

Server Scheduling to Balance Priorities, Fairness and Average Quality of Service

Nikhil Bansal

IBM T. J. Watson Research Center
nikhil@us.ibm.com

Kirk R. Pruhs *

Computer Science Department
University of Pittsburgh
kirk@cs.pitt.edu

Abstract

Often server systems do not implement the best known algorithms for optimizing average Quality of Service (QoS) out of concern of that these algorithms may be insufficiently fair to individual jobs. The standard method for balancing average QoS and fairness is optimize the ℓ_p metric, $1 < p < \infty$. 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 almost fully scalable for the ℓ_p norms of flow and stretch. And that the standard nonclairvoyant algorithm for optimizing average QoS, Shortest Elapsed Time First (*SETF*), is also almost fully scalable for the ℓ_p norms of flow. 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 .

We explain how the apparent goals of the Unix CPU scheduling policy, which incorporates individual job priorities, can be formalized using the weighted ℓ_p norm of flows. We then show that the online algorithm, Highest Density First (*HDF*), and the nonclairvoyant algorithm, Weighted Shortest Elapsed Time First (*WSETF*), are almost fully scalable.

These results argue that the scheduling algorithms *SJF*, *SRPT*, *SETF*, *HDF* and *WSETF* will not starve jobs until the system is near peak capacity.

Categories and Subject Descriptors

C.4: Performance of Systems: Performance Attributes

F.2.2: Nonnumerical Algorithms and Problems: Sequencing and Scheduling

*Supported in part by a grant from the US Air Force, and by NSF grants CCR-0098752, ANIR-0123705, ANI-0325353, and CCF-0448196.

Terms

Algorithms, Performance, Theory

Keywords

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

1 Introduction

1.1 Motivation

Tanenbaum [33, page 704] describes the generic Unix CPU scheduling policy as follows. Each process initially has a *nice* value in the range -20 to 20. Lower *nice* values correspond to processes that are more important. Users can set the *nice* value of a process to be in the range from 0 to 20 with a *nice* system call. Only the system administrator can give a process a negative *nice* value. Once a second the *priority* of a process is recalculated using the formula:

$$priority = CPUusage + nice + base$$

Here the *CPUusage* parameter is an exponential weighted moving average of past CPU usage, the *nice* parameter is the *nice* value for the process, and the *base* parameter is used to give higher priority to jobs that have just returned from some sort of interruption (say for I/O). The high priority jobs are those whose computed *priority* value is smallest. The jobs with highest priority are then scheduled using a Round Robin (RR) policy, typically with the quantum on the order of 100 milliseconds. Round Robin shares the processor equally among all processes.

Round Robin represents an apparent effort to balance between optimizing the worst case Quality of Service (QoS) and optimizing the average case QoS. If the goal was to optimize worst case QoS then the best algorithm would be First Come First Served (FCFS). FCFS optimizes the maximum flow time objective. If the goal was to optimize average QoS then Shortest Elapsed Time First (SETF) is generally considered to be the best nonclairvoyant algorithm [15].

Processes with lower *nice* values get more of the CPU, but the *CPUusage* parameter works to try to prevent starvation. That is, the *CPUusage* parameter will be high for processes that have been run a lot recently, and thus these processes will have a higher computed *priority*, and thus these processes will be given less CPU time in the near future. So it seems that the Unix system designers' goals for the process scheduling policy were:

Goal A: Amongst jobs of the same priority, there should be some balance between optimizing for average QoS and optimizing for worst case QoS.

Goal B: Higher priority jobs should get a greater share of the CPU resources, but lower priority jobs should not be starved.

In this paper we try to formalize these goals and then analyze algorithms with respect to this formalization.

In the literature, the most common QoS measure for a single process/job J_i is clearly flow/response/waiting time $F_i = C_i - r_i$, where C_i is the time that the job completes and r_i is the time that the job enters the system. Another common QoS metric is stretch S_i , which is the ratio F_i/p_i , where p_i is the volume of the job. 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 volume 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.

Let us first consider **Goal A**. 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 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 metric to determine line breaks [25, page 97]. The ℓ_p , $1 < p < \infty$, metric still considers the average in the sense that it takes into account all values, but because x^p is strictly a convex function of x , the ℓ_p norm more severely penalizes outliers than the standard ℓ_1 norm. We thus propose that **Goal A** can be captured by considering optimizing the ℓ_p norm of either flow or stretch.

The most common way that priorities of jobs is formalized is to assume that each job J_i has a positive weight w_i and then to have the objective function be maximizing the weighted QoS. By far the most commonly studied QoS measure for a collection of equal priority jobs is average flow time, and logically enough, the most commonly studied QoS measure for jobs with variable priorities is weighted flow time $\sum w_i \cdot F_i$, e.g. [4, 8, 12, 13]. It is easy to see that even an optimal algorithm for optimizing weighted flow time does not in general accomplish **Goal B** as it can starve low weight jobs if there are always higher weight jobs to be run.

If one wishes to achieve both **Goal A** and **Goal B**, then we propose that the appropriate objective function to optimize would be something like the weighted ℓ_p norms of flow or stretch, that is, $(\sum w_i F_i^p)^{1/p}$ or $(\sum w_i S_i^p)^{1/p}$, where $p > 1$ is some small constant. Note that in any competitive schedule for the weighted ℓ_p norm of flow, a low weight job J_i would eventually be scheduled even in the face of a constant stream of high weight jobs.

This leads us consider server scheduling algorithms for optimizing unweighted and weighted ℓ_p norms, $1 < p < \infty$, of the two standard QoS metrics: flow and stretch.

1.2 Our Results

We summarize our results in table 1.

We first focus on the unweighted case, that is, each job is of equal importance. We show that are no $n^{o(1)}$ -competitive online server scheduling algorithms for any ℓ_p metric, $1 < p < \infty$ of either flow or stretch. Perhaps this is a bit surprising, at least for the flow metric, as there are optimal online algorithms, *SRPT* and *FCFS*, for the ℓ_1 and ℓ_∞ norms.

This negative result motivates us to fall back to 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 $A_s(I)$ denotes the the schedule that algorithm A with a speed s produces on input I , and similarly $Opt_1(I)$ denotes the adversarial schedule for I with a unit speed processor. A $(1 + \epsilon)$ -speed $O(1)$ -competitive algorithm be *almost fully scalable* [31]. In this definition, the constant in the big O may depend on ϵ . If A is a randomized algorithm then the expected value of $F(A_s(I))$ is used. Resource augmentation analysis was proposed as a method for analyzing scheduling algorithms in [24], and the current notation and terminology we use is from [30]. If $s = 1$ then A is simply said to be c -competitive, or have competitive ratio c .

We first consider the standard online clairvoyant algorithms aimed at average QoS, that is, Shortest Job First (*SJF*), and *SRPT*. We show that the algorithms *SJF* and *SRPT* are almost fully scalable with respect to all ℓ_p norms of flow and stretch. Apache, currently one of the most widely used web servers [21], uses a separate process for each request, and uses a First Come First Served (*FCFS*) policy to determine the request that is allocated a free process slot [20]. Although it has been argued that *SRPT* would improve average flow time [6, 16, 19, 32]. the most commonly cited reason for adopting *FCFS* is a desire for some degree of fairness to individual jobs [19].

We show that the standard nonclairvoyant algorithm *SETF* is also almost fully scalable for all ℓ_p norms of flow. For the ℓ_p norm of stretch, we show that given constant speed-up, *SETF* is poly-logarithmically competitive in B , the ratio of the length of the longest job to the shortest job. And we give an essentially matching lower bound. Note that all of the results assume that p is constant, so that multiplicative factors, that are a function of p alone, are absorbed into the constant in the asymptotic notation.

