Server scheduling to balance priorieties, fairness, and average qualityof service

SIAM J. COMPUT.
Vol. 39, No. 7, pp. 3311–3335

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 ℓ_p norm, $1 < p < \infty$. Thus we consider server scheduling strategies to optimize the ℓ_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 ℓ_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 ℓ_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 ℓ_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 ℓ_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 ϵ.

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 ℓ_1 norm, of the response times, and the maximum, or ℓ_∞ norm, of the response times. It is well known that the scheduling algorithm Shortest Remain-

*Received by the editors September 28, 2009; accepted for publication (in revised form) May 12, 2010; published electronically July 29, 2010.
http://www.siam.org/journals/sicomp/39-7/77222.html
†IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 (nikhil@us.ibm.com).
‡Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260 (kirk@cs.pitt.edu). This author's work was supported in part by a grant from the US Air Force, an IBM faculty award, and NSF grants CCR-0098752, ANIR-0123705, CNS-0325353, CCF-0448196, CCF-0514058, IIS-0534531, and CCF-0830558.

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 performance [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 optimizing 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, $\left(\sum_{i=1}^{n} F_i^p/n\right)^{1/p}$, 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.
>
> — *Operating Systems Concepts*, Silberschatz and Galvin.

The variance is the expected squared distance from the mean, $\sum_{i=1}^{n}(F_i - Z)^2$, where $Z = \sum_{i=1}^{n} F_i/n$. 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 $\sum_{i=1}^{n} F_i^2$, 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.

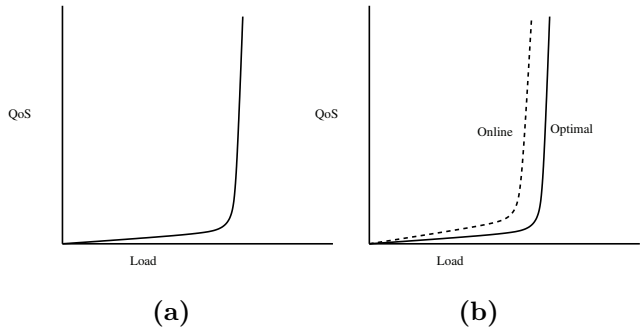

FIG. 1. (a) *Standard QoS curve.* (b) *The worst possible QoS curve of an* $(1+\epsilon)$*-speed,* $O(1)$*-competitive online algorithm.*

We first give a negative result that shows that optimizing for the ℓ_2 norm of response times is much harder than optimizing for the ℓ_1 and ℓ_∞ 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 ℓ_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

$$\max_I \frac{F(A_s(I))}{F(Opt_1(I))} \leq c,$$

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 ϵ) such that for all inputs I,

$$\frac{F(A_{1+\epsilon}(I))}{F(Opt_1(I))} \leq c.$$

Such a scalable algorithm is $O(1)$-competitive on inputs I where $Opt_1(I)$ is approximately $Opt_{1+\epsilon}(I)$. The loads in the usual performance curve shown in Figure 1(a), where $Opt_1(I)$ is not approximately $Opt_{1+\epsilon}(I)$, 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 *SRPT* is a scalable algorithm for the objective of the ℓ_p norm of response times. This result suggests that perhaps the fear that *SRPT* will unnecessarily starve jobs is unfounded since *SRPT* scales almost as well with load as the optimal algorithm for the ℓ_p norm of response times.

1.2. Our results. We analyze the performance of various standard scheduling algorithms for the objectives of the ℓ_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 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 *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, *SJF* and *SRPT*. We show that the algorithms *SJF* and *SRPT* are scalable with respect to all ℓ_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 ℓ_p norms, $1 < p < \infty$.

Algorithm	Speed	Competitive ratio		Section
		ℓ_p Norm of flow	ℓ_p Norm of stretch	
General clairvoyant algorithm	1	$n^{\Omega(1)}$	$n^{\Omega(1)}$	4
SJF	$(1+\epsilon)$	$O(1/\epsilon)$	$O(1/\epsilon)$	5
SRPT	$(1+\epsilon)$	$O(1/\epsilon)$	$O(1/\epsilon)$	6
SETF	$(1+\epsilon)$	$O(1/\epsilon^{2+2/p})$	$O(\frac{1}{\epsilon^{3+1/p}} \cdot \lg^{1+1/p}(\frac{B}{\epsilon}))$	7
RR	$(1+\epsilon)$	$\Omega(n^{1-2\epsilon p})$	$\Omega(n)$	8
General nonclairvoyant algorithm	$(1+\epsilon)$		$\Omega(\min\{n, \log B\})$	4
		Weighted ℓ_p norm of flow		
HDF	$(1+\epsilon)$	$O(1/\epsilon^2)$		9
WSETF	$(1+\epsilon)$	$O(1/\epsilon^{2+2/p})$		10

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 ℓ_p norms of flow. For the ℓ_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 ℓ_p norm of flow would be $(\sum_{i=1}^{n} w_i F_i^p)^{1/p}$. We show that both *HDF* and the nonclairvoyant algorithm *WSETF* are scalable for the ℓ_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 ℓ_p norm of flow, $1 < p < \infty$, and for sufficiently small ϵ. 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-

