# Protective Scheduling

Eric J. Friedman*
Gavin Hurley†
School of Operations Research and Industrial Engineering,
Cornell University, Ithaca, NY 14853.

November 7, 2002

**Abstract**

We define and develop a theory of *protective* scheduling algorithms. The work is motivated by webserving where the standard algorithm, Processor Sharing (PS), has come under criticism recently. It often has much worse performance than the well known Shortest Remaining Processing Time (SRPT) algorithm. However, SRPT can "starve" large jobs. A protective scheduling algorithm avoids this; it is one where no job does worse than it would have under PS. The main result of this work is a characterization of all such algorithms. We also introduce a selection of new algorithms which are protective and analyze their performance with respect to average flow time. We also present several bounds on the losses the arise because of protectiveness and on the performance of the various algorithms that we propose.

## 1 Introduction

Currently, when presented with overlapping requests, most webservers use a simple scheduling algorithm, processor sharing (PS) both for job creation and transmission. While PS is natural in many settings, its value for webserving has recently been strongly questioned. Bansal and Harchol-Balter [BHB01] have argued that, since users only care about the completion time of their jobs, one should use a more efficient scheduling algorithm. In particular, they suggest using shortest remaining processing time (SRPT), which is known to minimize average flow time. In [BHB01] they show that SRPT can significantly reduce the average flow time when compares to PS, sometimes by a factor of 60!

However, one strong impediment to using SRPT in web serving is the fact that long jobs can be starved, whereby a series of small jobs can cause unbounded delays for somewhat longer jobs [BCM98, Sta95, Tan92]. For a variety of reasons, one would not want to delay large jobs too much in a quest to reduce average flow times. This is typically expressed by stating that SRPT is not "fair", although we find the term protective more fitting than "fair". Thus, we say that a scheduling algorithm is protective if every job completes earlier under it that it would under PS.

Somewhat surprisingly, [BHB01] shows that in many cases SRPT has shorter expected flow times than PS for all size jobs and thus behaves as if it were protective. Nonetheless, there are cases where it is not protective and this provides an impediment to its widespread implementation.

*Work supported in part by National Science Foundation Grant No. ANI–9730162.
Email:friedman@orie.cornell.edu, www:http://www.orie.cornell.edu/∼friedman
†Work supported in part by an O'Reilly Foundation Scholarship

In [FH02], Friedman and Henderson proposed a new scheduling algorithm, the Fair Sojourn Protocol which attempts to attain the efficiency of SRPT while guaranteeing protectiveness, thus providing an attractive alternative.

In this paper, we extend that analysis and develop a theory of protective scheduling algorithms. We provide necessary and sufficient conditions for an algorithm to be protective, suggest several new and interesting protective algorithms and analyze their performance.

The remainder of the paper is organized as follows. In Section 2 we describe the model and notation and define protectiveness formally. Section 3 introduces the notion of slack and fully characterizes which algorithms are protective. In Section 4 we introduce some protective algorithms and analyze their performance. Finally, Section 5 provides bounds on these and other algorithms.

## 2 Model and Protectiveness

In the standard scheduling theory notation, the problem we study is as follows: We schedule $n$ jobs preemptively on a single server. The jobs have release times $r_1, r_2, \ldots, r_n$ (we assume $r_1 \leq r_2 \leq \ldots \leq r_n$ for simplicity) and processing requirements $p_1, p_2, \ldots, p_n$. Throughout the paper, we study the online version of this problem; we only know of a job's existence and processing time at its release time. The server has service rate one and can divide its service among any number of jobs in any proportion. No time lags result from such division of service, or from preemptions. For example, at time $t$, PS simply divides its service evenly among all jobs that have release time less than $t$ but have not yet completed.

The objective is to minimize average flow time, where a job's flow time is the difference between its completion time and release time. This problem is traditionally denoted as $(1|r_j, pmtn| \sum_j C_j)$. As is well known [Sch68], the SRPT algorithm produces a schedule that minimizes average flow time. Our goal will be to construct online scheduling algorithms that minimize average flow time, while also guaranteeing protectiveness:

**Definition 1** *An online scheduling algorithm $x$ is* **protective** *if for any input sequence, the flow time of any job under $x$ is not greater than it would have been under PS.*

See [FH02] for an example of how SRPT is not protective.

Our choice as PS as a benchmark for protectiveness is an attempt to formalize the common notion of PS as a benchmark of nonstarvation. While this is not formally stated it is often an unstated assumption (See, e.g., [HBBSA01, BCM98, Tan92, Sta95]). Recall that a job's stretch (or slowdown) is the ratio between its completion time and processing time. Assuming an M/G/1 arrival process, it is shown in [BHB01] that the expected stretch is the same for all jobs under PS. Thus PS is "fair" in a certain appealing, but informal, sense.