Turning back to weighted norms, we first show that the clairvoyant algorithm Highest Density First (*HDF*), is almost fully scalable. We then introduce a new nonclairvoyant algorithm, Weighted Shortest Elapsed Time First (*WSETF*), and show that it is almost fully scalable. *HDF* always runs the job that has the largest weight to volume ratio. *HDF* is the natural generalization of *SJF*. For a job J_i , let $x_i(t)$ denote the amount of work done on that job by time t . We define the norm of a job J_i at time t as $n_i(t) = \frac{x_i(t)}{w_i}$. Amongst the jobs with the smallest norm, *WSETF* splits the processor proportionally to weights of the jobs. 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.

(a)(b)

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

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. Average QoS curves such as those in figure 1(a) are ubiquitous in server systems [26]. That is, there is a relatively modest degradation in average QoS as the load increases until some threshold is reached — this threshold is essentially the capacity of the system — after which any increase in the load precipitously degrades the average QoS. The concept of load is not so easy to formally define, but generally reflects the number of users of the system. If A is an $(1 + \epsilon)$ -speed c -competitive server scheduling algorithm, and $Opt_1(I) \leq d \cdot Opt_{1+\epsilon}(I)$ then A is at

worst $(c \cdot d)$ -competitive on input 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 $(1 + \epsilon)$ -speed c -competitive online algorithm A should be at no worse than shown in figure 1(b). That is, A should scale reasonably well up to quite near the capacity of the system.

It might be tempting to conclude that all reasonable algorithms should have such scaling guarantees. However, we show that this is not the case in section 8. 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 [33].

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
<i>SJF</i>	$(1 + \epsilon)$	$O(1/\epsilon)$	$O(1/\epsilon)$	5
<i>SRPT</i>	$(1 + \epsilon)$	$O(1/\epsilon)$	$O(1/\epsilon)$	6
<i>SETF</i>	$(1 + \epsilon)$	$O(1/\epsilon^{2+2/p})$	$O(\frac{1}{\epsilon^{3+1/p}} \cdot \lg^{1+1/p}(\frac{B}{\epsilon}))$	7
<i>RR</i>	$(1 + \epsilon)$	$\Omega(n^{1-2\epsilon p})$	$\Omega(n)$	8
General Nonclairvoyant Algorithm	$(1 + \epsilon)$		$\Omega(\min\{n, \log B\})$	4
		Weighted ℓ_p Norm of Flow		
<i>HDF</i>	$(1 + \epsilon)$	$O(1/\epsilon^2)$		9
<i>WSETF</i>	$(1 + \epsilon)$	$O(1/\epsilon^{2+2/p})$		10

Table 1: Resource augmentation results for unweighted and weighted ℓ_p norms.

2 Related Results

The results in the literature that are closest in spirit to those here are found in a series of papers, including [6, 16, 19, 32]. These papers also argue that *SRPT* will not unnecessarily starve jobs any more than Processor Sharing 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 $\Omega(n^{1/3})$, the competitive ratio of every randomized nonclairvoyant algorithm against an oblivious adversary is $\Omega(\log n)$ [28]. The randomized nonclairvoyant algorithm *RMLF*, proposed in [23], is $O(\log n)$ -competitive against an oblivious adversary [7]. The online clairvoyant algorithm *SRPT* is optimal. The online clairvoyant algorithm *SJF* is almost fully scalable [8]. The nonclairvoyant algorithm *SETF* is almost fully scalable [24]. *RR* is also known to be $O(1)$ -speed $O(1)$ -competitive for average

flow time [17].

For minimizing average stretch, [29] show that *SRPT* is 2-competitive. In the offline case, there is a PTAS for average stretch [9]. In the non-clairvoyant case, [5] showed that any algorithm is $\Omega(n)$ competitive (without speedup) and gave a $(1 + \epsilon)$ -speed $O(\text{poly}(\epsilon^{-1} \cdot \log B))$ -competitive algorithm

There have been some prior results on scheduling problems with ℓ_p norms. Load balancing in the ℓ_p norm is discussed in [1, 3, 2]. PTAS for ℓ_p norms of completion times, without release times, is given in [18]. Highest Density First was shown to be $(1 + \epsilon)$ -speed $O(1/\epsilon)$ -competitive for weighted flow time in [8]. This immediately implies a similar result for *SJF* for average flow time and for average stretch. [4] give an $O(\log W)$ competitive algorithm for weighted flow time. Other results for weighted flow time can be found in [13, 12, 4].

Subsequent to this research, two papers [11, 14] extend our results to the multiprocessor setting. The paper [11] shows that *HDF* is almost fully scalable in the multiprocessor setting. The paper [14] shows that almost fully scalable 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

We assume a collection of jobs $\mathcal{J} = J_1, \dots, J_n$. For J_i , the release time is denoted by r_i , the volume/size by p_i , and weight by w_i . The density d_i of a job J_i is w_i/p_i . Without loss of generality, we may assume that all job sizes and job weights are integers.

A schedule is a function from time to the job being run at that time. If a speed s processor works on a job J_i for an amount of time t then the volume of J_i is reduced by $s \cdot t$. Note that we allow preemption, that is not requirement that the processing of a job be continuous in time. We use $p_i(t)$ to denote the remaining 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 of J_i in \mathcal{S} is $S_i(\mathcal{S})/p_i$. 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 only knows a lower bound on p_i equal to the length of time that has run p_i .

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 with 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 $WF^p = \sum_i (w_i F_i^p)$. We use Opt to denote the optimal schedule for the objective function under consideration. So $F^p(Opt_s(\mathcal{I}))$ denotes the the optimal schedule for the F^p measure on \mathcal{I} with a speed s processor, as well as the value of the F^p objective function on this schedule.

A job J_i is alive in a schedule \mathcal{S} at time t , if $r_i \leq t \leq C_i(\mathcal{S})$. We use $U(\mathcal{S}, t)$ to denote the collection of alive jobs at time t in the schedule \mathcal{S} . Let $V(t, \alpha)$ denote the aggregate unfinished volume at time t among those jobs J_j that satisfy the conditions in the list α . So for example, later we will consider $V(r_i, J_j \in U(Opt_1, r_i), r_j \leq r_i, p_j \leq p_i)$, which is the amount of volume that Opt_1 has left on jobs J_j that arrived before J_i , are smaller in size than J_i and that Opt_1 has not finished by time r_i . Let $P(\alpha)$ will denote the size of the jobs J_j that satisfy the conditions in the list α . Note for simplicity of notation, we always implicitly use j as the local index to the jobs being described

in this notation.

For a schedule \mathcal{S} , we use $\text{Age}^p(\mathcal{S}, t)$ denote the sum over all jobs $J_i \in U(\mathcal{S}, t)$ of $(t - r_i)^{p-1}$. Note that $F^p(\mathcal{S}) = p \int_t \text{Age}^p(\mathcal{S}, t) dt$.

An oblivious adversary must specify the complete input before an online algorithm begins executions [10].

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 [10] 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 0 to time 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 $F^p(Opt)$. Consider first the case that A doesn't 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 $3L^\alpha/2$ onwards 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 flowtime.

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 $S^p(Opt)$. Consider first the case that A doesn't 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 jobs and then process the unit jobs. Hence, $S^p(\text{Opt}) = O(1 + L^\alpha \cdot L^p) = O(L^{\alpha+p})$. 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 $S^p(\text{Opt}) = O(L^\alpha \cdot 1^p + L^{\alpha+\beta} \cdot 1^p + (L^\alpha/L)^p)$. Algebraic simplification shows that $\alpha + \beta \leq \alpha p - p$ for our choice of α and β . Hence the dominant term is $L^{\alpha p - p}$, and $S^p(\text{Opt}) = O(L^{\alpha p - p})$. 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 $3L^\alpha/2$ onwards 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 once 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 input instance from [22]. Consider n jobs of sizes $1, 2, 4, 8, \dots, B = 2^n$ arriving at time 0. By scheduling the jobs from shortest to longest, $S^p(\text{Opt}) = O(n)$. While for any non-clairvoyant algorithm A , $S^p(A, s) = \Omega(n^{p+1}/s^p)$.