alently, *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 $\Omega(n^{1/3})$, and the competitive ratio of every randomized nonclairvoyant algorithm against an oblivious adversary is $\Omega(\log n)$ [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 $\mathcal{J} = J_1, \ldots, J_n$. 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 $C_i(\mathcal{S})$ of a job J_i in a schedule $\mathcal{S}$ is the first time t after r_i when $p_i(t) = 0$. The flow time of J_i in $\mathcal{S}$ is $F_i(\mathcal{S}) = C_i(\mathcal{S}) - r_i$. The stretch S_i of J_i in $\mathcal{S}$ is $F_i(\mathcal{S})/p_i$. If the schedule $\mathcal{S}$ 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 $X(A(\mathcal{I}), s)$ 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 ℓ_p norm of flow is $(\sum_i F_i^p)^{1/p}$, the ℓ_p norm of stretch is $(\sum_i S_i^p)^{1/p}$, and the ℓ_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 ℓ_p norm of flow, and a similar relationship holds for stretch and weighted flow. We use Opt to denote the optimal schedule for the objective function under consideration. So $F^p(Opt_s(\mathcal{I}))$ 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 $\mathcal{S}$ at time t if $r_i \leq t \leq C_i(\mathcal{S})$, and it has age $t - r_i$ at this time. We use $U(\mathcal{S}, t)$ to denote the collection of unfinished jobs at time t in the schedule $\mathcal{S}$. For a schedule $\mathcal{S}$, we use $Age^p(\mathcal{S}, t)$ to denote the sum over all jobs $J_i \in U(\mathcal{S}, t)$ of $(t - r_i)^{p-1}$. For a schedule $\mathcal{S}$, we use $SAge^p(\mathcal{S}, t)$ to denote the sum over all jobs $J_i \in U(\mathcal{S}, t)$ of $(t - r_i)^{p-1}/p_i^p$. Note that $F^p(\mathcal{S}) = p \int_t Age^p(\mathcal{S}, t)dt$. 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 $\frac{1}{p}F_i^p$. Thus, the total payment made by all the jobs is proportional to F^p. Thus to show that an algorithm $A_{1+\epsilon}$ is $O(\frac{1}{\epsilon})$-competitive for the ℓ_p norm of flow, it is sufficient to show that at any time t the following *local competitiveness condition* holds:

$$Age^p(U(A_{1+\epsilon}, t), t) = O\left(\frac{1}{\epsilon^p}\right) \cdot Age^p(U(Opt_1, t), t). \tag{1}$$

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

$$SAge^p(U(A_{1+\epsilon}, t), t) = O\left(\frac{1}{\epsilon^p}\right) \cdot SAge^p(U(Opt_1, t), t). \tag{2}$$

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 α. So, for example, $V(r_i, Opt_1, r_j \leq r_i, p_j \leq p_i)$ is the amount of volume that Opt_1 has unfinished on jobs J_j that arrived not after r_i, are not larger in size than J_i, and that 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 α. Finally let $Q(\alpha)$ denote the aggregate size of the jobs J_j that satisfy the conditions in the list α. 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 ℓ_p norms of flow and stretch for $1 < p < \infty$.

THEOREM 1. *Let $p > 1$. The competitive ratio, with respect to the ℓ_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 ℓ_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, α, and β 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^α 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^α. 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, $F^p(Opt) = O(L^p + L^\alpha \cdot L^p) = O(L^{\alpha+p})$. The competitive ratio is then $\Omega(L^{\alpha p-\alpha-p})$, which is $\Omega(L)$ by our choice of α.

Now consider the case that A finishes the long job by time L^α. 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 $F^p(Opt) = O(L^\alpha \cdot 1^p + L^{\alpha+\beta} \cdot (1/L^\beta)^p + L^{\alpha p})$. It is obvious that the dominant term is $L^{\alpha p}$, and hence, $F^p(Opt) = O(L^{\alpha p})$. Now consider the subcase that A has at least $L/2$ unit jobs unfinished by time $3L^\alpha/2$. Since these unfinished unit jobs must have been delayed by at least $L^\alpha/2$, $F^p(A) = \Omega(L \cdot L^{\alpha p})$. Clearly in this subcase the competitive ratio is $\Omega(L)$. Alternatively, consider the subcase that A has at most $L/2$ unit jobs unfinished by time $3L^\alpha/2$. Then, as the total unfinished volume by time $3L^\alpha/2$ is L, A has at least $L/2$ unfinished in small jobs at time $3L^\alpha/2$. By the convexity F^p when restricted to jobs of size $1/L^\beta$ (we can gift A the completion of unit jobs), the optimal strategy for A from time $3L^\alpha/2$ onward is to delay each small job by the same amount. Thus A delays $L^{\alpha+\beta}/2$ short jobs by at least $L/2$. Hence in this case, $F^p(A) = \Omega(L^{\alpha+\beta} \cdot L^p)$. This gives a competitive ratio of $\Omega(L^{\alpha+\beta+p-\alpha p})$, which by the choice of β is $\Omega(L)$. As the total number of jobs in the instance is $O(L^{\alpha+\beta})$, the competitive ratio of any online algorithm with respect to the measure F^p is $L = n^{1/(\alpha+\beta)} = n^{(p-1)/(3p-1)}$ and hence $n^{(p-1)/(p(3p-1))}$ with respect to the ℓ_p norm of flow time.

We now consider the S^p objective. In this case, $\alpha = \frac{2p+1}{p-1}$, 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^α. 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, $S^p(Opt) = O(1 + L^\alpha \cdot L^p) = O(L^{\alpha+p})$. The competitive ratio is $\Omega(L^{\alpha p-\alpha-2p})$, which is $\Omega(L)$ by our choice of α.

Now consider the case that A finishes the long job by time L^α. 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 $S^p(Opt) = O(L^\alpha \cdot 1^p + L^{\alpha+\beta} \cdot 1^p + (L^\alpha/L)^p)$. Algebraic simplification shows that $\alpha+\beta \le \alpha p - p$ for our choice of α and β. Hence the dominant term is $L^{\alpha p-p}$, and $S^p(Opt) = O(L^{\alpha p-p})$. 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 $L^\alpha/2$,

$S^p(A) = \Omega(L \cdot L^{\alpha p})$. 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 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, $S^p(A) = \Omega(L^{\alpha+\beta} \cdot L^{(\beta+1)p})$. This gives a competitive ratio of $S^p(A) = \Omega(L^{\alpha+\beta+\beta p+2p-\alpha p})$. By the choice of α and β, 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. ∎

