the schedule $A_s(\mathcal{I})$. On some occasions, particularly when the speed $s$ is typographically complex, it is more convenient to use the notation $X(A, s)$ instead. Some objective functions that we consider are $F = \sum_i F_i$, $F^p = \sum_i F_i^p$, $S = \sum_i S_i$, $S^p = \sum_i S_i^p$, $WF = \sum_i w_i F_i$, and $WF^p = \sum_i w_i F_i^p$. The $\ell_p$ norm of flow is $(\sum_i F_i^p)^{1/p}$, the $\ell_p$ norm of stretch is $(\sum_i S_i^p)^{1/p}$, and the $\ell_p$ norm of weighted flow is $(\sum_i w_i F_i^p)^{1/p}$. Thus an algorithm being $c$-competitive for objective $F^p$ is equivalent to being $c^{1/p}$-competitive for the $\ell_p$ norm of flow, and a similar relationship holds for stretch and weighted flow. We use $\text{Opt}$ to denote the optimal schedule for the objective function under consideration. So $F^p(\text{Opt}_s(\mathcal{I}))$ denotes the value of the $F^p$ objective function on the optimal schedule for this objective function. Although when the objective is understood from the context, we will drop it from the notation.

A job $J_i$ is unfinished in a schedule $S$ at time $t$ if $r_i \leq t \leq C_i(S)$, and it has age $t - r_i$ at this time. We use $U(S, t)$ to denote the collection of unfinished jobs at time $t$ in the schedule $S$. For a schedule $S$, we use $\text{Age}^p(S, t)$ to denote the sum over all jobs $J_i \in U(S, t)$ of $(t - r_i)^{p-1}$. For a schedule $S$, we use $S\text{Age}^p(S, t)$ to denote the sum over all jobs $J_i \in U(S, t)$ of $(t - r_i)^{p-1}/p_i^p$. Note that $F^p(S) = p \int_{r_i}^t \text{Age}^p(S, t) dt$. That is, if at every time step $t$, each unfinished job pays an amount equal to its age to the $(p - 1)$st power, then over the lifetime of any job, the total amount paid by that job is exactly equal to $\frac{1}{p} F^p$. Thus, the total payment made by all the jobs is proportional to $F^p$. Thus to show that an algorithm $A_{1+\epsilon}$ is $O(\frac{1}{\epsilon^p})$-competitive for the $\ell_p$ norm of flow, it is sufficient to show that at any time $t$ the following local competitiveness condition holds:

$$\text{(1)} \quad \text{Age}^p(U(A_{1+\epsilon}, t), t) = O\left(\frac{1}{\epsilon^p}\right) \cdot \text{Age}^p(U(\text{Opt}_1, t), t).$$

Similarly, to show that an algorithm $A_{1+\epsilon}$ is $O(\frac{1}{\epsilon^p})$-competitive for the $\ell_p$ norm of stretch, it is sufficient to show that at any time $t$ the following local competitiveness condition holds:

$$\text{(2)} \quad S\text{Age}^p(U(A_{1+\epsilon}, t), t) = O\left(\frac{1}{\epsilon^p}\right) \cdot S\text{Age}^p(U(\text{Opt}_1, t), t).$$

Let $V(u, S, \alpha)$ denote the unfinished volume at time $u$ in the schedule $S$ of those jobs $J_j$ that satisfy the conditions in the list $\alpha$. So, for example, $V(r_i, \text{Opt}_1, r_j \leq r_i, p_j \leq p_i)$ is the amount of volume that $\text{Opt}_1$ has unfinished on jobs $J_j$ that arrived not after $r_i$, are not larger in size than $J_i$, and that $\text{Opt}_1$ has not finished by time $r_i$. Similarly, we let $P(u, S, \alpha)$ denote the aggregate sizes of the unfinished jobs $J_j$ at time $u$ in the schedule $S$ that satisfy the conditions in the list $\alpha$. Finally let $Q(\alpha)$ denote the aggregate size of the jobs $J_j$ that satisfy the conditions in the list $\alpha$. Note that for simplicity of notation, we always implicitly use $j$ as the local index to the jobs being described in this notation.

An oblivious adversary must specify the complete input before an online algorithm begins executions [8].

4. General lower bounds. In this section we show that there are no online algorithms that are $O(1)$-competitive with respect to the $\ell_p$ norms of flow and stretch for $1 < p < \infty$. 

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.
Theorem 1. Let $p > 1$. The competitive ratio, with respect to the $\ell_p$ norm of flow time, of any randomized algorithm $A$ against an oblivious adversary is $\Omega(n^{(p-1)/p(3p-1)})$. The competitive ratio, with respect to the $\ell_p$ norm of stretch, of any randomized algorithm $A$ against an oblivious adversary is $\Omega(n^{(p-1)/3p^2})$.

Proof. We use Yao’s minimax principle for online cost minimization problems [8] and lower bound the expected value of the ratio of the objective functions on $A$ and Opt on input distribution which we specify. The inputs are parameterized by integers $L$, $\alpha$, and $\beta$ in the following manner. A long job of size $L$ arrives at $t = 0$. From time 0 until time $L^\alpha - 1$ a job of size 1 arrives every unit of time. With probability 1/2 this is all of the input. With probability 1/2, $L^{\alpha+\beta}$ short jobs of length $1/L^\beta$ arrive every $1/L^\beta$ time units from time $L^\alpha$ until $2L^\alpha - 1/L^\beta$.

We first consider the $F^p$ objective. In this case, $\alpha = \frac{p+1}{p-1}$, and $\beta = 2$. We now compute $F^p(A)$ and $F^p(Opt)$. First consider the case that $A$ does not finish the long job by time $L^\alpha$. Then with probability 1/2 the input contains no short jobs. Then $F^p(A)$ is at least the flow of the long job, which is at least $L^{\alpha p}$. In this case the adversary could first process the long job and then process the unit jobs. Hence, $F^p(Opt) = O(L^p + L^\alpha \cdot L^p) = O(L^{\alpha p + p})$. The competitive ratio is then $\Omega(L^p - \alpha - p)$, which is $\Omega(L)$ by our choice of $\alpha$.

Now consider the case that $A$ finishes the long job by time $L^\alpha$. Then with probability 1/2 the input contains short jobs. One strategy for the adversary is to finish all jobs, except for the long job, when they are released. Then $F^p(A)$ is at least the flow of the long job, which is at least $L^{\alpha p}$. In this case the adversary could first process the long job and then process the unit jobs. Hence, $F^p(Opt) = O(L^p + L^\alpha \cdot L^p) = O(L^{\alpha p + p})$. The competitive ratio is $\Omega(L^{\alpha p - \alpha - p})$, which is $\Omega(L)$ by our choice of $\alpha$.

We now consider the $S^p$ objective. In this case, $\alpha = \frac{p+1}{p-1}$, and $\beta = 1$. We now compute $S^p(A)$ and $S^p(Opt)$. First consider the case that $A$ does not finish the long job by time $L^\alpha$. Then with probability 1/2 the input contains no short jobs. Then $S^p(A)$ is at least the stretch of the long job, which is at least $(L^\alpha /L)^p = L^{p(\alpha - 1)}$. In this case the adversary could first process the long job and then process the unit jobs. Hence, $S^p(Opt) = O(1 + L^\alpha \cdot L^p) = O(L^{\alpha p + p})$. The competitive ratio is $\Omega(L^{\alpha p - \alpha - 2p})$, which is $\Omega(L)$ by our choice of $\alpha$.

Now consider the case that $A$ finishes the long job by time $L^\alpha$. Then with probability 1/2 the input contains short jobs. One strategy for the adversary is to finish all jobs, except for the long job, when they are released. Then $S^p(A)$ is at least the stretch of the long job, which is at least $(L^\alpha /L)^p = L^{p(\alpha - 1)}$. Algebraic simplification shows that $\alpha + \beta \leq \alpha p - p$ for our choice of $\alpha$ and $\beta$. Hence the dominant term is $L^{p(\alpha - p)}$, and $S^p(Opt) = O(L^{p(\alpha - p)})$. Now consider the subcase that $A$ has at least $L/2$ unit jobs unfinished at time $3L^\alpha/2$. Since these unfinished unit jobs must have been delayed by at least $L/2$,
$S^p(A) = \Omega(L \cdot L^{op})$. Clearly in this subcase the competitive ratio is $\Omega(L^{p+1})$. Alternatively, consider the subcase that $A$ has at most $L/2$ unit jobs unfinished by time $3L^\alpha/2$. Then $A$ has at least $L/2$ unfinished volume in small jobs at time $3L^\alpha/2$. By the convexity of $S^p$ when restricted to jobs of size $1/L^\beta$, the optimal strategy for $A$ from time $3L^\alpha/2$ onward always delays each small job by the same amount (we can gift $A$ the competition of the unit jobs at time $3L^\alpha/2$). Thus $A$ delays $L^{\alpha+\beta}/2$ short jobs by at least $L/2$. Hence in this case, $S^p(A) = \Omega(L^{\alpha+\beta+L^{(\beta+1)p}})$. This gives a competitive ratio of $S^p(A) = \Omega(L^{\alpha+\beta+L^{(\beta+1)p}})$. By the choice of $\alpha$ and $\beta$, this gives a competitive ratio of $\Omega(L^{2p+1})$.