PS also satisfies several other notions of fairness and protectiveness. For example, it satisfies Shenker's inverse golden rule which states that a job should not harm other jobs more severely than it is harmed by them [She90]. This idea is formalized in [MS92, FM99] for the fair-share service discipline and its generalization. Rephrasing that work in the context of this paper, it states that one requirement that a scheduling algorithm should satisfy is that no job do worse under it than it would if all jobs were identical to it, both in their release and processing times. Clearly, PS satisfies this requirement.

While we do not have an axiomatic justification for this definition, it appears to be commonly accepted and the common language of an important debate as discussed earlier.

# 3 Characterization

In this section, we provide necessary and sufficient conditions for an algorithm to be protective. The keys to our analysis are two simple ideas: 1) processor sharing provides a stable ordering among jobs and 2) the slack of a job, which measures the degree to which the protectiveness constraint binds. Our construction of a slack system not only provides a useful theoretical tool, but also provides a useful algorithmic framework for practical implementation.

## 3.1 The Slack System

The observation that processor sharing provides a stable ordering is encapsulated in the following simple lemma[1], proved in [FH02]:

**Lemma 1** *Under PS if two jobs $a, b$ are in the system then the order in which they complete is independent of future jobs being released.*

In the following we parallel the construction used in the Fair Sojourn Protocol (FSP) algorithm introduced in [FH02], with some important modifications. Using an idea similar to that in Fair Queuing [PG93], we maintain a virtual PS-schedule. (A similar idea is used for nonpreemptive scheduling in [PSW95].), jobs are indexed according to their completion time in this PS-schedule. As we will see below, FSP devotes all of the machine's processing ability to the job of lowest index in this ordering that has processing time remaining.

Thus, at time $t$ we consider all jobs that were released before $t$ but would not have completed under PS. Order these by the jobs' remaining processing time in the virtual PS-schedule and index them by $1 \leq \eta \leq m$ where $m$ is the number of such jobs at time $t$. Let $v = (v_\eta)_{1 \leq \eta \leq m}$ be the ordered list where $v_\eta$ is the remaining processing time for job $\eta$ under PS and if $\eta' > \eta$ then $v_\eta \leq v_{\eta'}$. Note that under any protective scheduling algorithm, every released but uncompleted job must be in this vector. Let $w_\eta$ be job $\eta$'s remaining processing time under some protective scheduling algorithm.

Lastly, we construct the slack vector, $s$, which will be useful in determining what jobs may be served. When an algorithm is making a decision on which job to process, it must ensure that its decision means that it stays protective. For this purpose, it is useful to calculate the "slack" or room for manoeuver the protocol has.

Suppose that at time $t$ our algorithm has $m$ jobs in the virtual PS-schedule. These have remaining work $v_\eta$ in the PS-schedule ($v_1 \leq v_2 \leq \ldots \leq v_m$) and $w_\eta$ in our protective schedule.

How tightly does the protectiveness constraint bind? Consider the first job. If no further jobs are released, it will complete at time $mv_1$ in the virtual PS-schedule. If our algorithm chooses to work on and serve job 1 to completion then it will complete at time $w_1$ and thus we define its slack, denoted $s_1$, by $s_1 = mv_1 - w_1$. Similarly, the second job will complete at time $v_1 + (m-1)v_2$ in the virtual PS-schedule and time $w_1 + w_2$ in a schedule which serves only jobs 1 and 2, thus we set $s_2 = v_1 + (m-1)v_2 - (w_1 + w_2)$. In general, the slack of the $\eta$-th job is given by

$$s_\eta = \sum_{1 \leq \nu < \eta} v_\nu + (m - \eta + 1)v_\eta - \left( \sum_{1 \leq \nu \leq \eta} w_\nu \right)$$

For a job that has completed but remains in the virtual PS-schedule we define $s_\eta = \infty$.

[1]Note that this result is also known in the context of fair queuing [PG93]. In fact, FSP can be interpreted as a fair queuing protocol. See [FH02] for further discussion

We turn now to the question of the effect of two events on the slack vector $s$.

**Event type 1:** Spend time $\triangle t$ processing job $\zeta$.

Assuming that no jobs are released between $t$ and $t+\triangle t$, we know that $v_\nu(t+\triangle t) = v_\nu(t) + \frac{\triangle t}{m}$. Our schedule works only on job $\zeta$ and so $w_\eta(t+\triangle t) = w_\eta(t)$ for $\eta \neq \zeta$ and $w_\zeta(t+\triangle t) = w_\zeta(t) - \triangle t$. A quick calculation shows that this gives $s_\eta(t+\triangle t) = s_\eta(t) - \triangle t$ for $\eta < \zeta$ and $s_\eta(t+\triangle t) = s_\eta(t)$ for $\eta \geq \zeta$, i.e. spending $\triangle t$ time processing $\zeta$ decreases the slack of all jobs strictly of strictly lower $v$-index than $\zeta$ by $\triangle t$.