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 input 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 $S^p(Opt) = O(n)$. 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 ℓ_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 ℓ_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 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 $\mathcal{D}$ denote $U(SJF_{1+\epsilon}, t) \setminus U(Opt_1, t)$. An *association scheme* maps each $J_i \in \mathcal{D}$ to an $\epsilon p_i/4(1+\epsilon)$ amount of volume from $V(t, Opt_1, r_j \leq t - \frac{\epsilon}{4(1+\epsilon)}(t - r_i), p_j \leq p_i)$ so that no amount of volume is mapped to by more than one job in $\mathcal{D}$. 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 $J_i \in \mathcal{D}$. Suppose the association scheme maps J_i to $v_{\sigma(j)}$ volume from job $J_{\sigma(j)}$ in Opt_1 for $j = 1, 2, \ldots, |U(Opt_1, t)|$. 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 $j = 1, \ldots, |U(Opt_1, t)|$, 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

$$\sum_{j:v_{\sigma(j)}>0} \frac{v_{\sigma(j)}}{p_{\sigma(j)}} \geq \sum_{j:v_{\sigma(j)}>0} \frac{v_{\sigma(j)}}{p_i} \geq \frac{\epsilon p_i}{4(1+\epsilon)} \cdot \frac{1}{p_i} = \frac{\epsilon}{4(1+\epsilon)} \tag{3}$$

fraction of jobs in 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_i, the value of Age^p for $J_{\sigma(j)}$ is $\Omega(\epsilon^{p-1})$ times the value of Age^p of J_i. The local competitiveness condition for flow (1) thus follows together with (3) and summing up over all jobs in $\mathcal{D}$.

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

LEMMA 4. *Given any arbitrary scheduling instance J, let Opt_1 denote any schedule using a speed 1 processor. Then for all times u and for all values p, the following inequality holds:*

$$V(u, Opt_1, p_j \le p) \ge \frac{\epsilon}{1+\epsilon} P(u, SJF_{1+\epsilon}, p_j \le p) + \frac{1}{1+\epsilon} V(u, SJF_{1+\epsilon}, p_j \le p).$$

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

First, suppose that $SJF_{1+\epsilon}$ is working on some unfinished job J_j with $p_j \le p$. Then, the right side decreases at least as fast as the left side since $SJF_{1+\epsilon}$ 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 $SJF_{1+\epsilon}$, 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 $\le p$ under $SJF_{1+\epsilon}$, then the right side is zero and the inequality is trivially satisfied.

In the case that 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 $p_j \le p$, 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 $\mathcal{D}$ such that $p_1 \le p_2 \le \cdots \le p_k$, and consider them in this order. Assume that we are considering job J_i. Let t' denote the time $t - \frac{\epsilon}{4(1+\epsilon)}(t - r_i)$. Associate with J_i an $\epsilon p_i / 4(1+\epsilon)$ amount of volume taken arbitrarily from $V(t, Opt_1, r_j \le t', p_j \le p_i)$, provided that it has not been previously associated with a lower indexed job in $\mathcal{D}$.

Consider some fixed job $J_i \in \mathcal{D}$. To show that this procedure does not fail on job J_i, it clearly suffices to show that $V(t, Opt_1, p_j \le p_i, r_j \le t')$ contains enough volume to associate all the jobs $J_1, \ldots, J_i$; that is,

$$V(t, Opt_1, p_j \le p_i, r_j \le t') \ge \frac{\epsilon}{4(1+\epsilon)} Q(J_j \in \mathcal{D}, p_j \le p_i). \tag{4}$$

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

$$Q(J_j \in \mathcal{D}, r_j \le r_i, p_j \le p_i) + Q(J_j \in \mathcal{D}, r_j > r_i, p_j \le p_i).$$

Since Opt_1 has to finish all jobs in $\mathcal{D}$ that are released after r_i, by time t, the latter term can be upper bounded as

$$(t - r_i) \ge Q(J_j \in \mathcal{D}, r_j > r_i, p_j \le p_i). \tag{5}$$

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

$$P(r_i, SJF_{1+\epsilon}, p_j \le p_i) \ge Q(J_j \in \mathcal{D}, r_j \le r_i, p_j \le p_i). \tag{6}$$

Combining (5) and (6), we have

$$Q(J_j \in \mathcal{D}, p_j \le p_i) \le P(r_i, SJF_{1+\epsilon}, p_j \le p_i) + (t - r_i). \tag{7}$$

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

$$V(t, Opt_1, p_j \le p_i, r_j \le t') \ge V(r_i, Opt_1, p_j \le p_i) + Q(r_j \in (r_i, t'], p_j \le p_i) - (t - r_i). \tag{8}$$

By Lemma 4 with $u = r_i$ and $p = p_i$, the first term on the right side of (8) can be bounded as

$$V(r_i, Opt_1, p_j \le p_i) \ge \frac{\epsilon}{1+\epsilon} P(r_i, SJF_{1+\epsilon}, p_j \le p_i) + \frac{1}{1+\epsilon} V(r_i, SJF_{1+\epsilon}, p_j \le p_i). \tag{9}$$

Combining (8) and (9) we get that

$$\begin{aligned} V(t, Opt_1, p_j \le p_i, r_j \le t') &\ge \frac{\epsilon}{1+\epsilon} P(r_i, SJF_{1+\epsilon}, p_j \le p_i) \\ &\quad + \frac{1}{1+\epsilon} V(r_i, SJF_{1+\epsilon}, p_j \le p_i) \\ &\quad + Q(r_j \in (r_i, t'], p_j \le p_i) - (t - r_i) \\ &\ge \frac{\epsilon}{4(1+\epsilon)} P(r_i, SJF_{1+\epsilon}, p_j \le p_i) \\ &\quad + \frac{4+3\epsilon}{4(1+\epsilon)} V(r_i, SJF_{1+\epsilon}, p_j \le p_i) \\ &\quad + Q(r_j \in (r_i, t'], p_j \le p_i) - (t - r_i), \end{aligned} \tag{10}$$

where the second inequality follows by noting that

$$P(r_i, SJF_{1+\epsilon}, p_j \le p_i) \ge V(r_i, SJF_{1+\epsilon}, p_j \le p_i).$$

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

$$\frac{4+3\epsilon}{4(1+\epsilon)} V(r_i, SJF_{1+\epsilon}, p_j \le p_i) + Q(r_j \in (r_i, t'], p_j \le p_i) \ge \frac{4+5\epsilon}{4(1+\epsilon)} (t - r_i). \tag{11}$$