5 Analysis of SJF

In this section we show that SJF is a $(1 + \epsilon)$ -speed $O(1/\epsilon)$ -competitive for ℓ_p norms of flow and stretch. Recall that SJF always runs a job of minimal size. We will show that the competitiveness of SJF will follow from the existence of what we call an *association scheme*, which informally is an injection from the jobs unfinished by SJF to the volume of jobs unfinished by Opt . We first define our association scheme. In Lemma 3 we show that an association scheme always exists. Finally, in Theorem 4 we show that the competitiveness of SJF follows directly from the existence of this association scheme.

SJF Association Scheme: Fix some time t . Let us use $\mathcal{D}(t)$ to denote $U(SJF, t) - U(\text{Opt}, t)$. Let $1, \dots, k$ denote the indices of jobs in $\mathcal{D}(t)$ such that $p_1 \leq p_2 \leq \dots \leq p_k$. Index the jobs in $U(\text{Opt}, t)$ in some arbitrary order. Consider the jobs in $\mathcal{D}(t)$ in the order in which they are indexed. Assume that we are considering job J_i . Let t' denote the time $t - \frac{\epsilon}{4(1+\epsilon)}(t - r_i)$. Our association scheme for time t associates with J_i an $\epsilon p_i/4(1 + \epsilon)$ amount of volume V_i from $V(t, J_j \in U(\text{Opt}, t), r_j \leq t', p_j \leq p_i)$. None of the volume in V_i can have been previously associated to a lower indexed job in $\mathcal{D}(t)$. The volume in W_i is always preferentially taken from the lowest indexed jobs in $U(\text{Opt}, t)$.

The following lemma will be useful in proving the existence of the association scheme.

Lemma 2 For all times u and for all values p_i ,

$$V(u, J_j \in U(\text{Opt}, u), r_j \leq u, p_j \leq p_i) \geq \frac{\epsilon}{1+\epsilon} P(J_j \in U(\text{SJF}, u), p_j \leq p_i) + \frac{1}{1+\epsilon} V(u, J_j \in U(\text{SJF}, u), p_j \leq p_i)$$

Proof: The proof is by induction on the time u . First consider the case that SJF is running a job. If there is a job $J_j \in U(\text{SJF}, u)$ with $p_j \leq p_i$, then the righthand side decreases at least as fast as the lefthand side since SJF has a $(1+\epsilon)$ -speed processor. If there is no such job J_j , then the righthand side is zero. In the case that a job J_j , with $p_j \leq p_i$, arrives, both sides increase by p_j . In the case that Opt finishes a job, the lefthand side is continuous. In the case that SJF finishes a job, the righthand side does not increase. \square

Lemma 3 The SJF association scheme always exists. That is, there is always sufficient volume in $V(t, J_j \in U(\text{Opt}, t), r_j \leq t', p_j \leq p_i)$ to assign to each $J_i \in \mathcal{D}$.

Proof: Fix a time t . For notational simplicity let $\mathcal{D} = \mathcal{D}(t)$. Consider a fixed job $J_i \in \mathcal{D}$. The association scheme will not fail on job J_i if, the amount of volume in jobs of size $\leq p_i$ that Opt had left at time r_i , plus the volume of jobs of size $\leq p_i$ that arrived during $(r_i, t']$, minus the work that Opt did during $(r_i, t]$, minus the volume that is allocated to J_1, \dots, J_i is nonnegative. By the definition of the association scheme, the amount of volume in $V(t, J_j \in U(\text{Opt}, t))$ that is associated to one of the jobs J_1, \dots, J_i is at most $\frac{\epsilon}{4(1+\epsilon)} \cdot P(J_j \in \mathcal{D}, p_j \leq p_i)$. Opt can finish at most $(t - r_i)$ volume during time $(r_i, t]$. Therefore the association scheme will not fail on J_i if:

$$V(r_i, J_j \in U(\text{Opt}, r_i), r_j \leq r_i, p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) - (t - r_i) - \frac{\epsilon}{4(1+\epsilon)} P(J_j \in \mathcal{D}, p_j \leq p_i) \geq 0 \quad (1)$$

Now using the fact that

$$\frac{\epsilon}{4(1+\epsilon)} P(J_j \in \mathcal{D}, p_j \leq p_i) = \frac{\epsilon}{4(1+\epsilon)} P(J_j \in \mathcal{D}, r_j \leq r_i, p_j \leq p_i) + \frac{\epsilon}{4(1+\epsilon)} P(J_j \in \mathcal{D}, r_j > r_i, p_j \leq p_i) \quad (2)$$

to prove equation 1 it is sufficient to prove

$$V(r_i, J_j \in U(\text{Opt}, r_i), r_j \leq r_i, p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) - \frac{\epsilon}{4(1+\epsilon)} P(J_j \in \mathcal{D}, r_j \leq r_i, p_j \leq p_i) - \frac{\epsilon}{4(1+\epsilon)} P(J_j \in \mathcal{D}, r_j > r_i, p_j \leq p_i) \geq (t - r_i) \quad (3)$$

Since Opt had to finish all jobs in \mathcal{D} , that are released after r_i , by time t , it must be the case that:

$$(t - r_i) \geq P(J_j \in \mathcal{D}, r_j > r_i, p_j \leq p_i) \quad (4)$$

Then by substitution, to prove equation 3 it suffices to prove:

$$V(r_i, J_j \in U(\text{Opt}, r_i), r_j \leq r_i, p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) - \frac{\epsilon}{4(1+\epsilon)} P(J_j \in \mathcal{D}, r_j \leq r_i, p_j \leq p_i) \geq \left(\frac{4+5\epsilon}{4(1+\epsilon)} \right) (t - r_i) \quad (5)$$

We now concentrate on replacing the term $V(r_i, J_j \in U(\text{Opt}, r_i), r_j \leq r_i, p_j \leq p_i)$ in equation 5. This is where the $1 + \epsilon$ speedup is crucially used. By Lemma 2, with $u = r_i$,

$$V(r_i, J_j \in U(\text{Opt}, r_i), r_j \leq r_i, p_j \leq p_i) \geq \frac{\epsilon}{1 + \epsilon} P(J_j \in U(\text{SJF}, r_i), p_j \leq p_i) + \frac{1}{1 + \epsilon} V(r_i, J_j \in U(\text{SJF}, r_i), p_j \leq p_i) \quad (6)$$

To prove equation 5 it is then sufficient by substitution using equation 6 to prove:

$$\begin{aligned} & \frac{\epsilon}{1 + \epsilon} P(J_j \in U(\text{SJF}, r_i), p_j \leq p_i) + \frac{1}{1 + \epsilon} V(r_i, J_j \in U(\text{SJF}, r_i), p_j \leq p_i) \\ & + P(r_j \in (r_i, t'], p_j \leq p_i) - \frac{\epsilon}{4(1 + \epsilon)} P(J_j \in \mathcal{D}, r_j \leq r_i, p_j \leq p_i) \geq \left(\frac{4 + 5\epsilon}{4(1 + \epsilon)} \right) (t - r_i) \end{aligned} \quad (7)$$

Since

$$\begin{aligned} P(J_j \in U(\text{SJF}, r_i), p_j \leq p_i) & \geq P(J_j \in U(\text{SJF}, t), r_j \leq r_i, p_j \leq p_i) \\ & \geq P(J_j \in \mathcal{D}, r_j \leq r_i, p_j \leq p_i) \end{aligned}$$

to prove equation 7 it suffices to prove

$$\begin{aligned} & \frac{3\epsilon}{4(1 + \epsilon)} P(J_j \in U(\text{SJF}, r_i), p_j \leq p_i) + \frac{1}{1 + \epsilon} V(r_i, J_j \in U(\text{SJF}, r_i), p_j \leq p_i) \\ & + P(r_j \in (r_i, t'], p_j \leq p_i) \geq \left(\frac{4 + 5\epsilon}{4(1 + \epsilon)} \right) (t - r_i) \end{aligned} \quad (8)$$