Thus the overall competitive ratio is at least $\Omega(L)$, and noting that $n = O(L^{\alpha+\beta})$, the desired result follows. \[ \square \]

It is easy to see that there is no $O(1)$-speed, $O(1)$-competitive nonclairvoyant algorithm for $S^p$, $1 < p < \infty$, using the instance from [17]. In this instance, there are $n$ jobs with sizes $1, 2, 4, 8, \ldots, B = 2^n$ all of which arrive at time 0. By scheduling the jobs from shortest to longest, it is easy to see that each job has a stretch of at most 2, and hence $S^p(\text{Opt}) = O(n)$. For any nonclairvoyant (possibly randomized) algorithm $A$ with a speed $s \geq 1$ processor, it is not hard to see that the job of size $2^i$ incurs a stretch of at least $\Omega(n-i)/s$, as it is indistinguishable from jobs of size greater than $2^i$ until they receive a service of $2^i$; a formal argument can be found in [3]. Thus, for any nonclairvoyant algorithm $A$, $S^p(A, s) = \Omega(n^{p-1}/s^p)$, which implies a lower bound of $\Omega(n)$ on the competitive ratio of any nonclairvoyant algorithm even with an $O(1)$ speed for all $\ell_p$ norms of stretch. Note that in the above instance $B = O(2^n)$, and hence this also implies a lower bound of $\Omega(\log B)$.

5. Analysis of SJF. This section is devoted to proving the following theorem.

**Theorem 2.** SJF is $(1 + \epsilon)$-speed, $O(\frac{1}{\epsilon})$-competitive for the $\ell_p$ norms of flow time and stretch for $p \geq 1$.

Recall that SJF always runs a job of minimal size. We will fix an input, some 1-speed processor schedule $\text{Opt}_1$ for this instance, and a time $t$, and then argue that SJF satisfies the local competitiveness conditions (1) and (2) at this time $t$.

Let $D$ denote $U(\text{SJF}_{1, t}, t) \setminus U(\text{Opt}_1, t)$. An association scheme maps each $J_i \in D$ to an $ep_i/4(1 + \epsilon)$ amount of volume from $V(t, \text{Opt}_1, r_j \leq t - \frac{\epsilon}{4(1 + \epsilon)}(t - r_i), p_j \leq p_i)$ so that no amount of volume is mapped to by more than one job in $D$. We show in Lemma 3 that the existence of an association scheme implies that the local competitiveness conditions hold. Lemma 4 proves an invariant that we then use in Lemma 5 to show that an association scheme exists.

**Lemma 3.** If an association scheme exists, then the local competitiveness conditions (1) and (2) hold at time $t$.

**Proof.** Consider an arbitrary job $J_i \in D$. Suppose the association scheme maps $J_i$ to $v_{\sigma(j)}$ volume from job $J_{\sigma(j)}$ in $\text{Opt}_1$ for $j = 1, 2, \ldots, \lfloor U(\text{Opt}_1, t) \rfloor$. The properties of the association scheme guarantee that $\sum_j v_{\sigma(j)} \geq \epsilon p_i/(4(1 + \epsilon))$. Moreover, each job $J_{\sigma(j)}$ with $v_{\sigma(j)} > 0$ has size $p_{\sigma(j)}$ at most $p_i$ and age at least $\epsilon/(4(1 + \epsilon))$ times the age of $J_i$. For each $j = 1, \ldots, \lfloor U(\text{Opt}_1, t) \rfloor$, we can view $J_i$ as being associated with a $v_{\sigma(j)}/p_{\sigma(j)}$ fraction of $J_{\sigma(j)}$. Since $\sum_j v_{\sigma(j)} \geq \epsilon p_i/(4(1 + \epsilon))$ and $p_{\sigma(j)} \leq p_i$ whenever $v_{\sigma(j)} > 0$, this implies that $J_i$ is associated with at least a

$$\sum_{j: v_{\sigma(j)}>0} \frac{v_{\sigma(j)}}{p_{\sigma(j)}} \geq \sum_{j: v_{\sigma(j)}>0} \frac{v_{\sigma(j)}}{p_i} \geq \frac{ep_i}{4(1 + \epsilon)} \cdot \frac{1}{p_i} = \frac{\epsilon}{4(1 + \epsilon)}$$

fraction of jobs in $\text{Opt}_1$. As the age of any job $J_{\sigma(j)}$ with $v_{\sigma(j)} > 0$ is least $\frac{\epsilon}{4(1 + \epsilon)}$.
times the age of $J_j$, the value of $\text{Age}^p$ for $J_{\sigma(j)}$ is $\Omega(e^{p-1})$ times the value of $\text{Age}^p$ of $J_j$. The local competitiveness condition for flow (1) thus follows together with (3) and summing up over all jobs in $D$.

The argument for stretch is very similar. Instead of $\text{Age}^p$ above, we consider $\text{SAge}^p$. The key observation is that the association scheme associates job $J_i \in D$ with a volume of jobs of no larger size, and age at least $\Omega(\epsilon)$ times that of $J_i$. Hence $\text{SAge}^p$ of each of these jobs is $\Omega(e^{p-1})$ times the value of $\text{Age}^p$ of $J_i$.

**Lemma 4.** Given any arbitrary scheduling instance $J$, let $\text{Opt}_1$ denote any schedule using a speed 1 processor. Then for all times $u$ and for all values $p$, the following inequality holds:

$$V(u, \text{Opt}_1, p_j \leq p) \geq \frac{\epsilon}{1+\epsilon} P(u, \text{SJF}_{1+\epsilon}, p_j \leq p) + \frac{1}{1+\epsilon} V(u, \text{SJF}_{1+\epsilon}, p_j \leq p).$$

**Proof.** The proof is by induction on the events that might happen at time $u$.

First, suppose that $\text{SJF}_{1+\epsilon}$ is working on some unfinished job $J_j$ with $p_j \leq p$. Then, the right side decreases at least as fast as the left side since $\text{SJF}_{1+\epsilon}$ has a $(1+\epsilon)$-speed processor, and the left side can decrease at a rate of at most 1. Moreover, in the event that job $J_j$ finishes under $\text{SJF}_{1+\epsilon}$, this only helps the inequality (as $J_j$ ceases to contribute to the first term on the right side). On the other hand, if there is no job $J_j$ with size $\leq p$ under $\text{SJF}_{1+\epsilon}$, then the right side is zero and the inequality is trivially satisfied.

In the case that $\text{Opt}_1$ finishes a job, this does not affect the inequality as the left side varies continuously with time. The only remaining event is that a new job $J_j$, with $p_j \leq p$, arrives at time $u$. In this case, both sides increase by exactly $p_j$, which implies the desired result.

**Lemma 5.** An association scheme exists.

**Proof.** We give a procedure for constructing an association scheme and prove its correctness. Let $1, \ldots, k$ denote the indices of jobs in $D$ such that $p_1 \leq p_2 \leq \cdots \leq p_k$, and consider them in this order. Assume that we are considering job $J_j$. Let $t'$ denote the time $t - \frac{\epsilon p_j}{2(1+\epsilon)} (t - r_i)$. Associate with $J_j$ an $\epsilon p_j/4(1+\epsilon)$ amount of volume taken arbitrarily from $V(t, \text{Opt}_1, r_j \leq t', p_j \leq p_j)$, provided that it has not been previously associated with a lower indexed job in $D$.

Consider some fixed job $J_j \in D$. To show that this procedure does not fail on job $J_j$, it clearly suffices to show that $V(t, \text{Opt}_1, p_j \leq p_i, r_j \leq t')$ contains enough volume to associate all the jobs $J_1, \ldots, J_i$; that is,

$$V(t, \text{Opt}_1, p_j \leq p_i, r_j \leq t') \geq \frac{\epsilon}{4(1+\epsilon)} Q(J_j \in D, p_j \leq p_i).$$

We first upper bound the right side. Clearly, $Q(J_j \in D, p_j \leq p_i)$ can be written as

$$Q(J_j \in D, r_j \leq r_i, p_j \leq p_j) + Q(J_j \in D, r_j > r_i, p_j \leq p_j).$$

Since $\text{Opt}_1$ has to finish all jobs in $D$ that are released after $r_i$, by time $t$, the latter term can be upper bounded as

$$t - r_i \geq Q(J_j \in D, r_j > r_i, p_j \leq p_i).$$

To upper bound the first term, we note that any job unfinished at time $t$ under $\text{SJF}_{1+\epsilon}$ that arrived before time $r_i$ must also be unfinished at time $r_i$. Thus,

$$P(r_i, \text{SJF}_{1+\epsilon}, p_j \leq p_i) \geq Q(J_j \in D, r_j \leq r_i, p_j \leq p_i).$$
Combining (5) and (6), we have
\begin{equation}
Q(J_i \in D, p_j \leq p_i) \leq P(r_i, SJF_{1+\epsilon}, p_j \leq p_i) + (t - r_i).
\end{equation}

We now lower bound the left side of (4). Since Opt_1 can do at most \((t - r_i)\) work during \([r_i, t]\), the unfinished volume at time \(t\), composed of jobs that arrived by \(t'\), must be at least unfinished volume at \(r_i\), plus new volume that arrived during \([r_i, t']\) minus \((t - r_i)\). Applying this argument for jobs of size at most \(p_i\), we get
\begin{equation}
V(t, \text{Opt}_1, p_j \leq p_i, r_j \leq t') \geq V(r_i, \text{Opt}_1, p_j \leq p_i) + Q(r_j \in (r_i, t'], p_j \leq p_i) - (t - r_i).
\end{equation}

By Lemma 4 with \(u = r_i\) and \(p = p_i\), the first term on the right side of (8) can be bounded as
\begin{equation}
V(r_i, \text{Opt}_1, p_j \leq p_i) \geq \frac{\epsilon}{1 + \epsilon} P(r_i, SJF_{1+\epsilon}, p_j \leq p_i) + \frac{1}{1 + \epsilon} V(r_i, SJF_{1+\epsilon}, p_j \leq p_i).
\end{equation}