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 $\le p_i$ during the entire time $[r_i, t]$. These jobs of size $\le p_i$ 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

$$V(r_i, SJF_{1+\epsilon}, p_j \le p_i) + Q(r_j \in (r_i, t'], p_j \le p_i) \ge (1+\epsilon)(t' - r_i) = \frac{4+3\epsilon}{4}(t - r_i). \tag{12}$$

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

$$\left(\frac{(4+5\epsilon)}{(1+\epsilon)(4+3\epsilon)} \right) \cdot (V(r_i, SJF_{1+\epsilon}, p_j \le p_i) + Q(r_j \in (r_i, t'], p_j \le p_i)) \ge \frac{4+5\epsilon}{4(1+\epsilon)} (t - r_i). \tag{13}$$

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* ℓ_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 $\mathcal{D}$ 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 $\mathcal{D}$ now must be sufficiently old relative to their size. An *association scheme* maps each $J_i \in \mathcal{D}$ to an $\epsilon p_i/4(1+\epsilon)$ amount of volume from $V(t, Opt_1, r_j \leq t - \frac{\epsilon}{4(1+\epsilon)}(t - r_i), p_j \leq p_i)$ so that no amount of volume is mapped to by more than one job.

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 $\mathcal{D}$, any job J_i in $U(SRPT_{1+\epsilon}, t)$ that does not lie in $\mathcal{D}$ 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:

$$F^p(SRPT_{1+\epsilon}) = p\int_t Age^p(SRPT_{1+\epsilon}, t)dt \tag{14}$$

$$= O\left(\int_t Age^p(\mathcal{D}, t)dt + \int_t Age^p(Opt_1, t)dt + \sum_{i=1}^{n}\int_{x=0}^{8p_i/\epsilon} x^{p-1}dx\right) \tag{15}$$

$$= O\left(\int_t Age^p(\mathcal{D}, t)dt + F^p(Opt_1) + \sum_{i=1}^{n}\left(\frac{8p_i}{\epsilon}\right)^p\right) \tag{16}$$

$$= O\left(\int_t Age^p(\mathcal{D}, t)dt + F^p(Opt_1) + \frac{1}{\epsilon^p}F^p(Opt_1)\right) \tag{17}$$

$$= O\left(\int_t Age^p(\mathcal{D}, t)dt + \frac{1}{\epsilon^p}F^p(Opt_1)\right) \tag{18}$$

$$= O\left(\frac{1}{\epsilon^p}F^p(Opt_1)\right). \tag{19}$$

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 $\int_t Age^p(SRPT_{1+\epsilon}, t)dt$ 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 (19) follows by reasoning identical to that used in the proof of Theorem 2 to show that $\int_t Age^p(\mathcal{D}, t)dt = O\left(\frac{1}{\epsilon^p}F^p(Opt_1)\right)$.

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:*

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

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 $\le p$, 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 $\le p$, 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 $\le p$. In fact, in this case we will show that the following stronger inequality holds:

$$V(u, Opt_1, p_j \le p) \ge \frac{1}{1+\epsilon} P(u, SRPT_{1+\epsilon}, p_j \le p) - \frac{1}{1+\epsilon} p. \tag{20}$$

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 $u' \le u$ 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

$$\begin{aligned} V(u, Opt_1, p_j \le p) &\ge V(u', Opt_1, p_j \le p) + Q(p_j \le p, r_j \in [u', u]) - (u - u') \\ &\ge Q(p_j \le p, r_j \in [u', u]) - (u - u'). \end{aligned} \tag{21}$$

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 $\le p$ 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 $\le p$ 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:

$$\begin{aligned} P(u, SRPT_{1+\epsilon}, p_j \le p) &= V(u, SRPT_{1+\epsilon}, p_j \le p) \\ &= V(u', SRPT_{1+\epsilon}, p_j \le p) + Q(p_j \le p, r_j \in [u', u]) \\ &\quad -(1+\epsilon)(u - u') \\ &= p + Q(p_j \le p, r_j \in [u', u]) - (1+\epsilon)(u - u'), \end{aligned}$$

which implies that

$$u - u' = \frac{1}{1+\epsilon} \left(p + Q(p_j \le p, r_j \in [u', u]) - P(u, SRPT_{1+\epsilon}, p_j \le p) \right).$$

Combining this with (21) we get that

$$V(u, Opt_1, p_j \le p) \ge Q(p_j \le p, r_j \in [u', u]) \\ -\frac{1}{1+\epsilon}\left(p + Q(p_j \le p, r_j \in [u', u]) - P(u, SRPT_{1+\epsilon}, p_j \le p)\right)$$

or, equivalently,

$$V(u, Opt_1, p_j \le p) \ge \frac{\epsilon}{1+\epsilon} Q(p_j \le p, r_j \in [u', u]) \\ -\frac{1}{1+\epsilon}p + \frac{1}{1+\epsilon}P(u, SRPT_{1+\epsilon}, p_j \le p).$$

Since $Q(p_j \le p, r_j \in [u', u])$ is nonnegative, we know that

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

This then implies (20). ∎

LEMMA 9. *An association scheme exists.*

Proof. To construct the association scheme, we consider the jobs in $\mathcal{D}$ in nondecreasing order of their sizes and associate job J_i with any $\epsilon p_i/4(1+\epsilon)$ units of volume in $V(t, Opt_1, r_j \le t', p_j \le p_i)$ that have not been associated previously with a lower index job in $\mathcal{D}$. This construction will clearly not fail on job J_i if $V(t, Opt_1, r_j \le t', p_j \le p_i)$ contains enough volume to associate all the jobs $J_1, \ldots, J_i$. Thus it is sufficient to prove

$$V(t, Opt_1, r_j \le t', p_j \le p_i) \ge \frac{\epsilon}{4(1+\epsilon)} Q(J_j \in \mathcal{D}, p_j \le p_i). \tag{22}$$