Since $V(r_i, J_j \in U(\text{SJF}, r_i), p_j \leq p_i) \leq P(J_j \in U(\text{SJF}, r_i), p_j \leq p_i)$, to prove equation 8 it suffices by substitution to show that

$$\frac{4 + 3\epsilon}{4(1 + \epsilon)} V(r_i, J_j \in U(\text{SJF}, r_i), p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) \geq \left(\frac{4 + 5\epsilon}{4(1 + \epsilon)} \right) (t - r_i) \quad (9)$$

During $[r_i, t']$, *SJF* does exactly $(1 + \epsilon)(t' - r_i)$ work on jobs with size at most p_i . The jobs that *SJF* worked on during $[r_i, t']$ either had to arrive before r_i or during $[r_i, t']$. Therefore, it trivially follows that:

$$(1 + \epsilon)(t' - r_i) \leq V(r_i, J_j \in U(\text{SJF}, r_i), p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) \quad (10)$$

By the definition of t' and algebraic simplification, $(1 + \epsilon)(t' - r_i) = \frac{4 + 3\epsilon}{4}(t - r_i)$. Therefore equation 10 is equivalent to:

$$\begin{aligned} (t - r_i) & \leq \frac{4}{4 + 3\epsilon} V(r_i, J_j \in U(\text{SJF}, r_i), p_j \leq p_i) \\ & + \frac{4}{4 + 3\epsilon} P(r_j \in (r_i, t'], p_j \leq p_i) \end{aligned} \quad (11)$$

To prove equation 9 it suffices, by substitution using equation 11, to prove:

$$\begin{aligned} & \frac{4 + 3\epsilon}{4(1 + \epsilon)} V(r_i, J_j \in U(\text{SJF}, r_i), p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) \geq \\ & \frac{4 + 5\epsilon}{(4 + 3\epsilon)(1 + \epsilon)} V(r_i, J_j \in U(\text{SJF}, r_i), p_j \leq p_i) + \frac{4 + 5\epsilon}{(4 + 3\epsilon)(1 + \epsilon)} P(r_j \in (r_i, t'], p_j \leq p_i) \end{aligned} \quad (12)$$

Now, it is easy to see that equation 12 is true since,

$$1 \geq \frac{4 + 5\epsilon}{(4 + 3\epsilon)(1 + \epsilon)}$$

and

$$\frac{4 + 3\epsilon}{4(1 + \epsilon)} \geq \frac{4 + 5\epsilon}{(4 + 3\epsilon)(1 + \epsilon)}$$

□

Theorem 4 *SJF is $(1 + \epsilon)$ -speed $O(\frac{1}{\epsilon})$ -competitive for the ℓ_p norms of flowtime and stretch, for $p \geq 1$.*

Proof: We show that this follows from the existence of the *SJF* association scheme.

Let us first consider flow. Consider an arbitrary time t . Let J_j be an arbitrary job in $U(\text{Opt}, t)$. By the definition of the association scheme, the unfinished volume in J_j is associated with a (possibly empty) collection $\mathcal{D}_j \subseteq \mathcal{D}$. If \mathcal{D}_j is not empty, let J_i be a job in \mathcal{D}_j .

By the definition of the association scheme, job J_j is $\Omega(\epsilon)$ as old as J_i . Hence, the contribution of J_i to $\text{Age}^p(U(\text{SJF}, t), t)$ is $O(1/\epsilon^{p-1})$ times the contribution of J_j to $\text{Age}^p(U(\text{Opt}, t), t)$. By the definition of the association scheme, job J_i is associated with only jobs $J_k \in U(\text{Opt}, t)$ of smaller size. Hence by the definition of the association scheme, J_i is associated with at most $O(1/\epsilon)$ jobs in $U(\text{Opt}, t)$. Hence, combining the above two observations gives us that

$$\text{Age}^p(U(\text{SJF}, t), t) = O\left(\frac{1}{\epsilon^p}\right) \cdot \text{Age}^p(U(\text{Opt}, t), t)$$

Since for an algorithm A , $F^p(A) = p \int_t \text{Age}^p(U(A, t), t) dt$, $F^p(\text{SJF}) = O(1/\epsilon^p) \cdot F^p(\text{Opt})$. By taking the p^{th} root, the theorem for flow follows.

The argument for stretch is identical except that $S\text{Age}^p(X, t)$, which is the sum over all jobs $J_i \in U(X, t)$ of $(t - r_i)^{p-1}/p_i^p$, is used in place of $\text{Age}^p(X, t)$. □

6 Analysis of SRPT

In this section we show that *SRPT* is a $(1 + \epsilon)$ -speed $O(1/\epsilon)$ -competitive for ℓ_p norms of flow and stretch. Recall that *SRPT* always runs a job with the least remaining unfinished volume. The overall structure of the analysis of *SRPT* is similar to that of *SJF*, but the analysis of is somewhat more involved because we need to handle the remaining volume of *SRPT* more carefully.

SRPT Association Scheme: The scheme is identical to the *SJF* association scheme except that $\mathcal{D}(t)$ consists of only those jobs $J_i \in U(\text{SRPT}, t) - U(\text{Opt}, t)$ with release time and volume that satisfy $(t - r_i) \geq 8p_i/\epsilon$. That is, the jobs in \mathcal{D} have to be sufficiently old relative to their size.

We begin with a lemma analogous to Lemma 2.

Lemma 5 For all times u and values of p_i ,

$$V(u, J_i \in U(\text{Opt}, u), r_j \leq u, p_j \leq p_i) \geq \frac{\epsilon}{1+\epsilon} P(J_j \in U(\text{SRPT}, u), p_j \leq p_i) + \frac{1}{1+\epsilon} V(u, J_j \in U(\text{SRPT}, u), p_j \leq p_i) - \frac{1}{1+\epsilon} p_i$$

Proof: The proof is by induction on the time u . If SRPT is running a job with size $\leq p_i$, then the righthand side of the inequality decreases at least as fast as the lefthand side since SRPT has a $(1+\epsilon)$ speed processor. If SRPT is running a job with remaining volume $> p_i$, then the righthand side of the inequality is $-p_i$ and the lefthand side is nonnegative.

Now consider the case that SRPT is running a job J_l with size $> p_i$ and remaining volume $\leq p_i$. Let J_k , $k \neq l$ be a job in $U(\text{SRPT}, u)$ with $p_k \leq p_i$. It must be the case that $p_k(u) \geq p_l(u)$ since J_l is being run at time u . Since $p_k \leq p_i < p_l$, it must be the case that $r_k > r_l$, or SRPT would have preferred J_k to J_l at time r_l , and later, contradicting the definition of J_l . Similarly, it must be the case that $p_k(u) = p_k$ because whichever of J_j and J_k that had the least remaining volume at time r_k would also have the least remaining volume throughout the interval $[r_k, u]$. Therefore

$$P(J_j \in U(\text{SRPT}, u), p_j \leq p_i) = V(u, J_j \in U(\text{SRPT}, u), p_j \leq p_i) = \frac{\epsilon}{1+\epsilon} P(J_j \in U(\text{SRPT}, u), p_j \leq p_i) + \frac{1}{1+\epsilon} V(u, J_j \in U(\text{SRPT}, u), p_j \leq p_i) \quad (13)$$

Among those jobs in $U(\text{SRPT}, u)$ with $p_j \leq p_i$, let J_k be the one with the earliest release time. And $V(u, J_i \in U(\text{Opt}, u), r_j \leq u, p_j \leq p_i)$ is at least $P(J_j \in U(\text{SRPT}, u), p_j \leq p_i)$ minus the quantity of the time during $[r_k, u]$ that SRPT ran J_i over $(1+\epsilon)$. This is how much volume Opt would have left if it gave top priority to those jobs in $U(\text{SRPT}, u)$ with $p_j \leq p_i$ since time r_k . The division by $(1+\epsilon)$ is because Opt has a $(1+\epsilon)$ times slower processor. The results then follows because SRPT processed J_j since time r_k for at most $p_k \leq p_i$ time units.