Combining (8) and (9) we get that
\begin{equation}
V(t, \text{Opt}_1, p_j \leq p_i, r_j \leq t') \geq \frac{\epsilon}{1 + \epsilon} P(r_i, SJF_{1+\epsilon}, p_j \leq p_i)
+ \frac{1}{1 + \epsilon} V(r_i, SJF_{1+\epsilon}, p_j \leq p_i)
+ Q(r_j \in (r_i, t'], p_j \leq p_i) - (t - r_i)
\geq \frac{\epsilon}{4(1 + \epsilon)} P(r_i, SJF_{1+\epsilon}, p_j \leq p_i)
+ \frac{4 + 3\epsilon}{4(1 + \epsilon)} V(r_i, SJF_{1+\epsilon}, p_j \leq p_i)
+ Q(r_j \in (r_i, t'], p_j \leq p_i) - (t - r_i),
\end{equation}

where the second inequality follows by noting that
\begin{equation}
P(r_i, SJF_{1+\epsilon}, p_j \leq p_i) \geq V(r_i, SJF_{1+\epsilon}, p_j \leq p_i).
\end{equation}

Getting back to proving (4), it suffices to show that the right side of (10) exceeds \(\frac{\epsilon}{4(1 + \epsilon)}\) times the right side of (7). By canceling the common terms, it suffices to show that
\begin{equation}
\frac{4 + 3\epsilon}{4(1 + \epsilon)} V(r_i, SJF_{1+\epsilon}, p_j \leq p_i) + Q(r_j \in (r_i, t'], p_j \leq p_i) \geq \frac{4 + 5\epsilon}{4(1 + \epsilon)} (t - r_i).
\end{equation}

We now relate \((t - r_i)\) to the terms on the left side of this expression. As \(J_i\) is unfinished under \(SJF_{1+\epsilon}\) at time \(t\), it must be the case that \(SJF_{1+\epsilon}\) was working on jobs of size \(\leq p_i\) during the entire time \([r_i, t]\). These jobs of size \(\leq p_i\) that \(SJF_{1+\epsilon}\) worked on during \([r_i, t']\) either had to arrive before \(r_i\) or during \([r_i, t']\), and it must be the case that
\begin{equation}
V(r_i, SJF_{1+\epsilon}, p_j \leq p_i) + Q(r_j \in (r_i, t'], p_j \leq p_i) \geq (1 + \epsilon)(t' - r_i) = \frac{4 + 3\epsilon}{4} (t - r_i).
\end{equation}

The equality above follows by our choice of \(t' = t - \epsilon(t - r_i)/(4(1 + \epsilon))\). Multiplying (12) by \(\frac{(4 + 5\epsilon)}{(1 + \epsilon)(4 + 3\epsilon)}\), we obtain that
\begin{equation}
\frac{4 + 5\epsilon}{(1 + \epsilon)(4 + 3\epsilon)} (V(r_i, SJF_{1+\epsilon}, p_j \leq p_i) + Q(r_j \in (r_i, t'], p_j \leq p_i)) \geq \frac{4 + 5\epsilon}{4(1 + \epsilon)} (t - r_i).
\end{equation}
Now, (11) follows by observing that the right side of (13) is same as that of (11), but the coefficient of each term of the left side of (11) is larger than the corresponding coefficient in the left side of (13).

6. Analysis of SRPT. This section is devoted to proving the following theorem.

Theorem 6. SRPT is $(1 + \epsilon)$-speed, $O(\frac{1}{\epsilon})$-competitive for the $\ell_p$ norms of flow time and stretch for $p \geq 1$.

Recall that SRPT always runs a job with the least remaining unfinished volume. We will fix an input, some 1-speed processor schedule $Opt_1$, and a time $t$, and then argue that SRPT satisfies the local competitiveness conditions (1) and (2) at this time $t$. The overall structure of the analysis of SRPT is similar to that of SJF, but the analysis is somewhat more involved because we need to handle the remaining volume under SRPT more carefully.

We define $D$ be the collection of jobs $J_i \in U(SRPT_{1+\epsilon}, t) \setminus U(Opt_1, t)$, with the additional condition that the release time and size satisfy $(t - r_i) \geq 8p_i/\epsilon$. Intuitively this additional condition guarantees that the jobs in $D$ now must be sufficiently old relative to their size. An association scheme maps each $J_i \in D$ to an $ep_i/4(1 + \epsilon)$ amount of volume from $V(t, Opt_1, r_j \leq t - \frac{4p_i}{4(1+\epsilon)}(t - r_i), p_j \leq p_i)$ so that no amount of volume is mapped to by more than one job.

We show in Lemma 7 that the existence of an association scheme implies that the local competitiveness conditions hold. Lemma 8 proves an invariant that we then use in Lemma 9 to show that an association scheme exists.

Lemma 7. If an association scheme exists, then the local competitiveness conditions hold at time $t$.

Proof. By the definition of $D$, any job $J_i$ in $U(SRPT_{1+\epsilon}, t)$ that does not lie in $D$ either must lie in $U(Opt_1, t)$ or must have age no more than $8p_i/\epsilon$. Thus, we can bound $F^p(SRPT_{1+\epsilon})$ as follows:

\begin{equation}
F^p(SRPT_{1+\epsilon}) = p \int_t^\infty \text{Age}^p(SRPT_{1+\epsilon}, t) dt 
\end{equation}

\begin{equation}
= O \left( \int_t^\infty \text{Age}^p(D, t) dt + \int_t^\infty \text{Age}^p(Opt_1, t) dt + \sum_{i=1}^n \int_{x=0}^{8p_i/\epsilon} x^{p-1} dx \right) 
\end{equation}

\begin{equation}
= O \left( \int_t^\infty \text{Age}^p(D, t) dt + F^p(Opt_1) + \sum_{i=1}^n \left( \frac{8p_i}{\epsilon} \right)^p \right) 
\end{equation}

\begin{equation}
= O \left( \int_t^\infty \text{Age}^p(D, t) dt + \frac{1}{\epsilon^{p}} F^p(Opt_1) \right) 
\end{equation}

\begin{equation}
= O \left( \frac{1}{\epsilon^{p}} F^p(Opt_1) \right). 
\end{equation}

Line (14) is from the definition of $F^p$. Line (15) follows since $\int_{x=0}^{8p_i/\epsilon} x^{p-1} dx$ is the maximum contribution to $\int_t^\infty \text{Age}^p(SRPT_{1+\epsilon}, t) dt$ that job $J_i$ can make before it reaches age $8p_i/\epsilon$. Line (16) follows from the definition of $F^p(Opt_1)$. Line (17) follows since the flow time of $J_i$ is at least $p_i$ in $Opt_1$. Line (18) follows by reasoning identical to that used in the proof of Theorem 2 to show that $\int_t^\infty \text{Age}^p(D, t) dt = O \left( \frac{1}{\epsilon^{p}} F^p(Opt_1) \right)$.

The result for stretch also follows directly by observing that $J_i$ is associated only with volume composed of jobs with size no more than $p_i$, and the contribution of these jobs to stretch is at least as large. □
Lemma 8. For all times $u$ and values of $p$, and for any 1-speed processor schedule $Opt_1$, the following inequality holds:

$$V(u, Opt_1, p_j \leq p) \geq \frac{\epsilon}{1+\epsilon}P(u, SRPT_{1+\epsilon}, p_j \leq p) + \frac{1}{1+\epsilon}V(u, SRPT_{1+\epsilon}, p_j \leq p) - \frac{1}{1+\epsilon}p.$$

Proof. The proof is by induction on the event that may happen at time $u$. First we note that the case of job arrivals and either $SRPT_{1+\epsilon}$ or $Opt_1$ finishing a job is quite easy and is handled exactly as in the proof of Lemma 4. Thus it suffices to consider the event when jobs are being worked on. We consider three cases.

First, if $SRPT_{1+\epsilon}$ is running a job with size $\leq p$, then since $SRPT_{1+\epsilon}$ has a $(1+\epsilon)$-speed processor, the right side of the inequality decreases at rate 1, which is at least as fast as the rate at which the left side can possibly decrease, and hence the inequality holds inductively.

Second, if $SRPT_{1+\epsilon}$ is running a job with remaining volume $> p$, this must mean that there is no job alive of size $\leq p$, and hence the right side of the inequality is $-p/(1+\epsilon)$, while the left side is nonnegative.