By an argument identical to that used to obtain (7) (except that $SJF_{1+\epsilon}$ is replaced by $SRPT_{1+\epsilon}$), we can upper bound the right side of (22) by

$$Q(J_j \in \mathcal{D}, p_j \le p_i) \le P(r_i, SRPT_{1+\epsilon}, p_j \le p_i) + (t - r_i). \tag{23}$$

By the argument analogous to that used for (8), we can lower bound the left side of (22) as

$$V(t, Opt_1, p_j \le p_i, r_j \le t') \ge V(r_i, Opt_1, p_j \le p_i) + Q(r_j \in (r_i, t'], p_j \le p_i) - (t - r_i). \tag{24}$$

Applying Lemma 8 with $u = r_i$ and $p = p_i$, the first term on the right side of (24) can be bounded as

$$V(r_i, Opt_1, p_j \le p_i) \ge \frac{\epsilon}{1+\epsilon}P(r_i, SRPT_{1+\epsilon}, p_j \le p_i) \\ +\frac{1}{1+\epsilon}V(r_i, SRPT_{1+\epsilon}, p_j \le p_i) - \frac{1}{1+\epsilon}p_i. \tag{25}$$

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

$$\frac{4+3\epsilon}{4(1+\epsilon)}V(r_i, SJF_{1+\epsilon}, p_j \le p_i) + Q(r_j \in (r_i, t'], p_j \le p_i) - \frac{1}{1+\epsilon}p_i \ge \frac{4+5\epsilon}{4(1+\epsilon)}(t - r_i). \tag{26}$$

We now focus on computing a suitable upper bound on the term $(t - r_i)$. Since $SRPT_{1+\epsilon}$ did not finish J_i by time t', since during the time period $(r_i, t']$ it must have

been the case that $SRPT_{1+\epsilon}$ was running only jobs with remaining volume at most p_i, and since $SRPT_{1+\epsilon}$ has a $(1+\epsilon)$-speed processor,

$$(1+\epsilon)(t'-r_i) \le V(r_i, SRPT_{1+\epsilon}, p_j(r_i) \le p_i) + Q(r_j \in (r_i, t'], p_j \le p_i). \tag{27}$$

By the nature of 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

$$V(r_i, SRPT_{1+\epsilon}, p_j(r_i) \le p_i) \le V(r_i, SRPT_{1+\epsilon}, p_j \le p_i) + p_i. \tag{28}$$

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

$$\frac{4+3\epsilon}{4}(t-r_i) \le V(r_i, SRPT_{1+\epsilon}, p_j \le p_i) + Q(r_j \in (r_i, t'], p_j \le p_i) + p_i. \tag{29}$$

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

$$\left(\frac{4+2\epsilon}{4}\right)(t-r_i) \le V(r_i, SRPT_{1+\epsilon}, p_j \le p_i) + Q(r_j \in (r_i, t'], p_j \le p_i) - p_i. \tag{30}$$

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

$$\begin{aligned}\left(\frac{(4+5\epsilon)}{(1+\epsilon)(4+2\epsilon)}\right) \cdot (V(r_i, SRPT_{1+\epsilon}, p_j \le p_i) + Q(r_j \in (r_i, t'], p_j \le p_i) - p_i) \\ \ge \frac{4+5\epsilon}{4(1+\epsilon)}(t-r_i).\end{aligned} \tag{31}$$

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

To analyze the performance of $SETF_{1+\epsilon}$ 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 *SETF* to the performance of *SJF*. We begin by defining an intermediate policy *MLF* and show that the performances of *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 ϵ. 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 $O(\log_{1+\epsilon}(\frac{B}{\epsilon}))$. *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 $\sum_{J_j \in U_{\leq z}(A,t)} p_{\leq z}(j,t)$. 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, $V_{\leq x}(SETF_s, t) \leq V_{\leq x}(A_s, t)$ 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

$$F^p(SETF(\mathcal{I}), s) \leq F^p(MLF(\mathcal{I}), s/(1+\epsilon)). \tag{32}$$

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 $p_j + \epsilon \leq \epsilon(1+\epsilon)^i$. The processing time of J_j in $\mathcal{J}$ is then $\epsilon(1+\epsilon)^i$. Let $\mathcal{K}$ be the instance obtained from $\mathcal{J}$ by decreasing each job size by ϵ. Thus, each job in $\mathcal{K}$ has size $\ell_k = \epsilon((1+\epsilon)^k - 1)$ for some k. Note that this ℓ_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 $\mathcal{I}$ to $\mathcal{J}$, 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

$$F^p(MLF(\mathcal{I}), s/(1+\epsilon)) \leq F^p(MLF(\mathcal{K}), s/(1+\epsilon)). \tag{33}$$

Finally we create an instance $\mathcal{L}$ by replacing each job of size $\epsilon((1+\epsilon)^k - 1)$ in $\mathcal{K}$ 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 $J_j \in \mathcal{K}$, we denote the corresponding jobs in $\mathcal{L}$ 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 $\mathcal{K}$ is that it can be modeled by the execution of *SJF* on instance $\mathcal{L}$. More precisely, for any time t and for any speed s, $SJF_s(\mathcal{L})$ is working on a job $J_{j,b} \in \mathcal{L}$ if and only if $MLF_s(\mathcal{K})$ is working on job $J_j \in \mathcal{K}$ that is in level b at time t. In particular, this implies that the completion time of J_j in $MLF_s(\mathcal{K})$ is exactly the completion time of some job $J_{j,b} \in SJF_s(\mathcal{L})$. Hence,

$$F^p(MLF(\mathcal{K}), s/(1+\epsilon)) \leq F^p(SJF(\mathcal{L}), s/(1+\epsilon)). \tag{34}$$

By Theorem 2, we know that

$$F^p(SJF(\mathcal{L}), s/(1+\epsilon) = O(1/\epsilon^p) F^p(Opt(\mathcal{L}), s/(1+\epsilon)^2). \tag{35}$$

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

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

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

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

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

Proof. We first show that for all jobs $J_j \in \mathcal{J}$, we have that $F(J_j, SJF(\mathcal{J}), s) \leq (1/s) F(J_j, SJF(\mathcal{J}), 1)$. 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 $y_j((t-r_j)/s, s) \geq y_j(t-$

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

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

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

LEMMA 12.