The case of job arrivals, and SRPT or Opt finishing a job are handled as in the proof of Lemma 2. \square

Lemma 6 The SRPT association scheme always exists. That is, there is always sufficient volume in $V(t, J_j \in U(\text{Opt}, t), r_j \leq t', p_j \leq p_i)$ to assign to each $J_i \in \mathcal{D}(t)$.

Proof: Fix a time t , and again let $\mathcal{D} = \mathcal{D}(t)$. Consider a fixed job $J_i \in \mathcal{D}$. Analogously to equation 1 in the analysis of SJF , the association scheme will not fail on job J_i if

$$V(r_i, J_j \in U(\text{Opt}, r_i), r_j \leq r_i, p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) - (t - r_i) - \frac{\epsilon}{4(1+\epsilon)} P(J_j \in \mathcal{D}, p_j \leq p_i) \geq 0 \quad (14)$$

Following the same line of reasoning as in the analysis of SJF , to prove this equation it is sufficient to establish the following equation, which is equivalent to equation 5 in the analysis of SJF :

$$V(r_i, J_j \in U(\text{Opt}, r_i), r_j \leq r_i, p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) - \frac{\epsilon}{4(1+\epsilon)} P(J_j \in \mathcal{D}, r_j \leq r_i, p_j \leq p_i) \geq \left(\frac{4+5\epsilon}{4(1+\epsilon)} \right) (t - r_i) \quad (15)$$

By Lemma 5, with $u = r_i$, it is the case that:

$$V(r_i, J_i \in U(\text{Opt}, r_i), r_j \leq r_i, p_j \leq p_i) \geq \frac{\epsilon}{1+\epsilon} P(J_j \in U(\text{SRPT}, r_i), p_j \leq p_i) + \frac{1}{1+\epsilon} V(r_i, J_j \in U(\text{SRPT}, r_i), p_j \leq p_i) - \frac{1}{1+\epsilon} p_i \quad (16)$$

Now by substitution using equation 16, to prove equation 15 it is sufficient to prove:

$$\begin{aligned} & \frac{\epsilon}{1+\epsilon} P(J_j \in U(\text{SRPT}, r_i), p_j \leq p_i) + \frac{1}{1+\epsilon} V(r_i, J_j \in U(\text{SRPT}, r_i), p_j \leq p_i) \\ & + P(r_j \in (r_i, t'], p_j \leq p_i) - \frac{\epsilon}{4(1+\epsilon)} P(J_j \in \mathcal{D}, r_j \leq r_i, p_j \leq p_i) - \frac{1}{1+\epsilon} p_i \\ & \geq \left(\frac{4+5\epsilon}{4(1+\epsilon)} \right) (t - r_i) \end{aligned} \quad (17)$$

Continuing on as in the analysis of *SJF*, using the obvious facts that

$$P(J_j \in U(\text{SRPT}, r_i), p_j \leq p_i) \geq P(J_j \in \mathcal{D}, r_j \leq r_i, p_j \leq p_i) \quad (18)$$

and

$$P(J_j \in U(\text{SRPT}, r_i), p_j \leq p_i) \geq V(r_i, J_j \in U(\text{SRPT}, r_i), p_j \leq p_i) \quad (19)$$

it suffices to show that

$$\begin{aligned} & \frac{4+3\epsilon}{4(1+\epsilon)} V(J_j \in U(\text{SRPT}, r_i), p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) - \frac{1}{1+\epsilon} p_i \\ & \geq \left(\frac{4+5\epsilon}{4(1+\epsilon)} \right) (t - r_i) \end{aligned} \quad (20)$$

Note the similarity of equation 9 in the analysis of *SJF* and equation 20.

We now take a bit of a detour to compute an upper bound on the term $(t - r_i)$. Since *SRPT* did not finish J_i by time t , *SRPT* did not finish J_i by time t' , and during the time period $(r_i, t']$ it must have been the case that *SRPT* was running only jobs with remaining volume at most p_i . Since *SRPT* has a $(1 + \epsilon)$ speed processor:

$$(1 + \epsilon)(t' - r_i) \leq V(r_i, J_j \in U(\text{SRPT}, r_i), p_j(r_i) \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) \quad (21)$$

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

$$V(r_i, J_j \in U(\text{SRPT}, r_i), p_j(r_i) \leq p_i) \leq V(r_i, J_j \in U(\text{SRPT}, r_i), p_j \leq p_i) + p_i \quad (22)$$

Combining equation 21 and equation 22 we get that

$$(1 + \epsilon)(t' - r_i) \leq V(r_i, J_j \in U(\text{SRPT}, r_i), p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) + p_i \quad (23)$$

By the definition of t' and algebraic simplification, $(1 + \epsilon)(t' - r_i) = \frac{4+3\epsilon}{4}(t - r_i)$. Therefore, we get that

$$\frac{4+3\epsilon}{4}(t - r_i) \leq V(r_i, J_j \in U(\text{SRPT}, r_i), p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) + p_i \quad (24)$$

Since $J_i \in \mathcal{D}$, the age $(t - r_i)$ of J_i is at least $\frac{8p_i}{\epsilon}$, or equivalently $p_i \leq \frac{\epsilon}{8}(t - r_i)$. By subtracting $\frac{2\epsilon}{8}(t - r_i)$ from the lefthand side of equation 24 and subtracting $2p_i$ from the righthand side of equation 24 we get that

$$\frac{4+2\epsilon}{4}(t - r_i) \leq V(r_i, J_j \in U(\text{SRPT}, r_i), p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) - p_i \quad (25)$$

or equivalently,

$$(t - r_i) \leq \frac{2}{2 + \epsilon} (V(r_i, J_j \in U(SRPT, r_i), p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) - p_i) \quad (26)$$

Using equation 26, to prove equation 20 it suffices by substitution to show that

$$\begin{aligned} \frac{4 + 3\epsilon}{4(1 + \epsilon)} V(J_j \in U(SRPT, r_i), p_j \leq p_i) + P(r_j \in (r_i, t'], p_j \leq p_i) - \frac{1}{1 + \epsilon} p_i \geq \\ \left(\frac{4 + 5\epsilon}{4(1 + \epsilon)} \right) \left(\frac{2}{2 + \epsilon} \right) V(r_i, J_j \in U(SRPT, r_i), p_j \leq p_i) \\ + \left(\frac{4 + 5\epsilon}{4(1 + \epsilon)} \right) \left(\frac{2}{2 + \epsilon} \right) P(r_j \in (r_i, t'], p_j \leq p_i) - \frac{2}{2 + \epsilon} p_i \end{aligned} \quad (27)$$

It is easy to see that equation 27 always holds by comparing term by term, that is

$$\frac{4 + 3\epsilon}{4(1 + \epsilon)} \geq \left(\frac{4 + 5\epsilon}{4(1 + \epsilon)} \right) \left(\frac{2}{2 + \epsilon} \right) \quad (28)$$

and

$$1 \geq \left(\frac{4 + 5\epsilon}{4(1 + \epsilon)} \right) \left(\frac{2}{2 + \epsilon} \right) \quad (29)$$

and $\frac{1}{1 + \epsilon} \leq \frac{2}{2 + \epsilon}$. \square

Theorem 7 *SRPT is $(1 + \epsilon)$ -speed $O(\frac{1}{\epsilon})$ -competitive for the ℓ_p norms of flowtime and stretch, for $p \geq 1$.*