We now consider the final case when $SRPT_{1+\epsilon}$ is running a job $J_l$ with size $> p$ and remaining volume $\leq p$. In fact, in this case we will show that the following stronger inequality holds:

$$V(u, Opt_1, p_j \leq p) \geq \frac{1}{1+\epsilon}P(u, SRPT_{1+\epsilon}, p_j \leq p) - \frac{1}{1+\epsilon}p.$$

Note that this inequality is stronger than the inequality that we want to prove since the size of a job is at least as large as its remaining volume. Let $u' \leq u$ be the last time when $SRPT_{1+\epsilon}$ was running $J_l$ and its remaining volume just reached exactly $p$. Even if $Opt_1$ was working on jobs of size at most $p$ during the entire interval $[u', u]$, we have

$$V(u, Opt_1, p_j \leq p) \geq V(u', Opt_1, p_j \leq p) + Q(p_j \leq p, r_j \in [u', u]) - (u - u')$$

$$\geq Q(p_j \leq p, r_j \in [u', u]) - (u - u').$$

Now consider the $SRPT_{1+\epsilon}$ schedule. Note that, since $SRPT_{1+\epsilon}$ was running $J_l$ at time $u'$, no other jobs of size $\leq p$ were present under $SRPT_{1+\epsilon}$. Moreover, as $J_l$ is still alive at time $u$, $SRPT_{1+\epsilon}$ must have been working on jobs of remaining volume at most $p$ during $[u', u]$. Also, since $J_l$ is being run at time $u$ (by the nature of the $SRPT$ scheduling algorithm), it must be that any job $J_j$ of size $\leq p$ that arrived during $[u', u]$ is either already completed or has not been executed at all; i.e., $p_j(u) = p$. Using these observations, we have the following chain of equalities:

$$P(u, SRPT_{1+\epsilon}, p_j \leq p) = V(u, SRPT_{1+\epsilon}, p_j \leq p)$$

$$= V(u', SRPT_{1+\epsilon}, p_j \leq p) + Q(p_j \leq p, r_j \in [u', u])$$

$$-(1+\epsilon)(u - u')$$

$$= p + Q(p_j \leq p, r_j \in [u', u]) - (1+\epsilon)(u - u'),$$

which implies that

$$u - u' = \frac{1}{1+\epsilon}(p + Q(p_j \leq p, r_j \in [u', u]) - P(u, SRPT_{1+\epsilon}, p_j \leq p)).$$
Combining this with (21) we get that
\[ V(u, \text{Opt}_1, p_j \leq p) \geq Q(p_j \leq p, r_j \in [u', u]) \]
\[ - \frac{1}{1 + \epsilon} (p + Q(p_j \leq p, r_j \in [u', u]) - P(u, \text{SRPT}_{1+\epsilon}, p_j \leq p)) \]
or, equivalently,
\[ V(u, \text{Opt}_1, p_j \leq p) \geq \frac{\epsilon}{1 + \epsilon} Q(p_j \leq p, r_j \in [u', u]) \]
\[ - \frac{1}{1 + \epsilon} p + \frac{1}{1 + \epsilon} P(u, \text{SRPT}_{1+\epsilon}, p_j \leq p). \]
Since \( Q(p_j \leq p, r_j \in [u', u]) \) is nonnegative, we know that
\[ V(u, \text{Opt}_1, p_j \leq p) \geq \frac{1}{1 + \epsilon} P(u, \text{SRPT}_{1+\epsilon}, p_j \leq p) - \frac{1}{1 + \epsilon} p. \]
This then implies (20).

**Lemma 9.** An association scheme exists.

**Proof.** To construct the association scheme, we consider the jobs in \( D \) in nondecreasing order of their sizes and associate job \( J_i \) with any \( cp_i/4(1+\epsilon) \) units of volume in \( V(t, \text{Opt}_1, r_j \leq t', p_j \leq p_i) \) that have not been associated previously with a lower index job in \( D \). This construction will clearly not fail on job \( J_i \) if \( V(t, \text{Opt}_1, r_j \leq t', p_j \leq p_i) \) contains enough volume to associate all the jobs \( J_1, \ldots, J_i \). Thus it is sufficient to prove
\[ (22) \quad V(t, \text{Opt}_1, r_j \leq t', p_j \leq p_i) \geq \frac{\epsilon}{4(1 + \epsilon)} Q(J_i \in D, p_j \leq p_i). \]
By an argument identical to that used to obtain (7) (except that \( \text{SJF}_{1+\epsilon} \) is replaced by \( \text{SRPT}_{1+\epsilon} \)), we can upper bound the right side of (22) by
\[ (23) \quad Q(J_i \in D, p_j \leq p_i) \leq P(r_i, \text{SRPT}_{1+\epsilon}, p_j \leq p_i) + (t - r_i). \]
By the argument analogous to that used for (8), we can lower bound the left side of (22) as
\[ (24) \quad V(t, \text{Opt}_1, p_j \leq p_i, r_j \leq t') \geq V(r_i, \text{Opt}_1, p_j \leq p_i) + Q(r_j \in (r_i, t'], p_j \leq p_i) - (t - r_i). \]
Applying Lemma 8 with \( u = r_i \) and \( p = p_i \), the first term on the right side of (24) can be bounded as
\[ (25) \quad V(r_i, \text{Opt}_1, p_j \leq p_i) \geq \frac{\epsilon}{1 + \epsilon} P(r_i, \text{SRPT}_{1+\epsilon}, p_j \leq p_i) \]
\[ + \frac{1}{1 + \epsilon} V(r_i, \text{SRPT}_{1+\epsilon}, p_j \leq p_i) - \frac{1}{1 + \epsilon} p_i. \]
Getting back to proving (22), it suffices to show that the right side of (25) exceeds the right side of (23). Thus by canceling the common terms, it is sufficient to prove
\[ (26) \quad \frac{4 + 3\epsilon}{4(1 + \epsilon)} V(r_i, \text{SJF}_{1+\epsilon}, p_j \leq p_i) + Q(r_j \in (r_i, t'], p_j \leq p_i) - \frac{1}{1 + \epsilon} p_i \geq \frac{4 + 5\epsilon}{4(1 + \epsilon)} (t - r_i). \]
We now focus on computing a suitable upper bound on the term \((t - r_i)\). Since \( \text{SRPT}_{1+\epsilon} \) did not finish \( J_i \) by time \( t' \), since during the time period \((r_i, t']\) it must have
been the case that \( \text{SETF}_{1+\epsilon} \) was running only jobs with remaining volume at most \( p_i \), and since \( \text{SETF}_{1+\epsilon} \) has a \((1+\epsilon)\)-speed processor,

\[
(1+\epsilon)(t' - r_i) \leq V(r_i, \text{SETF}_{1+\epsilon}, p_j \leq p_i) + Q(r_j \in (r_i, t'], p_j \leq p_i).
\]

By the nature of \( \text{SRPT} \), at any time \( u \) and for any volume \( v \), there can be at most one job that has size at least \( v \) and remaining volume less than \( v \). By applying this argument with \( v = p_i \), we obtain that

\[
V(r_i, \text{SRPT}_{1+\epsilon}, p_j \leq p_i) \leq V(r_i, \text{SRPT}_{1+\epsilon}, p_j \leq p_i) + p_i.
\]

By our choice of \( t' = t - \epsilon(t - r_i)/(4(1+\epsilon)) \), it follows that \((1+\epsilon)(t' - r_i) = \frac{4+3\epsilon}{4}(t - r_i)\).

Together with (27) and (28), this implies

\[
\frac{4+3\epsilon}{4}(t - r_i) \leq V(r_i, \text{SRPT}_{1+\epsilon}, p_j \leq p_i) + Q(r_j \in (r_i, t'], p_j \leq p_i) + p_i.
\]

Since \( J_i \in D \), the age \((t - r_i)\) of \( J_i \) is at least \( \frac{8p_i}{\epsilon} \) or, equivalently, \( p_i \leq \frac{\epsilon}{8}(t - r_i) \). By subtracting \( \frac{2p_i}{\epsilon}(t - r_i) \) from the left side of (29) and subtracting \( 2p_i \) from the right side of (29), we get that

\[
\left(\frac{4+2\epsilon}{4}\right)(t - r_i) \leq V(r_i, \text{SRPT}_{1+\epsilon}, p_j \leq p_i) + Q(r_j \in (r_i, t'], p_j \leq p_i) - p_i.
\]

Multiplying (30) by \( \frac{(4+5\epsilon)}{(1+\epsilon)(4+2\epsilon)} \), we obtain that

\[
\left(\frac{4+5\epsilon}{(1+\epsilon)(4+2\epsilon)}\right) \cdot \left(V(r_i, \text{SRPT}_{1+\epsilon}, p_j \leq p_i) + Q(r_j \in (r_i, t'], p_j \leq p_i) - p_i\right)
\]

\[
\geq \frac{4+5\epsilon}{4(1+\epsilon)}(t - r_i).
\]

Now, (26) follows by observing that the right side of (31) is same as that of (26), but the coefficient of each positive (respectively, negative) term of the left side of (26) is larger (respectively, smaller) than the corresponding term on the left side of (31).

\[
7. \textbf{Analysis of SETF.} \text{ We now consider the analysis of SETF. Recall that SETF always runs a job } J_j \text{ that has been run the least, that is, for which } x_j(t) \text{ is minimal. We first show that SETF is a } (1+\epsilon)\text{-speed, } O(1/\epsilon^{2+2/p})\text{-competitive algorithm for minimizing the } \ell_p \text{ norm of flow time. We then show that SETF is } (1+\epsilon)\text{-speed, } O(\log^2 B)\text{-competitive for minimizing the } \ell_p \text{ norm of stretch, where } B \text{ is the ratio of the maximum to minimum job size.}
\]

To analyze the performance of \( \text{SETF}_{1+\epsilon} \) on an input \( \mathcal{I} \), we perform a series of transformations, where at each stage we possibly require an additional speed-up of \((1+\epsilon)\). This sequence of transformations gives a fairly general way to relate the performance of \( \text{SETF} \) to the performance of \( \text{SJF} \). We begin by defining an intermediate policy MLF and show that the performances of \( \text{SETF} \) and MLF are related (modulo a factor of \((1+\epsilon)\) in the speed-up). We then transform the input \( \mathcal{I} \) slightly to an instance \( \tilde{\mathcal{I}} \) and show that the performance of MLF on \( \tilde{\mathcal{I}} \) is related to the performance of MLF on \( \mathcal{I} \). The crucial property of the instance \( \tilde{\mathcal{I}} \) is that the performance of MLF on \( \tilde{\mathcal{I}} \) can be viewed as the performance of SJF on another input instance \( \mathcal{L} \), where \( \mathcal{L} \) is a union of various geometrically scaled down copies of \( \mathcal{I} \). The steps thus far relate
MLF(\mathcal{I})$ to $SJF(\mathcal{L})$. The final step, and the heart of the analysis, is a general scheme that relates the performance of $SJF$ on $\mathcal{L}$ to the performance of $SJF$ on $\mathcal{I}$. This gives us a way to relate the performances of $SETF(\mathcal{I})$ and $SJF(\mathcal{I})$. Finally, the results about the competitiveness of $SETF$ for flow time and stretch follow from those about the competitiveness of $SJF$.