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

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

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

By Lemma 11,

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

Hence,

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

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 $\mathcal{J}$ are at most

$(1+\epsilon)^2$ times as long as they are in $\mathcal{I}$, we get that

$$\begin{aligned} F^p(Opt(\mathcal{L}), s/(1+\epsilon)^2) &= O(1/\epsilon^2) \cdot F^p\left(SJF(\mathcal{J}), \frac{s}{(1+2\epsilon)(1+\epsilon)^2}\right) \\ &= O(1/\epsilon^{p+2}) \cdot F^p\left(Opt(\mathcal{J}), \frac{s}{(1+2\epsilon)(1+\epsilon)^3}\right) \\ &= O(1/\epsilon^{p+2}) \cdot F^p\left(Opt(\mathcal{I}), \frac{s}{(1+2\epsilon)(1+\epsilon)^5}\right). \end{aligned} \tag{36}$$

By stringing together the inequalities (32), (33), (34), (35), and (36), we find that

$$F^p(SETF(\mathcal{I}), s) = O(1/\epsilon^{2p+2}) \cdot F^p\left(Opt(\mathcal{I}), \frac{s}{(1+2\epsilon)(1+\epsilon)^4}\right).$$

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* ℓ_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

$$S^p(SETF(\mathcal{I}), 1+\epsilon) \le S^p(MLF(\mathcal{I}), 1). \tag{37}$$

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

$$S^p(MLF(\mathcal{I}), 1) \le (1+\epsilon)^{2p} S^p(MLF(\mathcal{K}), 1). \tag{38}$$

Combining (32) and (33), we get that

$$S^p(SETF(\mathcal{I}), 1+\epsilon) \le (1+\epsilon)^{2p} S^p(MLF(\mathcal{K}), 1). \tag{39}$$

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 $\epsilon^2(1+\epsilon)^k \in \mathcal{L}(1)$. Thus the ratio of the pth power of stretch for these jobs in $\mathcal{L}(1)$ and $\mathcal{K}$ is $(\frac{\epsilon[(1+\epsilon)^k-1]}{\epsilon^2(1+\epsilon)^k})^p$, which is $(\frac{1-(1+\epsilon)^{-k}}{\epsilon})^p$. Now since $\epsilon[(1+\epsilon)^k - 1] \ge 1$ for all valid job sizes in $\mathcal{K}$, we have that $(1+\epsilon)^{-k} \le \frac{\epsilon}{1+\epsilon}$ and hence that $(\frac{1-(1+\epsilon)^{-k}}{\epsilon})^p \ge (\epsilon(1+\epsilon))^{-p}$. 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

$$S^p(MLF(\mathcal{K}), 1) \le (\epsilon(1+\epsilon))^p S^p(SJF(\mathcal{L}), 1). \tag{40}$$

Now, applying Theorem 2, it follows that

$$S^p(SJF(\mathcal{L}), 1+\epsilon) = O(1/\epsilon^p) S^p(Opt(\mathcal{L}), 1). \tag{41}$$

Combining (40) and (41), we get that

$$S^p(MLF(\mathcal{K}), 1) = O((1+\epsilon)^p) S^p(SJF(\mathcal{L}), 1). \tag{42}$$

Recall from Theorem 2 that

$$S^p(SJF(\mathcal{L}), 1+\epsilon) = O(1/\epsilon^p) S^p(Opt(\mathcal{L}), 1). \tag{43}$$

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

LEMMA 14.

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

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

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

and

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

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 $1 \le i \le L$. 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 $\sum_{i=1}^{L} (\frac{1}{s_i})^p$ is $O(\frac{1}{\epsilon^p} L^{p+1})$, which is $O(\frac{\log^{p+1}_{1+\epsilon}(B/\epsilon)}{\epsilon^p})$. ∎

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

$$S^p(SETF(\mathcal{I}), (1+\epsilon)^2) = O\left(\frac{(1+\epsilon)^{3p}}{\epsilon^p} \cdot \log^{p+1}_{1+\epsilon}\left(\frac{B}{\epsilon}\right)\right) S^p(SJF(\mathcal{J}), 1). \tag{44}$$

Now, by Theorem 2, we have that

$$S^p(SJF(\mathcal{J}), 1+\epsilon) = O(1/\epsilon^p) S^p(Opt(\mathcal{J}), 1). \tag{45}$$

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

$$S^p(Opt(\mathcal{J}), (1+\epsilon)^2) \le S^p(Opt(\mathcal{I}), 1). \tag{46}$$

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

$$S^p(SETF, (1+\epsilon)^5, \mathcal{I}) = O\left(\frac{1}{\epsilon^{2p}} \cdot \log^{p+1}_{1+\epsilon}\left(\frac{B}{\epsilon}\right)\right) S^p(Opt(\mathcal{I}), 1)$$

or, equivalently, that

$$S^p(SETF, (1+\epsilon)^5, \mathcal{I}) = O\left(\frac{1}{\epsilon^{3p+1}} \cdot \lg^{p+1}\left(\frac{B}{\epsilon}\right)\right) S^p(Opt(\mathcal{I}), 1).$$

Thus we conclude with the main result of this subsection.