Proof: We show that this follows from the existence of the *SRPT* association scheme. Let us first consider flow.

$$F^p(SRPT) = p \int_t \text{Age}(U(SRPT, t), t) \quad (30)$$

$$= O \left(\int_t \text{Age}(\mathcal{D}(t), t) + \int_t \text{Age}(U(Opt, t), t) + \sum_{i=1}^n \left(\frac{8p_i}{\epsilon} \right)^p \right) \quad (31)$$

$$= O \left(\int_t \text{Age}(\mathcal{D}(t), t) + F^p(Opt) + \sum_{i=1}^n \left(\frac{8p_i}{\epsilon} \right)^p \right) \quad (32)$$

$$= O \left(\int_t \text{Age}(\mathcal{D}(t), t) + F^p(Opt) + \frac{1}{\epsilon^p} F^p(Opt) \right) \quad (33)$$

$$= O \left(\int_t \text{Age}(\mathcal{D}(t), t) + \frac{1}{\epsilon^p} F^p(Opt) \right) \quad (34)$$

$$= O \left(\frac{1}{\epsilon^p} F^p(Opt) \right) \quad (35)$$

Line 30 is from the definition of F^p . Line 31 follows from the definition of $\mathcal{D}(t)$. Line 32 follows from the definition of $F^p(Opt)$. Line 33 follows since the flow time of J_i is at least p_i in *Opt*. Line 35 follows by using exactly the same reasoning as in the proof of Theorem 4 that showed that $\int_t \text{Age}(\mathcal{D}(t), t) = O(\frac{1}{\epsilon^p} F^p(Opt))$.

The argument for stretch is similar. \square

7 Analysis of SETF

We first show that *SETF* is $(1 + \epsilon)$ -speed $O(1/\epsilon^{2+2/p})$ -competitive for the ℓ_p norm of flowtime. We then show that *SETF* is $(1 + \epsilon)$ -speed poly-log-competitive for the ℓ_p norm of stretch. Recall *SETF* always runs a job J_j that has been run the least, that is for which $x_j(t)$ is minimal. We first compare *SETF* to an intermediate policy *MLF*, and then compare *MLF* to *Opt*. The idea of analyzing *SETF* to this intermediate policy *MLF* was also used in [5].

Our *MLF* is a variation of the standard Multilevel Feedback Queue algorithm [15, 33], 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 $q_i = \ell_{i+1} - \ell_i = \epsilon^2(1 + \epsilon)^i$, $i \geq 1$. 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.

Lemma 8 *For any instance \mathcal{I} , and for any $J_j \in \mathcal{I}$, 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 units. Let $U_{\leq z}(A, t)$ denote the set of unfinished jobs in algorithm A at time t which 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 $V_{\leq x}(SETF_s, t) \leq V_{\leq x}(A_s, t)$ for all algorithms A and all speeds s .

Suppose to reach a contradiction 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 $U(MLF_1, t)$ have at least $p_j/(1 + \epsilon)$ amount of work done on them, otherwise J_j wouldn't be run by MLF_1 at time $C_j(MLF_1)$ since there would be a lower level job than J_j at this time. 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 than 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. \square

Lemma 8 implies that

$$F^p(SETF(\mathcal{I}), s) \leq F^p(MLF(\mathcal{I}), s/(1 + \epsilon)) \quad (36)$$

Now our goal will be to relate *MLF* and *Opt*. 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 in this transformation from \mathcal{I} to \mathcal{K} , the size of a job doesn't decrease, and it increases by at most a factor of $(1 + \epsilon)^2$. 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

$$F^p(MLF(\mathcal{I}), s/(1 + \epsilon)) \leq F^p(MLF(\mathcal{K}), s/(1 + \epsilon)) \quad (37)$$

Finally we create an instance \mathcal{L} by replacing each job of size $\epsilon((1 + \epsilon)^k - 1)$ job in \mathcal{K} by k jobs of size q_1, \dots, q_{k-1} . Note that $\ell_k = q_0 + q_1 + \dots, q_{k-1}$. For a job of $J_j \in \mathcal{K}$, we denote the corresponding jobs in \mathcal{L} by $J_{j_0}, J_{j_1}, \dots, J_{j_{k-1}}$.

Notice that 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,

$$F^p(MLF(\mathcal{K}), s/(1+\epsilon)) \leq F^p(SJF(\mathcal{L}), s/(1+\epsilon)) \quad (38)$$

By Theorem 4, we know that

$$F^p(SJF(\mathcal{L}), s/(1+\epsilon)) = O(1/\epsilon^p)F^p(Opt(\mathcal{L}), s/(1+\epsilon)^2) \quad (39)$$

We relate the optimal schedule for \mathcal{L} back to the optimal schedule for \mathcal{J} . 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 $\mathcal{L} = \mathcal{L}(1) \cup \mathcal{L}(2) \cup \dots$. 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 $J_{j_{i-1}}$ 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 $J_{j_{i-2}}$ of size q_{i-2} and so on. Thus, \mathcal{L} is exactly $\mathcal{L}(1) \cup \mathcal{L}(2) \cup \dots$. Summarizing, we have that the $\mathcal{L}(k)$'s are geometrically scaled down copies of \mathcal{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 to obtain a good schedule for \mathcal{L} given a good schedule for \mathcal{J} . To put everything in place, we need the following lemma from [5] which relates \mathcal{J} and $\mathcal{L}(k)$.

Lemma 9 *Let \mathcal{J} and $\mathcal{L}(k)$ be as defined above. For all $x \geq 1$,*

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

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

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

Consider some instance where this condition is violated. Let J_j be the job and t be the earliest time for which $y_j((t-r_j)/s, s) < y_j(t-r_j, 1)$. Clearly, SJF_s is not working on J_j at time t , due to minimality of t . Thus, SJF_s is working on some other smaller job J_i . Since SJF_1 is not working on job J_i , SJF_1 has already finished J_i by some time $t' < t$. However, this means that $y_i((t'-r_i), s) < y_i(t'-r_i, 1)$, which contradicts the minimality of t .

We now show the main result of the lemma. It is easy to see that the flow time of every job $J_j(k) \in \mathcal{L}(k)$ (where $J_j(k)$ is job corresponding to $J_j \in \mathcal{J}$ but scaled down by $\epsilon(1+\epsilon)^{-k}$ times) 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 fact above, 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 size 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. \square

Lemma 10

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

Proof: Given the schedule $\text{SJF}(\mathcal{J})$, we construct a following schedule A for \mathcal{L} as follows. The jobs in $\mathcal{L}(k)$ are run with a speed $s_k = \epsilon(1 + \frac{\epsilon}{1+\epsilon})^{-k}$ processor using the algorithm SJF . 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 9, $F^p(A(\mathcal{L}(k)), 1 + 2\epsilon)$ will be at most $\left(\frac{\epsilon(1+\epsilon)^{-k}}{x_k}\right)^p = \left(\frac{(1+2\epsilon)}{(1+\epsilon)^2}\right)^{kp}$ times $F^p(\text{SJF}(\mathcal{J}), 1)$. Hence,

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

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

Hence by Lemma 10, Theorem 4, and the fact that that jobs lengths in \mathcal{J} are at most $(1 + \epsilon)^2$ times as long as they are in \mathcal{I} , we get that

$$\begin{aligned} F^p(\text{Opt}(\mathcal{L}), s/(1 + \epsilon)^2) &= O(1/\epsilon^2) \cdot F^p\left(\text{SJF}(\mathcal{J}), \frac{s}{(1 + 2\epsilon)(1 + \epsilon)^2}\right) \\ &= O(1/\epsilon^{p+2}) \cdot F^p\left(\text{Opt}(\mathcal{J}), \frac{s}{(1 + 2\epsilon)(1 + \epsilon)^3}\right) \\ &= O(1/\epsilon^{p+2}) \cdot F^p\left(\text{Opt}(\mathcal{I}), \frac{s}{(1 + 2\epsilon)(1 + \epsilon)^5}\right) \end{aligned} \quad (40)$$

By stringing together the inequalities 36, 37, 38, 39, and 40, we find that

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

Hence, we conclude with the main theorem for the flow analysis of SETF .