**Event type 2:** A job is released at time $t$.

Suppose the job has processing time $x$ and that $x$ satisfies $v_\zeta \leq x < v_{\zeta+1}$ for some $\zeta$.[2] Denote by $v(t^-), w(t^-)$ and $s(t^-)$ the vectors $v, w$ and $s$ just before the job is released. Recall that these vectors will increase in length by one and the the released job will now be at position $\zeta + 1$ in the ordering. A simple calculation shows that

$$\begin{aligned} s_\eta(t) &= s_\eta(t^-) + w_\eta(t), \quad \eta \leq \zeta \\ s_{\zeta+1}(t) &= s_\zeta(t^-) + (m-\zeta)(x - w_\zeta(t^-)) \\ s_\eta(t) &= s_{\eta-1}(t^-), \quad \eta > \zeta + 1 \end{aligned}$$

We refer to the trio of vectors $v(t), w(t)$ and $s(t)$ at any time $t$ as the *slack system*. Using the slack vector, we are able to give a characterization of what it means for an online scheduling algorithm to be protective:

**Theorem 2** *A scheduling algorithm $x$ is protective if and only if $s(t)$ is always non-negative.*

**Sketch of proof:** Suppose that at a time $t$, we have $s_\eta(t) < 0$ for some $\eta$, or expressed otherwise

$$\sum_{\zeta=1}^{\eta} w_\zeta > \sum_{\zeta=1}^{\eta-1} v_\zeta + (m - \eta + 1)v_\eta$$

If there are no further jobs released, the left hand side of the inequality gives the minimum time before jobs $\{1, 2, \ldots, \eta\}$ can finish, whereas the right hand side gives the time they will finish at in the virtual PS-schedule. Thus, not all of jobs $\{1, 2, \ldots, \eta\}$ can complete earlier than they do in the virtual PS-schedule and protectiveness is violated.

This proves the only if direction. For the if direction, it suffices to note that at any time $t$, if $s$ is non-negative, then the algorithm that always serves the uncompleted job with the lowest value of $\eta$ until all jobs are served will be protective. This uses the fact that jobs being released only increases slacks and that serving a job only affects the slack of jobs of strictly lower $v$-index. ■

# 4 The FSP, PFSP and OFSP algorithms

First we recall the FSP scheduling algorithm from [FH02]. This is precisely the algorithm we used in the second half of the proof of Theorem 2. That is, FSP serves the jobs in order of increasing $\eta$.

In order to discuss other protective scheduling algorithms we define two useful concepts:

[2] If a released job's processing time is the same as the remaining processing time of one or more jobs in the virtual PS-schedule, it is inserted into the ordering after these jobs; this convention forces the second inequality to be strict.

**Definition 2**
*1) A job is* **servable** *at time $t$ if a protective scheduling algorithm can serve it at time $t$, independent of future jobs being released.*
*2) A job is* **completable** *if at time $t$ if a protective scheduling algorithm can serve it to completion starting at time $t$, independent of future jobs being released.*

**Theorem 3**
*1) An uncompleted job $\eta$ is servable at time $t$ if and only if for all $\nu < \eta$ $s_\nu > 0$.*
*2) An uncompleted job $\eta$ is completable at time $t$ if and only if for all $\nu < \eta$ $s_\nu \geq w_\eta$.*

**Proof:** This follows easily from the fact that, when no jobs are released, serving a job $\eta$ for time $\triangle t$ decreases the slack on all jobs of strictly lower index than $\eta$ by $\triangle t$. ■

Note that FSP always chooses the lowest $v$-indexed uncompleted servable job and thus by the structure of the slack vector this job is necessarily completable.

Now, if we want to minimize average sojourn time, then by analogy to SRPT we should try to choose the job with the shortest remaining processing time that is servable. However, if this job is not completable, this might not be optimal. Thus, the job we choose should depend on our expectations of future release times. The next 2 scheduling algorithms take extreme expectations.

Pessimistic FSP (PFSP) chooses the completable job with the shortest remaining processing time, while Optimistic FSP (OFSP) chooses the servable job with the shortest remaining processing time. Both algorithms break ties by choosing the job of lowest $v$-index.

**Theorem 4**
*PFSP and OFSP are both protective.*

**Sketch of proof:** This follows easily again from recalling the effect that serving a job $\eta$ has on the slack vector. ■

## 4.1 Comparison of Algorithms

Now that we have a number of online protective algorithms, we need a means to compare their performance.