Our MLF is a variation of the standard Multilevel Feedback Queue algorithm [13, 28], where the quanta for the queues are set as a function of $\epsilon$. Let $\ell_i = \epsilon((1 + \epsilon)^i - 1)$, $i \geq 0$, and let $q_i = \ell_{i+1} - \ell_i = \epsilon^2(1 + \epsilon)^i$, $i \geq 0$. In MLF a job $J_j$ is in level $k$ at time $t$ if $\ell_k \leq x_j(t) < \ell_{k+1}$. The number of levels $L$ is $O(\log_{1+\epsilon}(\frac{1}{\epsilon}))$. MLF maintains the invariant that it is always running the earliest arriving job in the smallest nonempty level. It is crucial to note that MLF is defined only for the purposes of analysis. One consequence (implicit in the choice of quanta above) is that we can assume without loss of generality that the smallest job size in the instance is exactly 1 and the largest job size is at most $B$. This assumption about the absolute sizes of jobs is made only so that we can fix the sizes of quanta in MLF. In particular, $SETF$ does not use this fact in any way.

**Lemma 10.** For any instance $\mathcal{I}$ and for any job $J_j \in \mathcal{I}$, $J_j$ completes in $SETF_{1+\epsilon}$ before $J_j$ completes in MLF$_1$.

Proof. For a job with $x_j(t) \leq z$, let $p_{\leq z}(j,t) = \min(p_j, z) - x_j(t)$, that is, the amount of work that must be done on $J_j$ until either $J_j$ completes or until $J_j$ has been processed for $z$ time units. Let $U_{\leq z}(A,t)$ denote the set of unfinished jobs in algorithm $A$ at time $t$ that have less than $z$ work done on them. Let $V_{\leq z}(A,t)$ denote $\sum_{j \in U_{\leq z}(A,t)} p_{\leq z}(j,t)$. Now, as $SETF$ is always working on the job with least work done, it is easy to see that for any time $t$ and amount of work $x$, $V_{\leq z}(SETF_s, t) \leq V_{\leq z}(A_s, t)$ for all algorithms $A$ and all speeds $s$.

To reach a contradiction suppose that there is some job $J_j$ which completes earlier in $MLF_1$ than in $SETF_{1+\epsilon}$. Clearly $MLF_1$ must then finish the volume $V_{\leq p_j}(MLF_1, r_j)$ before time $C_j(MLF_1)$. Moreover, at time $C_j(MLF_1)$ all the jobs in $U(MLF_1, t)$ must be in level $k$ such that $\ell_k \leq p_j < \ell_{k+1}$ and hence have at least $p_j/(1+\epsilon)$ amount of work done on them; otherwise $J_j$ would not be run by MLF$_1$ at time $C_j(MLF_1)$. Consider the schedule $A_{1+\epsilon}$ that at all times $t$ runs the job that MLF$_1$ is running at time $t$ (and idles if this job is already completed). Hence, $A_{1+\epsilon}$ would have completed $J_j$, and all previously arriving jobs with processing time less that $p_j$, by time $C_j(MLF_1)$ since $A_{1+\epsilon}$ has a $(1 + \epsilon)$-speed processor. Hence, by the property of $SETF$ from the previous paragraph, $SETF_{1+\epsilon}$ completes $J_j$ by time $C_j(MLF_1)$, which is our contradiction.

Lemma 10 implies that

$$F^p(SETF(\mathcal{I}), s) \leq F^p(MLF(\mathcal{I}), s/(1+\epsilon)).$$

Thus we will focus henceforth on relating MLF and Opt. We begin by defining several instances that are derived from the original instance. Let the original instance be $\mathcal{I}$. Let $\mathcal{J}$ be the instance obtained from $\mathcal{I}$ as follows. Consider a job $J_j \in \mathcal{I}$ and let $i$ be the smallest integer such that $p_j + \epsilon \leq (1 + \epsilon)^i$. The processing time of $J_j$ in $\mathcal{J}$ is then $(1 + \epsilon)^i$. Let $\mathcal{K}$ be the instance obtained from $\mathcal{J}$ by decreasing each job size by $\epsilon$. Thus, each job in $\mathcal{K}$ has size $\ell_k = \epsilon((1 + \epsilon)^k - 1)$ for some $k$. Note that this $\ell_k$ is the same as the quanta for level $k$ in the definition of MLF. Moreover, in this transformation from $\mathcal{I}$ to $\mathcal{K}$, the size of a job does not decrease, and it increases by at most a factor of $(1 + \epsilon)$. This follows because in the worse case a job of size $\epsilon((1 + \epsilon)^i - 1)$ can increase in size to at most $\epsilon((1 + \epsilon)^{i+1} - 1)$. Similarly, in the
transformation from \( I \) to \( J \), the size of a job does not decrease, and it increases by a factor of at most \((1 + \epsilon)^2\). Since MLF has the property that increasing the length of a particular job will not decrease the completion time of any job, we can conclude that

\[
F^p(MLF(I), s/(1 + \epsilon)) \leq F^p(MLF(K), s/(1 + \epsilon)).
\]

Finally we create an instance \( L \) by replacing each job of size \( \epsilon((1 + \epsilon)^k - 1) \) in \( K \) by \( k \) jobs of size \( q_0, q_1, \ldots, q_{k-1} \), where \( q_i = \ell_{i+1} - \ell_i \), as defined in the definition of MLF. Note that \( \ell_k = q_0 + q_1 + \cdots + q_{k-1} \). For a job \( J_j \in K \), we denote the corresponding jobs in \( L \) by \( J_{j,0}, J_{j,1}, \ldots, J_{j,k-1} \), where job \( J_{j,i} \) has size \( q_i \).

A crucial observation about the execution of MLF on instance \( K \) is that it can be modeled by the execution of SJF on instance \( L \). More precisely, for any time \( t \) and for any speed \( s \), \( SJF_s(L) \) is working on a job \( J_{j,b} \in L \) if and only if \( MLF_s(K) \) is working on job \( J_j \in K \) that is in level \( b \) at time \( t \). In particular, this implies that the completion time of \( J_j \) in \( MLF_s(K) \) is exactly the completion time of some job \( J_{j,b} \in SJF_s(L) \). Hence,

\[
F^p(MLF(K), s/(1 + \epsilon)) \leq F^p(SJF(L), s/(1 + \epsilon)).
\]

By Theorem 2, we know that

\[
F^p(SJF(L), s/(1 + \epsilon)) = O(1/\epsilon^p)F^p(Opt(L), s/(1 + \epsilon)^2).
\]

We need to relate the optimal schedule for \( L \) back to the optimal schedule for \( I \). To do this we first relate \( L \) to \( J \) as follows. Let \( L(k) \) denote the instance obtained from \( J \) by multiplying each job size in \( J \) by \( \epsilon/(1 + \epsilon)^k \). Next, we remove from \( L(k) \) any job whose size is less than \( \epsilon^2 \). We claim that \( L = L(1) \cup L(2) \cup \cdots \). To see this, let us consider some job \( J_j \in J \) of size \( \epsilon(1 + \epsilon)^i \). Then, \( L(1) \) contains the corresponding job \( J_j(1) \) of size \( \epsilon/(1 + \epsilon)^i \epsilon (1 + \epsilon)^i = \epsilon^2(1 + \epsilon)^{i-1} = q_{i-1} \). Similarly, \( L(2) \) contains the job \( J_j(2) \) of size \( q_{i-2} \) and so on. In particular, for \( J_j \) of size \( \epsilon(1 + \epsilon)^i \), the job \( J_{j,k} \in L \) is identical to the job \( J_j(i - k) \in L(k) \) for \( k = 0, \ldots, i - 1 \). Summarizing, we have that the \( L(k)'s \) are geometrically scaled down copies of \( J \) and that \( L \) is exactly the union of these \( L(k)'s \).

Our idea at an intuitive level is as follows: Given a good schedule of \( J \), we first obtain a good schedule for \( L(k) \). This will be easy to do, as \( L(k) \) is a scaled down version of \( J \). We will then superimpose the schedules for each \( L(k) \) to obtain a good schedule for \( L \). This will give us a procedure for obtaining a good schedule for \( L \) given a good schedule for \( J \). To put everything in place, we need the following straightforward lemma from [3] which relates \( J \) and \( L(k) \).