Theorem 11 *SETF is $(1 + \epsilon)$ -speed $O(1/\epsilon^{2+2/p})$ -competitive for the ℓ_p norm of flowtime.*

We now turn our attention to the stretch analysis of SETF . The analysis is similar to the analysis of flow. By Lemma 8, it follows that

$$S^p(\text{SETF}(\mathcal{I}), 1 + \epsilon) \leq S^p(\text{MLF}(\mathcal{I}), 1) \quad (41)$$

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

$$S^p(\text{MLF}(\mathcal{I}), 1) \leq (1 + \epsilon)^{2p} S^p(\text{MLF}(\mathcal{K}), 1) \quad (42)$$

Combining equations 36 and 37 we get that

$$S^p(\text{SETF}(\mathcal{I}), 1 + \epsilon) \leq (1 + \epsilon)^{2p} S^p(\text{MLF}(\mathcal{K}), 1) \quad (43)$$

For a job of size $\epsilon[(1+\epsilon)^k - 1]$ in \mathcal{K} , the corresponding job of size $\epsilon^2(1+\epsilon)^k \in \mathcal{L}$ has equal flow time. Thus the ratio of the contributions to the objective S^p for \mathcal{L} and \mathcal{K} will be at least $(\frac{\epsilon^2(1+\epsilon)^k}{\epsilon[(1+\epsilon)^k - 1]})^p$, which is at least $(\frac{\epsilon}{1-(1+\epsilon)^{-k}})^p$. Now since $\epsilon[(1+\epsilon)^k - 1] \geq 1$ for all valid job sizes in \mathcal{K} , we get that $(1+\epsilon)^{-k} \leq \frac{\epsilon}{1+\epsilon}$ and hence that $(\frac{\epsilon}{1-(1+\epsilon)^{-k}})^p \leq (\epsilon(1+\epsilon))^p$. Thus we get that

$$S^p(\text{MLF}(\mathcal{K}), 1) \leq (\epsilon(1+\epsilon))^p S^p(\text{SJF}(\mathcal{L}), 1) \quad (44)$$

Now, applying Theorem 4 it follows that

$$S^p(\text{SJF}(\mathcal{L}), 1 + \epsilon) = O(1/\epsilon^p) S^p(\text{Opt}(\mathcal{L}), 1) \quad (45)$$

Combining equations 44 and 45 we get that

$$S^p(\text{MLF}(\mathcal{K}), 1) = O((1+\epsilon)^p) S^p(\text{SJF}(\mathcal{L}), 1) \quad (46)$$

Recall from Theorem 4 that

$$S^p(\text{SJF}(\mathcal{L}), 1 + \epsilon) = O(1/\epsilon^p) S^p(\text{Opt}(\mathcal{L}), 1) \quad (47)$$

We now prove a result similar to Lemma 10 for stretch.

Lemma 12

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

Proof: Set $s_i = \epsilon(1+\epsilon)^{-i}$ for $i = 1, \dots, \log_{1+\epsilon} \log_{1+\epsilon}(B/\epsilon)$, and $s_i = \epsilon/\log_{1+\epsilon}(B/\epsilon)$ for $\log_{1+\epsilon} \log_{1+\epsilon}(B/\epsilon) < i \leq L$. We run the jobs in $\mathcal{L}(i)$ using SJF on a speed s_i processor. Notice that the total speedup required is $\sum_{i=1}^L s_i$ which is at most $1 + \epsilon$.

By lemma 9, $S^p(\text{SJF}(\mathcal{L}(i)), s_i)$ is at most $(1/s_i^p) S^p(\text{SJF}(\mathcal{J}), 1)$. By simple algebraic calculation it can be seen that $\sum_i (\frac{1}{s_i})^p$ is $O(\frac{1}{\epsilon^p} L^{p+1})$. \square

Combining equations 43, 46, and 47, and Lemma 12, we get that

$$S^p(\text{SETF}(\mathcal{I}), (1 + \epsilon)^2) = O\left(\frac{(1 + \epsilon)^{3p}}{\epsilon^p} \cdot \log_{1+\epsilon}^{p+1}\left(\frac{B}{\epsilon}\right)\right) S^p(\text{SJF}(\mathcal{J}), 1) \quad (48)$$

Now by Theorem 4 we have that

$$S^p(\text{SJF}(\mathcal{J}), 1 + \epsilon) = O(1/\epsilon^p) S^p(\text{Opt}(\mathcal{J}), 1) \quad (49)$$

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

$$S^p(\text{Opt}(\mathcal{J}), (1 + \epsilon)^2) \leq S^p(\text{Opt}(\mathcal{I}), 1) \quad (50)$$

Now combining equations 48, 49 and 50 it follows that

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

or equivalently that

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

Thus we conclude with the main result of this subsection.

Theorem 13 *SETF is a $(1 + \epsilon)$ -speed $O\left(\frac{1}{\epsilon^{3+1/p}} \cdot \lg^{1+1/p}\left(\frac{B}{\epsilon}\right)\right)$ -competitive algorithm for the ℓ_p norm of stretch.*

8 Lower Bound for Round Robin

In this section we show the perhaps somewhat surprising result that 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 . Recall that Round Robin shares the processor equally among all processes. This lower bound uses a modification of the lower bound instances in [24, 27, 28].

Theorem 14 *For every $p \geq 1$, there is an $\epsilon > 0$ such that Round Robin (RR) is not an $(1 + \epsilon)$ -speed $n^{o(1)}$ -competitive algorithm for the ℓ_p norm of flow time.*

Proof: Suppose $\epsilon < 1/2p$ 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 $p_i = (1 - \frac{1+\epsilon}{i+1})p_{i-1}$ 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 has a flow time equal to its size. The job of size 1 has flow time $t' = 1 + \sum_{j=0}^n p_j$.
- Under RR (with a $1 + \epsilon$ speed processor) all the jobs keep accumulating and finish simultaneously at time $t = (2p_0 + \sum_{j=1}^n p_i)/(1 + \epsilon)$.

We now consider the relevant quantities. First observe that $p_i = \prod_{j=1}^i \frac{(j-\epsilon)}{j+1} = \frac{1}{i+1} \prod_{j=1}^i (1 - \epsilon/j)$. Using the fact that for all $x \geq 0$, $1 - x \leq e^{-x}$ we get that $p_i \leq \frac{1}{i+1} e^{-H(i)\epsilon}$ 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 $F^p(\text{SRPT}, 1)$, and hence $F^p(\text{Opt}, 1)$. Observing that $\sum_{j=0}^n p_j \leq \sum_{j=0}^{\infty} p_j \leq \sum_{j=0}^{\infty} j^{-(1+\epsilon)} = O(1)$ and that $\sum_{j=0}^n p_j^p \leq \sum_{j=0}^{\infty} j^{-(1+\epsilon)p} = O(1)$ we get that $F^p(\text{SRPT}, 1) = O(1)$.

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

9 Analysis of *HDF*

In this section we show that *HDF*, a natural generalization of *SJF* 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. 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 and 1, and release time r_i . We denote these w_i jobs by $J'_{i1}, \dots, J'_{iw_i}$. Let $X_i = \{J'_{i1}, \dots, J'_{iw_i}\}$ denote this collection of jobs obtained from J_i . Note that all jobs in \mathcal{I}' have the same weight.