THEOREM 15. *SETF is a* $(1+\epsilon)$*-speed,* $O(\frac{1}{\epsilon^{3+1/p}} \cdot \lg^{1+1/p}\left(\frac{B}{\epsilon}\right))$*-competitive algorithm for the* ℓ_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 ℓ_p norm of flow time, $1 \le p < \infty$, and for sufficiently small ϵ. 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 $p \ge 1$, there is an $\epsilon > 0$ such that RR is not an $(1+\epsilon)$-speed, $n^{o(1)}$-competitive algorithm for the ℓ_p norm of flow time. In particular, for every $p \ge 1$ and $\epsilon > 0$, RR is $\Omega(n^{(1-2\epsilon p)/p})$ for minimizing the ℓ_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 ith job has size $p_i = (1 - \frac{1+\epsilon}{i+1})p_{i-1}$ 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 $SRPT_1$ (with a speed 1 processor) has a flow time equal to its size. This follows as the arrival time of the ith job satisfies $r_i = \sum_{j=0}^{i-1} p_j$ for all $i \ge 1$. The job of size 1 has flow time equal to $1 + \sum_{j=0}^{n} p_j$.
- 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 $1 \le i \le n$, 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, $p_i = \Pi_{j=1}^{i} \frac{(j-\epsilon)}{j+1} = \frac{1}{i+1}\Pi_{j=1}^{i}(1-\epsilon/j)$. Using the fact that for all $x \ge 0$, $1 - x \le e^{-x}$, we get that $p_i \le \frac{1}{i+1} e^{-\epsilon \cdot H(i)}$, where $H(i)$ is the ith harmonic number. Finally, using the fact that $H(i) \ge \ln i$, we get that $p_i \le \frac{1}{i+1} i^{-\epsilon} \le i^{-(1+\epsilon)}$.

We now upper bound $F^p(SRPT, 1)$, and hence $F^p(Opt, 1)$. 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 $\sum_{j=0}^{n} p_j^p \le \sum_{j=0}^{\infty} j^{-(1+\epsilon)p} = O(1)$. For the single job of size 1, its flow time is $1 + \sum_{j=0}^{n} p_j \le \sum_{j=0}^{\infty} p_j \le 1 + \sum_{j=0}^{\infty} j^{-(1+\epsilon)} = O(1)$, 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 $e^{-2x} \le 1 - x$ for $x \le \frac{1}{2}$, we get that $p_i = \frac{1}{i+1}\Pi_{j=1}^{i}(1-\epsilon/j) \ge \frac{1}{i+1} e^{-2\epsilon H(i)}$. Since $H(i) \le \ln i + 1$, we get $p_i \ge \frac{1}{i+1}(ei)^{-2\epsilon} = \Omega(i^{-(1+2\epsilon)})$. Thus $(i+2)p_i = \Omega(i^{-2\epsilon})$. This gives us that $F^p(RR, 1+\epsilon)$ is at least $\sum_{i=1}^{n} i^{-2\epsilon p}$, which is $\Omega(n^{\delta})$ for $2\epsilon p \le 1 - \delta$. In particular, if $p = 2$, then *RR* is not O(1)-competitive even with a $(1+\epsilon)$ speed-up for any $\epsilon < \frac{1}{4}$. ∎

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 ℓ_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 ℓ_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 ℓ_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 $J'_{i1}, \ldots, J'_{iw_i}$. Let $X_i = \{J'_{i1}, \ldots, J'_{iw_i}\}$ 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,*

$$F^p(Opt(\mathcal{I}'), 1) \leq WF^p(Opt(\mathcal{I}), 1). \tag{47}$$

Proof. Let S be the schedule which minimizes the weighted ℓ_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 $f_i = F_i(Opt(\mathcal{I}), 1)$. 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

$$F^p(SJF(\mathcal{I}'), 1+\epsilon) = O\left(\frac{1}{\epsilon^p}\right) F^p(Opt(\mathcal{I}'), 1). \tag{48}$$

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.

$$WF^p(HDF(\mathcal{I}), 1+\epsilon) \leq \left(1 + \frac{1}{\epsilon}\right)^p F^p(SJF(\mathcal{I}'), 1). \tag{49}$$

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 $HDF_{1+\epsilon}(\mathcal{I})$, then at least $\frac{\epsilon}{1+\epsilon} w_i$ jobs in $X_i \in \mathcal{I}'$ are unfinished at time t in $SJF_1(\mathcal{I}')$.

The claim above immediately implies the result for the following reason. For notational simplicity let $f_i = F_i(HDF_{1+\epsilon}(\mathcal{I}))$. Consider the time t^- just before $C_i(HDF_{1+\epsilon}(\mathcal{I}))$. Then J_i contributes exactly $w_i f_i^p$ to $WF^p(HDF(\mathcal{I}), 1+\epsilon)$, 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 $F^p(SJF(\mathcal{I}'), 1)$. 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 $HDF_{i+1}(\mathcal{I})$ and there are fewer than $\epsilon w_i/(1+\epsilon)$ jobs from X_i left in $SJF_1(\mathcal{I}')$. Since J_i is unfinished in $HDF_{i+1}(\mathcal{I})$ and $HDF_{i+1}(\mathcal{I})$ has a $1+\epsilon$ faster processor, $HDF_{i+1}(\mathcal{I})$ has spent less than $p_i/(1+\epsilon)$ time on J_i, whereas $SJF_1(\mathcal{I}')$ 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 $HDF_{i+1}(\mathcal{I})$ was running $J_j \neq J_i$ while $SJF_1(\mathcal{I}')$ was working on some job from X_i. Since $t' \geq r_i$, it follows from the property of $HDF_{i+1}(\mathcal{I})$ that J_j has a higher density than that of J_i. This implies that jobs in X_j have a smaller size than X_i. Since $SJF_1(\mathcal{I}')$ 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 ℓ_p norms of flow time.*

Proof. By (48) and (49) we have that

$$WF^p(HDF(\mathcal{I})(1+\epsilon)^2) = O(1/\epsilon)^{2p} F^p(Opt(\mathcal{I}'), 1).$$

Combining this with (47) gives us the result. ∎

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 ℓ_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 $\mathcal{I}$ with weighted jobs to that of *SETF* on another instance $\mathcal{I}'$ that consists of unweighted jobs. We then use the results about (unweighted) ℓ_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 $\mathcal{I}$ of weighted jobs, let $\mathcal{I}'$ denote the instance defined as in the analysis of *HDF*. That is, each job $J_i \in \mathcal{I}$ is a collection $X_i = \{J'_{i1}, \ldots, J'_{iw_i}\}$ 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 $WSETF_s(\mathcal{I})$ and $SETF_s(\mathcal{I}')$ for any speed s processor.