**Lemma 11.** Let \( J \) and \( L(k) \) be as defined above, let \( s \geq 1 \) be some speed, and let \( x \geq 1 \) be any real number. Then, for any job \( J_i \in J \) and the corresponding job \( J_i(k) \in L(k) \), the flow time and stretch under SJF are related as

\[
F(J_i(k), SJF(L(k)), \epsilon(1 + \epsilon)^{-k} \cdot x \cdot s) \leq \frac{1}{x} F(J_i, SJF(J), s),
\]

\[
S(J_i(k), SJF(L(k)), \epsilon(1 + \epsilon)^{-k} \cdot x \cdot s) \leq \frac{(1 + \epsilon)^k}{\epsilon x} S(J_i, SJF(J), s).
\]

**Proof.** We first show that for all jobs \( J_j \in J \), we have that \( F(J_j, SJF(J), s) \leq (1/s)F(J_j, SJF(J), 1) \). Let \( y_j(x, s') \) denote the work done on job \( J_j \), after \( x \) units of time since it arrived, under SJF using an \( s' \)-speed processor. We will show a stronger invariant that, for all jobs \( J_j \) and all times \( t \), it is the case that \( y_j((t-r_j)/s, s) \geq y_j(t-
job in $r_j, 1$), by using induction on the time $t$. Suppose that at time $t$, $SJF_1$ works on job $J_j$. Then the invariant clearly continues to hold at time $t$ for jobs besides $J_j$. Consider any job $J_i$ unfinished for $SJF_1$ at time $t$ that is smaller than job $J_j$. By the definition of $J_i$, $SJF_1$ has already completed $J_i$. Since the invariant for $J_i$ holds at the time $C_i < t$ that $SJF_1$ completes job $J_i$, we have that $y_i((C_i - r_i)/s, s) \geq y_i(C_i - r_i, 1) = p_i$. Therefore $SJF_s$ has completed job $J_i$ by time $(C_i - r_i)/s \leq r_i + (t - r_i)/s \leq t$. Thus, by the definition of $SJF_s$, at time $t$ it either works on job $J_j$, and then the invariant for $J_j$ is preserved, or it works on some bigger job, and then $J_j$ is completed and the invariant for $J_j$ holds trivially.

We now show the main result of the lemma. Since $L(k)$ is obtained by scaling jobs in $J$ by $\epsilon(1 + \epsilon)^k$ (and possibly dropping some jobs if they become smaller than $\epsilon^2$), the flow time of every job $J_j(k) \in L(k)$ under $SJF$ with a speed $\epsilon(1 + \epsilon)^{-k}$ processor is at most that of the corresponding job $J_j \in J$, under $SJF$ with a unit speed processor. Thus by the argument in the above paragraph, running $SJF$ on $L(k)$ with an $x \cdot \epsilon(1 + \epsilon)^{-k}$-speed processor yields a $1/x$ times smaller flow time for each job in $L(k)$ than the corresponding job in $J$.

Finally, since the sizes of jobs in $L(k)$ are $\epsilon(1 + \epsilon)^{-k}$ times smaller than in $J$, the result for stretch follows from the result for flow time.

We are now ready to show that a good schedule for $L$ can be obtained from the $SJF$ schedule for $J$ using an additional speed-up of $(1 + 2\epsilon)$.

**Lemma 12.**

$$F^p(\text{Opt}(L), 1 + 2\epsilon) \leq O(1/\epsilon^2)F^p(SJF(J), 1).$$

**Proof.** Given the schedule $SJF(J)$, we construct a schedule $A$ algorithm for $L$ as follows. For each $k = 1, 2, \ldots$, $A$ runs the jobs in $L(k)$ with a speed $s_k = \epsilon(1 + \epsilon/1 + \epsilon)^{-k}$ processor using the algorithm $SJF$. As $L = \bigcup_{k \geq 1} L(k)$, this gives a schedule for $L$. Note that the total speed required by $A$ is at most

$$\sum_{i=1}^{\infty} s_i = \sum_{i=1}^{\infty} \frac{1}{(1 + \epsilon/1 + \epsilon)^i} = \frac{\epsilon}{1 - \frac{1}{1 + 2\epsilon}} = 1 + 2\epsilon.$$

By Lemma 11,

$$F^p(A(L(k)), s_k) \leq \left(\frac{\epsilon(1 + \epsilon)^{-k}}{s_k}\right)^p F^p(SJF(J, 1)) = \left(\frac{1 + 2\epsilon}{(1 + \epsilon)^2}\right)^{kp} F^p(SJF(J, 1)).$$

 Hence, 

$$\frac{F^p(A(L), 1 + 2\epsilon)}{F^p(SJF(J), 1)} \leq \sum_{i=1}^{\infty} \left(\frac{1 + 2\epsilon}{(1 + \epsilon)^2}\right)^{ip} \leq \sum_{i=0}^{\infty} \left(1 - \frac{\epsilon^2}{(1 + \epsilon)^2}\right)^i = O(1/\epsilon^2).$$

The proof then follows because $Opt$ is at least as good as $A$. 

Hence by Lemma 12, Theorem 2, and the fact that job lengths in $J$ are at most
Similarly, since each job in $S$ (39) Combining (32) and (33), we get that
\[
(1 + \epsilon)^2 \text{ times as long as they are in } \mathcal{I}, \text{ we get that}
\]
\[
F^p(\text{Opt}(\mathcal{L}), s/(1 + \epsilon)^2) = O(1/\epsilon^2) \cdot F^p \left( \text{SJF}(\mathcal{J}), \frac{s}{(1 + 2\epsilon)(1 + \epsilon)^2} \right)
\]
\[
= O(1/\epsilon^{p^2+2}) \cdot F^p \left( \text{Opt}(\mathcal{J}), \frac{s}{(1 + 2\epsilon)(1 + \epsilon)^p} \right)
\]
\[
= O(1/\epsilon^{p^2+2}) \cdot F^p \left( \text{Opt}(\mathcal{I}), \frac{s}{(1 + 2\epsilon)(1 + \epsilon)^p} \right). \tag{36}
\]
By stringing together the inequalities (32), (33), (34), (35), and (36), we find that
\[
F^p(\text{SETF}(\mathcal{I}), s) = O(1/\epsilon^{p^2+2}) \cdot F^p \left( \text{Opt}(\mathcal{I}), \frac{s}{(1 + 2\epsilon)(1 + \epsilon)^p} \right).
\]
Hence, we conclude with the main theorem for the flow analysis of SETF.

**Theorem 13.** SETF is $(1 + \epsilon)$-speed, $O(1/\epsilon^{2+2/p})$-competitive for the $\ell_p$ norm of flow time.

We now turn our attention to the stretch analysis of SETF. The analysis is similar to the analysis of flow, but we need a different choice of the speed-up factors $s_k$. By Lemma 10, it follows that
\[
S^p(\text{SETF}(\mathcal{I}), 1 + \epsilon) \leq S^p(\text{MLF}(\mathcal{I}), 1). \tag{37}
\]
Similarly, since each job in $J_i \in \mathcal{I}$ is at most $(1 + \epsilon)^2$ times smaller than the corresponding job in $\mathcal{K}$ and also has size no more than that of the corresponding job in $\mathcal{J}$, we get that
\[
S^p(\text{MLF}(\mathcal{I}), 1) \leq (1 + \epsilon)^{2p} S^p(\text{MLF}(\mathcal{K}), 1). \tag{38}
\]
Combining (32) and (33), we get that
\[
S^p(\text{SETF}(\mathcal{I}), 1 + \epsilon) \leq (1 + \epsilon)^{2p} S^p(\text{MLF}(\mathcal{K}), 1). \tag{39}
\]
Consider the execution of MLF on $\mathcal{K}$ using a 1-speed processor and that of SJF on $\mathcal{L}$ also using a 1-speed processor. By the correspondence between MLF on $\mathcal{K}$ and SJF on $\mathcal{L}$, we know that for each job in $\mathcal{K}$, say of size $\epsilon[(1 + \epsilon)^k - 1]$, its flow time is exactly equal to that of the corresponding job of size $\epsilon^p(1 + \epsilon)^k \in \mathcal{L}(1)$. Thus the ratio of the $p$th power of stretch for these jobs in $\mathcal{L}(1)$ and $\mathcal{K}$ is $(\frac{\epsilon[(1+\epsilon)^k-1]}{\epsilon^p(1+\epsilon)^k})^p$, which is $(\frac{1-(1+\epsilon)^{-k}}{1+\epsilon})^p$. Now since $\epsilon[(1+\epsilon)^k - 1] \geq 1$ for all valid job sizes in $\mathcal{K}$, we have that $(1 + \epsilon)^{-k} \leq \frac{\epsilon}{1+\epsilon}$ and hence that $(\frac{1-(1+\epsilon)^{-k}}{\epsilon^p(1+\epsilon)^k})^p \geq (\epsilon(1 + \epsilon))^{-p}$. Thus the ratio of contributions to the objective $S^p$ for $\mathcal{L}$ and $\mathcal{K}$ will be at least $1/(\epsilon(1 + \epsilon))^p$, which implies that
\[
S^p(\text{MLF}(\mathcal{K}), 1) \leq (\epsilon(1 + \epsilon))^p S^p(\text{SJF}(\mathcal{L}), 1). \tag{40}
\]
Now, applying Theorem 2, it follows that
\[
S^p(\text{SJF}(\mathcal{L}), 1 + \epsilon) = O(1/\epsilon^p) S^p(\text{Opt}(\mathcal{L}), 1). \tag{41}
\]
Combining (40) and (41), we get that
\[
S^p(\text{MLF}(\mathcal{K}), 1) = O((1 + \epsilon)^p) S^p(\text{SJF}(\mathcal{L}), 1). \tag{42}
\]
Recall from Theorem 2 that

$$S^p(SJF(\mathcal{L}), 1 + \epsilon) = O(1/\epsilon^p)S^p(Opt(\mathcal{L}), 1).$$

We now prove a result similar to that of Lemma 12 for stretch.