Lemma 15 *For \mathcal{I} and \mathcal{I}' as defined above,*

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

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 , 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. \square

From Theorem 4 we know that

$$F^p(\text{SJF}(\mathcal{I}'), 1 + \epsilon) = O\left(\frac{1}{\epsilon^p}\right) F^p(\text{Opt}(\mathcal{I}'), 1) \quad (52)$$

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 16

$$WF^p(\text{HDF}(\mathcal{I}), 1 + \epsilon) \leq \left(1 + \frac{1}{\epsilon}\right)^p F^p(\text{SJF}(\mathcal{I}'), 1) \quad (53)$$

Proof: We claim that for every job $J_i \in \mathcal{I}$ and every time t , if J_i is alive at time t in $\text{HDF}_{1+\epsilon}(\mathcal{I})$, then at least $\frac{\epsilon}{1+\epsilon} w_i$ jobs in $X_i \in \mathcal{I}'$ are alive 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 alive in $\text{HDF}_{i+1}(\mathcal{I})$ and there are fewer than $\epsilon / (1 + \epsilon) w_i$ jobs from X_i left in $\text{SJF}_1(\mathcal{I}')$. Since J_i is alive in $\text{HDF}_{i+1}(\mathcal{I})$ and $\text{HDF}_{i+1}(\mathcal{I})$ has a $1 + \epsilon$ faster processor, $\text{HDF}_{i+1}(\mathcal{I})$ 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 a 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 higher density than that of J_i . This implies that jobs in X_j have 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 alive at time t' , this contradicts our assumption of the minimality of t . \square

Theorem 17 *HDF is $(1 + \epsilon)$ -speed, $O(1/\epsilon^2)$ -competitive for minimizing the weighted ℓ_p norms of flow time.*

Proof: By Equations 52 and 53 we have that

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

Combining this with Equation 51 gives us the result. \square

10 Analysis of WSETF

We first describe the algorithm *WSETF*. We then show that *WSETF* is a $1 + \epsilon$ -speed, $O(1/\epsilon^{2+2/p})$ -competitive with respect to the ℓ_p norm of flow. 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, \dots, J_k are the jobs that have the smallest norm, then J_j , for $i = 1, \dots, k$, 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 by a collection $X_i = \{J'_{i1}, \dots, J'_{iw_i}\}$ of w_i identical jobs each of size p_i/w_i , weight 1, and release time r_i . Lemma 18 then relates the schedules $WSETF_s(\mathcal{I})$ and $SETF_s(\mathcal{I}')$ for any speed s processor.

Lemma 18 *At any time t , a job $J_i \in \mathcal{I}$ is alive 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 alive 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 $k = 1, \dots, w_i$ 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. \square

Theorem 19 *WSETF is a $1 + \epsilon$ -speed, $O(1/\epsilon^{2+2/p})$ -competitive non-clairvoyant algorithm for minimizing the weighted ℓ_p norms of flow time.*

Proof: By Lemma 18 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

$$WF^p(WSETF(\mathcal{I}), 1) = F^p(SETF(\mathcal{I}'), 1) \quad (54)$$

By Equation 51 we know that

$$F^p(Opt(\mathcal{I}'), 1) \leq WF^p(Opt(\mathcal{I}), 1) \quad (55)$$

By Theorem 11, we know that

$$F^p(SETF(\mathcal{I}'), (1 + \epsilon)) = O(1/\epsilon^{2p+2})F^p(Opt(\mathcal{I}'), 1) \quad (56)$$

Now, by Equations 54, 56 and 51 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.

References

- [1] N. Alon, Y. Azar, G. Woeginger, and T. Yadid. Approximation schemes for scheduling. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 493–500, 1997.
- [2] A. Avidor, Y. Azar, and J. Sgall. Ancient and new algorithms for load balancing in the l_p norm. *Algorithmica*, 29(3):422–441, 2001.
- [3] B. Awerbuch, Y. Azar, E. Grove, M. Kao, P. Krishnan, and J. Vitter. Load balancing in the l_p norm. In *IEEE Symposium on Foundations of Computer Science*, 1995.
- [4] N. Bansal and K. Dhamdhere. Minimizing weighted flow time. In *Symposium on Discrete Algorithms SODA*, pages 508–516, 2003.
- [5] N. Bansal, K. Dhamdhere, J. Konemann, and A. Sinha. Non-clairvoyant scheduling for minimizing mean slowdown. In *Symposium on Theoretical Aspects of Computer Science*, 2003.
- [6] N. Bansal and M. Harchol-Balter. Analysis of srpt scheduling: Investigating unfairness. In *ACM Sigmetrics*, pages 279–290, 2001.
- [7] L. Becchetti and S. Leonardi. Non-clairvoyant scheduling to minimize the average flow time on single and parallel machines. In *ACM Symposium on Theory of Computing*, pages 94–103, 2001.
- [8] L. Becchetti, S. Leonardi, A. M. Spaccamela, and K. Pruhs. Online weighted flow time and deadline scheduling. In *RANDOM-APPROX*, pages 36–47, 2001.

- [9] M. Bender, S. Muthukrishnan, and R. Rajaraman. Improved algorithms for stretch scheduling. In *ACM-SIAM Symposium on Discrete Algorithms*, 2002.
- [10] A. Borodin and R. El-Yaniv. *On-Line Computation and Competitive Analysis*. Cambridge University Press, 1998.
- [11] C. Bussema and E. Torng. Greedy multiprocessor server scheduling algorithms are effective with minimal resource augmentation, 2004.
- [12] C. Chekuri and S. Khanna. Approximation schemes for preemptive weighted flow time. In *ACM Symposium on Theory of Computing*, 2002.
- [13] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for weighted flow time. In *ACM Symposium on Theory of Computing*, 2001.
- [14] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multi-processor scheduling to minimize flow time with ϵ resource augmentation. In *ACM Symposium on Theory of Computing*, pages 363–372, 2004.
- [15] E. Coffman and P. Denning. *Operating Systems Theory*. Prentice Hall, 1973.
- [16] M. Crovella, R. Frangioso, and M. Harchol-Balter. Connection scheduling in web servers. In *USENIX Symposium on Internet Technologies and Systems*, pages 243–254, 1999.
- [17] J. Edmonds. Scheduling in the dark. *Theoretical Computer Science*, 235(1):109–141, 2000.
- [18] L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. In *European Symposium on Algorithms (ESA)*, pages 151–162, 1999.
- [19] M. Harchol-Balter, M. Crovella, and S. Park. The case for srpt scheduling of web servers.
- [20] <http://httpd.apache.org/docs/>.
- [21] <http://www.netcraft.com/survey/>.
- [22] B. Kalyanasundaram and K. Pruhs. Fault-tolerant scheduling. In *ACM Symposium on Theory of Computing*, 1994.
- [23] B. Kalyanasundaram and K. Pruhs. Minimizing flow time nonclairvoyantly. In *IEEE Symposium on Foundations of Computer Science*, pages 345–352, 1997.
- [24] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.
- [25] D. Knuth. *The TeXbook*. Addison Wesley, 1986.
- [26] J. Kurose and K. Ross. *Computer networking: A top-down approach featuring the Internet*. Addison-Wesley, 2002.
- [27] T. Matsumoto. Competitive analysis of the round robin algorithm. In *International Symposium on Algorithms and Computation*, pages 71–77, 1992.
- [28] R. Motwani, S. Phillips, and E. Torng. Nonclairvoyant scheduling. *Theoretical Computer Science*, 130(1):17–47, 1994.
- [29] S. Muthukrishnan, R. Rajaraman, A. Shaheen, and J. Gehrke. Online scheduling to minimize average stretch. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 433–442, 1999.

- [30] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *ACM Symposium on Theory of Computing*, pages 140–149, 1997.
- [31] Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling. In *Handbook on Scheduling*. CRC Press, 2004.
- [32] B. Schroeder and M. Harchol-Balter. Web servers under overload: how scheduling can help.
- [33] A. Tanenbaum. *Operating systems: design and implementation*. Prentice-Hall, 2001.