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

Proof. We view the schedule $WSETF_s(\mathcal{I})$ as follows: If at any time it is the case that $WSETF_s(\mathcal{I})$ 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 $WSETF_s(\mathcal{I})$ is exactly equal to the amount of service received by a job in X_i in $SETF_s(\mathcal{I}')$. 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 $SETF_s(\mathcal{I}')$, 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 ℓ_p norms of flow time.*

Proof. By Lemma 20 we know that if $J_i \in \mathcal{I}$ has flow time f_i in $WSETF_s(\mathcal{I})$, then the w_i jobs in X_i have flow time f_i in $SETF_s(\mathcal{I}')$. Thus the ℓ_p norm of unweighted flow time for $SETF_1(\mathcal{I}')$ is $(\sum_i w_i f_i^p)^{1/p}$, which is identical to the weighted flow time for $\mathcal{I}$ in $WSETF_1(\mathcal{I})$. This implies that

$$WF^p(WSETF(\mathcal{I}), 1) = F^p(SETF(\mathcal{I}')1). \tag{50}$$

By (47) we know that

$$F^p(Opt(\mathcal{I}'), 1) \leq WF^p(Opt(\mathcal{I}), 1). \tag{51}$$

By Theorem 13, we know that

$$F^p(SETF(\mathcal{I}'), (1+\epsilon)) = O(1/\epsilon^{2p+2}) F^p(Opt(\mathcal{I}'), 1). \tag{52}$$

Now, by (50), (52), and (47) we get that

$$WF^p(WSETF(\mathcal{I}), 1+\epsilon) = O(1/\epsilon^{2p+2})\, WF^p(Opt(\mathcal{I}), 1).$$

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

[1] *Apache: HTTP Server Project*, The Apache Software Foundation, 2009, http://httpd.apache.org/docs/.
[2] N. Bansal and K. Dhamdhere, *Minimizing weighted flow time*, ACM Trans. Algorithms, 3 (2007), article 39.
[3] N. Bansal, K. Dhamdhere, J. Könemann, and A. Sinha, *Non-clairvoyant scheduling for minimizing mean slowdown*, Algorithmica, 40 (2004), pp. 305–318.
[4] N. Bansal and M. Harchol-Balter, *Analysis of SRPT scheduling: Investigating unfairness*, SIGMETRICS Perform. Eval. Rev., 1 (2001), pp. 279–290.
[5] L. Becchetti and S. Leonardi, *Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines*, J. ACM, 51 (2004), pp. 517–539.
[6] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs, *Online weighted flow time and deadline scheduling*, J. Discrete Algorithms, 4 (2006), pp. 339–352.
[7] M. A. Bender, S. Muthukrishnan, and R. Rajaraman, *Approximation algorithms for average stretch scheduling*, J. Sched., 7 (2004), pp. 195–222.
[8] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, New York, 1998.
[9] C. Bussema and E. Torng, *Greedy multiprocessor server scheduling*, Oper. Res. Lett., 34 (2006), pp. 451–458.
[10] C. Chekuri and S. Khanna, *Approximation schemes for preemptive weighted flow time*, in Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing, 2002, pp. 297–305.
[11] C. Chekuri, S. Khanna, and A. Zhu, *Algorithms for minimizing weighted flow time*, in Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, 2001, pp. 84–93.
[12] C. Chekuri, A. Goel, S. Khanna, and A. Kumar, *Multi-processor scheduling to minimize flow time with ϵ resource augmentation*, in Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, 2004, pp. 363–372.
[13] E. Coffman and P. Denning, *Operating Systems Theory*, Prentice–Hall, Englewood Cliffs, NJ, 1973.
[14] M. Crovella, R. Frangioso, and M. Harchol-Balter, *Connection scheduling in web servers*, in Proceedings of the USENIX Symposium on Internet Technologies and Systems, 1999, pp. 243–254.
[15] J. Edmonds, *Scheduling in the dark*, Theoret. Comput. Sci., 235 (2000), pp. 109–141.
[16] M. Harchol-Balter, M. Crovella, and S. Park, *The Case for SRPT Scheduling of Web Servers*, Technical report MIT-LCS-TR-767, MIT, Cambridge, MA, 1998.
[17] B. Kalyanasundaram and K. R. Pruhs, *Fault-tolerant scheduling*, in SIAM J. Comput., 34 (2005), pp. 697–719.
[18] B. Kalyanasundaram and K. Pruhs, *Speed is as powerful as clairvoyance*, J. ACM, 47 (2000), pp. 617–643.
[19] B. Kalyanasundaram and K. Pruhs, *Minimizing flow time nonclairvoyantly*, J. ACM, 50 (2003), pp. 551–567.
[20] D. Knuth, *The TeXbook*, Addison–Wesley, Reading, MA, 1984.
[21] T. Matsumoto, *Competitive analysis of the round robin algorithm*, in Algorithms and Computation, Lecture Notes in Comput. Sci. 650, Springer, Berlin, 1992, pp. 71–77.
[22] R. Motwani, S. Phillips, and E. Torng, *Nonclairvoyant scheduling*, Theoret. Comput. Sci., 130 (1994), pp. 17–47.
[23] S. Muthukrishnan, R. Rajaraman, A. Shaheen, and J. E. Gehrke, *Online scheduling to minimize average stretch*, SIAM J. Comput., 34 (2005), pp. 433–452.
[24] K. Pruhs, *Competitive online scheduling for server systems*, SIGMETRICS Perform. Eval. Rev., 34 (2007), pp. 52–58.

[25] K. Pruhs, J. Sgall, and E. Torng, *Online scheduling*, in Handbook on Scheduling, CRC Press, Boca Raton, FL, 2004.
[26] B. Schroeder and M. Harchol-Balter, *Web servers under overload: How scheduling can help*, ACM Trans. Internet Technol., 6 (2006), pp. 20–52.
[27] A. Silberschatz and P. B. Galvin, *Operating System Concepts*, 5th ed., Addison–Wesley Longman, Reading, MA, 1998.
[28] A. Tanenbaum, *Operating Systems: Design and Implementation*, 2nd ed., Prentice-Hall, Upper Saddle River, NJ, 2001.