**Lemma 14.**

$$S^p(Opt(\mathcal{L}), 1 + \epsilon) = O\left(\frac{\log_{1+\epsilon}^p \left(\frac{B}{\epsilon}\right)}{\epsilon^p}\right)S^p(SJF(\mathcal{J}), 1).$$

**Proof.** Let \( L = \log_{1+\epsilon}(B/\epsilon) \) denote the total number of levels. Set

$$s_i = \epsilon(1 + \epsilon)^{-i} \quad \text{for} \quad i = 1, \ldots, \log_{1+\epsilon}(B/\epsilon)$$

and

$$s_i = \epsilon/\log_{1+\epsilon}(B/\epsilon) \quad \text{for} \quad \log_{1+\epsilon}(B/\epsilon) < i \leq L = \log_{1+\epsilon}(B/\epsilon).$$

We run the jobs in \( \mathcal{L}(i) \) using SJF on a speed \( s_i \) processor. The schedule for \( \mathcal{L} \) is then obtained by combining the schedules for each \( \mathcal{L}(i) \) for \( 1 \leq i \leq L \). Notice that the total speed-up required is \( \sum_{i=1}^{L} s_i \), which is at most \( \sum_{i=1}^{\infty} \epsilon(1 + \epsilon)^{-i} + \sum_{i=1}^{\log_{1+\epsilon}(B/\epsilon)} \epsilon/\log_{1+\epsilon}(B/\epsilon) = 1 + \epsilon \).

We now estimate the total stretch for \( \mathcal{L} \) by calculating the total stretch for each \( \mathcal{L}(i) \). By, Lemma 11, we have that \( S^p(SJF(\mathcal{L}(i)), s_i) \) is at most \( (1/s_i^p)S^p(SJF(\mathcal{J}), 1) \) by simple algebraic calculation it can be seen that \( \sum_{i=1}^{L}(1/s_i^p) \) is \( O\left(1/\epsilon^p L^{p+1}\right) \), which is \( O\left(\log_{1+\epsilon}^p(B/\epsilon)\right) \).

Combining (39), (42), (43), and Lemma 14, we get that

$$S^p(SETF(\mathcal{I}), (1 + \epsilon)^2) = O\left(\frac{(1 + \epsilon)^{3p}}{\epsilon^p} \cdot \log_{1+\epsilon}^{p+1} \left(\frac{B}{\epsilon}\right)\right)S^p(SJF(\mathcal{J}), 1).$$

Now, by Theorem 2, we have that

$$S^p(SJF(\mathcal{J}), 1 + \epsilon) = O(1/\epsilon^p)S^p(Opt(\mathcal{J}), 1).$$

Also, since each job \( J_i \in \mathcal{J} \) has size at most \( (1+\epsilon)^2 \) times more than the corresponding job \( J_i \in \mathcal{I} \) (and is not smaller), we trivially have that

$$S^p(Opt(\mathcal{J}), (1 + \epsilon)^2) \leq S^p(Opt(\mathcal{I}), 1).$$

Now, combining (44), (45), and (46) it follows that

$$S^p(SETF, (1 + \epsilon)^5, \mathcal{I}) = O\left(\frac{1}{\epsilon^p} \cdot \log_{1+\epsilon}^{p+1} \left(\frac{B}{\epsilon}\right)\right)S^p(Opt(\mathcal{I}), 1)$$

or, equivalently, that

$$S^p(SETF, (1 + \epsilon)^5, \mathcal{I}) = O\left(\frac{1}{\epsilon^{p+1}} \cdot \log_{1+\epsilon}^{p+1} \left(\frac{B}{\epsilon}\right)\right)S^p(Opt(\mathcal{I}), 1).$$

Thus we conclude with the main result of this subsection.

**Theorem 15.** SETF is a \((1 + \epsilon)-speed, O\left(\frac{1}{\epsilon^{p+1}} \cdot \log_{1+\epsilon}^{1+1/p} \left(\frac{B}{\epsilon}\right)\right)\)-competitive algorithm for the \( \ell_p \) norm of stretch.
8. Lower bound for RR. In this section we show that the standard Processor Sharing, or, equivalently, RR, algorithm is not \((1 + \epsilon)\)-speed, \(O(1)\)-competitive for any \(\ell_p\) norm of flow time, \(1 \leq p < \infty\), and for sufficiently small \(\epsilon\). Recall that RR at any time shares the processor equally among all jobs. This lower bound uses a modification of the lower bound instances in [18, 21, 22].

**Theorem 16.** For every \(p \geq 1\), there is an \(\epsilon > 0\) such that RR is not an \((1 + \epsilon)\)-speed, \(n^{o(1)}\)-competitive algorithm for the \(\ell_p\) norm of flow time. In particular, for every \(p \geq 1\) and \(\epsilon > 0\), RR is \(\Omega(n^{(1-2p)/p})\) for minimizing the \(\ell_p\) norm of flow time.

**Proof.** Suppose that \(\epsilon < 1/2p\) and RR has a processor of speed \(1 + \epsilon\). Consider the following instance. Two jobs of size \(p_0 = 1\) arrive at \(r_0 = 0\). Next we have a collection of \(n\) jobs whose release times and sizes are defined as follows. The first job of size \(p_1 = p_0(1 - \frac{1 + \epsilon}{2})\) arrives at time \(r_1 = p_0\). In general the \(i\)th job has size \(p_i = (1 - \frac{1 + \epsilon}{2^i})p_{i-1}\) and \(r_i = \sum_{j=0}^{i-1} p_j\). The instance has the following properties which are easily verified:

- Except for one job of size 1 which arrives at time 0, each job under SRPT (with a speed 1 processor) has a flow time equal to its size. This follows as the arrival time of the \(i\)th job satisfies \(r_i = \sum_{j=0}^{i-1} p_j\) for all \(i \geq 1\). The job of size 1 has flow time equal to 1 plus \(\sum_{j=0}^n p_j\).
- Under RR(1,\(\epsilon\)) (with a \((1 + \epsilon)\)-speed processor) all the jobs keep accumulating and finish simultaneously at time \(t = (2p_0 + \sum_{j=1}^n p_i)/(1 + \epsilon)\). This follows by observing that for any \(i\) such that \(1 \leq i \leq n\), all the jobs that have arrived thus far have remaining processing time exactly \(p_i\).

We now consider the relevant quantities. First observe that by definition, \(p_i = \Pi_{j=1}^{i} (1 - \frac{1}{2^j}) = \frac{1}{i+1} \Pi_{j=1}^{i} (1 - \epsilon/j)\). Using the fact that for all \(x \geq 0\), \(1 - x \leq e^{-x}\), we get that \(p_i \leq \frac{1}{i+1} e^{-e^{H(i)}}\), where \(H(i)\) is the \(i\)th harmonic number. Finally, using the fact that \(H(i) \geq \ln i\), we get that \(p_i \leq \frac{1}{i+1} i^{-\epsilon} \leq i^{-(1+\epsilon)}\).

We now upper bound \(F^p(SRPT, 1)\), and hence \(F^p(\text{Opt}, 1)\). As every job other than the first job of size 1 has a flow time equal to its size, the contribution to \(F^p\) is \(\sum_{i=0}^{\infty} p_i^p \leq \sum_{i=0}^{\infty} i^{-1+\epsilon} = O(1)\). For the single job of size 1, its flow time is \(1 + \sum_{i=0}^{n} p_i \leq \sum_{i=0}^{\infty} p_i \leq 1 + \sum_{i=0}^{\infty} i^{-1+\epsilon} = O(1)\), and hence we get that \(F^p(SRPT, 1) = O(1)\).

On the other hand, in RR(1,\(\epsilon\)) each job with size \(p_i\) has flow time at least \((i + 2)p_i\) (since this job shares the processor equally with at least \(i + 2\) jobs throughout its execution). We now lower bound \(p_i\). Using the fact that \(e^{-2\epsilon} \leq 1 - x\) for \(x \leq \frac{1}{2}\), we get that \(p_i = \frac{1}{i+1} \Pi_{j=1}^{i} (1 - \epsilon/j) \geq \frac{1}{i+1} e^{-2e^{H(i)}}\). Since \(H(i) \leq \ln i + 1\), we get \(p_i \geq \frac{1}{i+1} (ei)^{-2\epsilon} = \Omega(i^{-(1+2\epsilon)})\). Thus \((i + 2)p_i = \Omega(i^{-2\epsilon})\). This gives us that \(F^p(\text{RR}, 1 + \epsilon)\) is at least \(\sum_{i=1}^{n} i^{-2p_i}\), which is \(\Omega(n^4)\) for \(2\epsilon p \leq 1 - \delta\). In particular, if \(p = 2\), then RR is not \(O(1)\)-competitive even with a \((1 + \epsilon)\) speed-up for any \(\epsilon < \frac{1}{4}\). \(\square\)

9. Analysis of HDF. In this section we show that HDF, a natural generalization of SJF to the weighted case, is a \((1 + \epsilon)\)-speed, \(O(1/\epsilon^2)\)-competitive online algorithm for minimizing the weighted \(\ell_p\) norms of flow time. The algorithm HDF always works on the job which has the largest weight-to-size ratio. The ties are broken in favor of the partially executed job. For technical reasons, we require that the weights are positive integers. We will show that HDF is \((1 + \epsilon)\)-speed, \(O(1/\epsilon^2)\)-competitive for minimizing the weighted \(\ell_p\) norms of flow time. The main idea of the proof will be to reduce the weighted problem to an unweighted problem and then invoke the result for \(\ell_p\) norms of unweighted flow time. We first define the relevant notation.
Given an instance $\mathcal{I}$, we define an instance $\mathcal{I}'$ obtained by applying the following transformation to each job in $\mathcal{I}$: Consider a job $J_i \in \mathcal{I}$. The instance $\mathcal{I}'$ is obtained by replacing $J_i$ by $w_i$ identical jobs each of size $p_i/w_i$, weight 1, and release time $r_i$. We denote these $w_i$ jobs by $J_i^1, \ldots, J_i^{w_i}$. Let $X_i = \{J_i^1, \ldots, J_i^{w_i}\}$ denote this collection of jobs obtained from $J_i$. Note that all jobs in $\mathcal{I}'$ have the same weight.

**Lemma 17.** For $\mathcal{I}$ and $\mathcal{I}'$ as defined above,

$$F^p(\text{Opt}(\mathcal{I}'), 1) \leq WF^p(\text{Opt}(\mathcal{I}), 1).$$

**Proof.** Let $S$ be the schedule which minimizes the weighted $\ell_p$ norm of flow time for $\mathcal{I}$. Given $S$, we create a schedule for $\mathcal{I}'$ as follows. At any time $t$, $\mathcal{I}'$ will work on a job in $X_i$ if and only if $J_i$ is executed at time $t$ under $S$. Clearly, all jobs in $X_i$ finish when $J_i$ finishes execution. Thus no job in $X_i$ has a flow time higher than that of $J_i$. Let $f_i = F_i(\text{Opt}(\mathcal{I}), 1)$. By definition, the contribution of $J_i$ to $WF^p$ is $w_i f_i^p$. Also, the contribution to the measure $F^p$ of each of the $w_i$ jobs in $X_i$ will be at most $f_i^p$, and hence the total contribution of jobs in $X_i$ to $F^p$ is at most $w_i f_i^p$. Since the optimum schedule for $\mathcal{I}'$ can be no worse than the schedule constructed above, the result follows. 

From Theorem 2 we know that

$$F^p(\text{SJF}(\mathcal{I}'), 1 + \epsilon) = O\left(\frac{1}{\epsilon^p}\right) F^p(\text{Opt}(\mathcal{I}'), 1).$$

We now relate the performance of HDF on $\mathcal{I}$ with a $(1 + \epsilon)$ times faster processor to that of SJF on $\mathcal{I}'$.

**Lemma 18.**

$$WF^p(\text{HDF}(\mathcal{I}), 1 + \epsilon) \leq \left(1 + \frac{1}{\epsilon}\right)^p F^p(\text{SJF}(\mathcal{I}'), 1).$$

**Proof.** We claim that for every job $J_i \in \mathcal{I}$ and every time $t$, if $J_i$ is unfinished at time $t$ in $\text{HDF}_{1+\epsilon}(\mathcal{I})$, then at least $\frac{1}{1+\epsilon} w_i$ jobs in $X_i \in \mathcal{I}'$ are unfinished at time $t$ in $\text{SJF}_1(\mathcal{I}')$.

The claim above immediately implies the result for the following reason. For notational simplicity let $f_i = F_i(\text{HDF}_{1+\epsilon}(\mathcal{I}))$. Consider the time $t^{-}$ just before $C_i(\text{HDF}_{1+\epsilon}(\mathcal{I}))$. Then $J_i$ contributes exactly $w_i f_i^p$ to $WF^p(\text{HDF}(\mathcal{I}), 1 + \epsilon)$, while the $\geq \epsilon w_i/(1 + \epsilon)$ jobs in $X_i$ that are unfinished by time $t$ contribute at least $\epsilon w_i/(1 + \epsilon) f_i^p$ to $F^p(\text{SJF}(\mathcal{I}'), 1)$. By summing the contributions over all the jobs, the result follows.

We now prove the claim. Suppose for the sake of contradiction that $t$ is the earliest time when $J_i$ is unfinished in $\text{HDF}_{1+\epsilon}(\mathcal{I})$ and there are fewer than $\epsilon w_i/(1 + \epsilon)$ jobs from $X_i$ left in $\text{SJF}_1(\mathcal{I}')$. Since $J_i$ is unfinished in $\text{HDF}_{1+\epsilon}(\mathcal{I})$ and $\text{HDF}_{1+1}(\mathcal{I})$ has a $1 + \epsilon$ faster processor, $\text{HDF}_{1+1}(\mathcal{I})$ has spent less than $p_i/(1 + \epsilon)$ time on $J_i$, whereas $\text{SJF}_1(\mathcal{I}')$ has spent strictly more than $p_i/(1 + \epsilon)$ time on $X_i$. Thus there was some time $t'$, such that $r_i \leq t' < t$, during which $\text{HDF}_{1+1}(\mathcal{I})$ was running $J_j \neq J_i$ while $\text{SJF}_1(\mathcal{I}')$ was working on some job from $X_i$. Since $t' \geq r_i$, it follows from the property of $\text{HDF}_{1+1}(\mathcal{I})$ that $J_i$ has a higher density than that of $J_j$. This implies that jobs in $X_i$ have a smaller size than $X_j$. Since $\text{SJF}_1(\mathcal{I}')$ works on $X_i$ at time $t'$, it must have already finished all the jobs in $X_j$ by $t'$. Since $J_j$ is unfinished at time $t'$, this contradicts our assumption of the minimality of $t$. 

**Theorem 19.** HDF is $(1 + \epsilon)$-speed, $O(1/\epsilon^2)$-competitive for minimizing the weighted $\ell_p$ norms of flow time.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.
Proof. By (48) and (49) we have that
\[
WF^p(HDF(I))(1 + \epsilon)^2 = O(1/\epsilon^{2p}F^p(Opt(I'), 1).
\]
Combining this with (47) gives us the result. \(\square\)

10. Analysis of WSETF. We first describe the algorithm WSETF. We then show that WSETF is \((1 + \epsilon)\)-speed, \(O(1/\epsilon^{2+2/p})\)-competitive with respect to the weighted \(\ell_p\) norm of flow. For technical reasons, we require that the weights be positive integers. As in the analysis of HDF, the main step of our analysis will be to relate the behavior of WSETF on an instance \(I\) with weighted jobs to that of SETF on another instance \(I'\) that consists of unweighted jobs. We then use the results about (unweighted) \(\ell_p\) norms of flow time under SETF to obtain results for WSETF.

Algorithm WSETF. Define the norm \(n_i(t)\) of a job \(J_i\) at time \(t\) to be \(\frac{x_i(t)}{w_i}\). At all times, WSETF splits the processor, proportional to weights of the jobs, among the jobs \(J_i\) that have the smallest norm. So, if \(J_1, \ldots, J_k\) are the jobs that have the smallest norm, then \(J_j\) for \(i = 1, \ldots, k\) will receive a \(w_j/\sum_{i=1}^k w_i\) fraction of the processor.

Note that for all jobs \(J_i\) that WSETF executes, the norm increases at the same rate.

Given an instance \(I\) of weighted jobs, let \(I'\) denote the instance defined as in the analysis of HDF. That is, each job \(J_i \in I\) is a collection \(X_i = \{J'_i, \ldots, J'_{i, m_i}\}\) of \(w_i\) identical jobs each of size \(p_i/w_i\), weight 1, and release time \(r_i\). Lemma 20 then relates the schedules \(WSETF_s(I)\) and \(SETF_s(I')\) for any speed \(s\) processor.

Lemma 20. At any time \(t\), a job \(J_i \in I\) is unfinished and has had its volume reduced by an amount \(x_i(t)\) in \(WSETF(I)\) if and only if each job in \(X_i \in I'\) is unfinished and has had its volume reduced by exactly \(x_i(t)/w_i\) in \(SETF(I')\). In particular, this implies that if \(J_i\) has flow time \(f_i\) in \(WSETF_s(I)\), then each \(J'_{i, k} \in X_i\) for \(k = 1, \ldots, w_i\) has flow time \(f_i\) in \(SETF_s(I')\).

Proof. We view the schedule \(WSETF_s(I)\) as follows: If at any time it is the case that \(WSETF_s(I)\) reduces the volume of a job \(J_i\) by \(z\) units, then we think of it as allocating \(z/w_i\) units of processing to each of the \(w_i\) jobs in the collection \(X_i\). Thus the norm of job \(J_i\) in \(WSETF_s(I)\) is equal to the amount of service received by a job in \(X_i\) in \(SETF_s(I')\). Since WSETF at any time shares the processor among jobs with the smallest norm, in the ratio of their weights, this is identical to the behavior of \(SETF_s(I')\), which works equally on the jobs which have received the least service.

Theorem 21. WSETF is a \((1 + \epsilon)\)-speed, \(O(1/\epsilon^{2+2/p})\)-competitive nonclairvoyant algorithm for minimizing the weighted \(\ell_p\) norms of flow time.

Proof. By Lemma 20 we know that if \(J_i \in I\) has flow time \(f_i\) in \(WSETF_s(I)\), then the \(w_i\) jobs in \(X_i\) have flow time \(f_i\) in \(SETF_s(I')\). Thus the \(\ell_p\) norm of unweighted flow time for \(SETF(I')\) is \((\sum w_i f_i^p)^{1/p}\), which is identical to the weighted flow time for \(I\) in \(WSETF(I)\). This implies that
\[
WF^p(WSETF(I), 1) = F^p(SETF(I'), 1).
\]
Therefore, for some constant \(C\), we have that
\[
WF^p(WSETF(I), 1) = CWF^p(SETF(I'), 1) = F^p(SETF(I'), 1).
\]
By (47) we know that
\[
F^p(SETF(I'), 1) \leq WF^p(SETF(I'), 1).
\]
By Theorem 13, we know that
\[
F^p(SETF(I'), (1 + \epsilon)) = O(1/\epsilon^{2p+2})F^p(SETF(I'), 1).
\]
Now, by (50), (52), and (47) we get that
\[ W^p (WSETF(I), 1 + \epsilon) = O(\frac{1}{\epsilon^{2p+2}}) W^p (Opt(I), 1). \]
Thus the result follows.  

Acknowledgments. We thank Cliff Stein for helpful discussions. We thank the anonymous referees for comments that helped improve the presentation of the paper.

REFERENCES