**Definition 3**
*One online protective algorithm* **strictly dominates** *another if its average flow time is no greater on all inputs and is strictly less on at least one input.*

Note first that OFSP does not strictly dominate FSP. To see this, consider the job sequence $((0,9),(0,9),(10,5))$; label the jobs $A$, $B$ and $C$. Under PS these jobs finish at times 22, 22 and 23 respectively. Thus, FSP will finish them at times 9, 18 and 23 for a total flow time of 40. Now consider what OFSP does. Between times 0 and 9, it serves job $A$. Between time 9 and 10 it serves job $B$. At time 10 the slack system is therefore given by $v = (4,4,5)^T, w = (0,8,5)^T$ and $s = (\infty, 4, 0)^T$. Thus, job $C$ is servable and has the shortest remaining processing time. It will be served until time 14, at which time job $B$'s slack hits zero. Job $B$ is then served to completion (which occurs at time 22) and then job $C$ served to completion (which occurs at time 23). This gives a total flow time of 44, which is worse than FSP's.

In this example, when OFSP chooses to serve job $C$ at time 10, it is "optimistic" in the sense that it hopes a job will be released to preserve protectiveness. This does not occur and OFSP ends up doing worse than FSP. Suppose, however, that a job $D$ with processing requirement 10 is released at time 13. A simple calculation shows that the completion times of jobs $A$, $B$, $C$ and $D$ under FSP will be $(9, 18, 23, 33)$ giving a total flow time of 60, whereas under OFSP it will be $(9, 23, 15, 33)$ giving a total flow time of 57. Thus, on this input, OFSP does have lower total flow time.

Similarly, PFSP and OFSP are not ordered under this idea of strict domination. However, PFSP and FSP are, but to prove this we need the following lemmas.

**Lemma 5** *Consider any slack system $(v, w, s)$ arising at time $t$ from any scheduling algorithm. If the system satisfies the property that the $s$ vector is non-negative and there are no jobs released after $t$, then PFSP is the optimal algorithm to use after $t$ to minimize average flow time from $t$ onwards.*

**Proof:** Suppose that there are $m$ jobs with processing time remaining (i.e. $m$ entries of $w$ are strictly positive); index these jobs $(1, 2, \ldots, m)$.

As there are no further jobs released, any optimal schedule clearly a) will never split processing time between more than one job and b) will not contain any reversals. Thus we only need to consider schedules that serve jobs sequentially according to some ordering of $(1, 2, \ldots, m)$.

Suppose that the optimal schedule is given by $(i_1, i_2, \ldots, i_m)$. Suppose further that $PFSP$ would choose job $a$ to serve first and that $i_j = a$ for some $j > 1$, namely that the optimal schedule does *not* choose job $a$ to serve first.

**Claim 1:** All jobs served before $a$ in the optimal schedule have a lower $v$-index than $a$.

Suppose not. Let $\mathcal{B}$ be the set of jobs of higher $v$-index than $a$ served before $a$ in the optimal schedule. Let $b$ be the job in $\mathcal{B}$ of lowest $v$-index.

**Case 1:** $w_b \geq w_a$. Swap $b$ and $a$ in the optimal ordering. This will not increase the total flow time and is feasible as, by the properties of the slack vector, serving $a$ in $b$'s place reduces the slack on a smaller number of jobs by no greater an amount.

**Case 2:** $w_b < w_a$. We know that at time $t$ job $b$ was not completable (as otherwise it would have been PFSP's choice) but $a$ was. Thus there is a job $c$ of $v$-index less than $b$'s and at least as great as $a$'s with $s_c < w_b$. As there are no further jobs released, the only way $s_c$ can increase is for $c$ to be served (in which case $s_c$ becomes $\infty$). But our choice of $b$ as the job of lowest $v$-index in $\mathcal{B}$ means that this does not happen and so this case does not arise.

Having established this preliminary fact, we are ready to prove the lemma, using a standard interchange argument such as that used in [Sch68]. However, the jobs to interchange must be chosen carefully.

Let $d$ be the last job among $(i_1, i_2, \ldots, i_{j-1})$ that has the property that before serving $d$, job $a$ is completable (note that at least $i_1$ satisfies this), but after serving $d$, it no longer is. If $a$ remains completable after all jobs $(i_1, i_2, \ldots, i_{j-1})$ then let $d = i_{j-1}$.

**Claim 2:** All jobs served between $d$ and $a$ in the optimal schedule have lower $v$-index than $d$.

Suppose not. Then there is a job $e$ which is served between $d$ and $a$ and which also has $v$-index between that of $d$ and $a$'s (we showed in Claim 1 that none could have $v$-index greater than $a$'s).

We know that $w_e \geq w_a$ as if $w_e < w_a$ then $PFSP$ would have chosen $e$ instead at time $t$. However, we know that $a$ was completable before $d$ was served but not afterwards. Thus, serving $d$ created a "blocking job", name it $f$, such that $s_f < w_a$. By the properties of slacks $f$ has lower

$v$-index than $e$. But then $w_e \geq w_a > s_f$ which means that $e$ was not completable and so could not have been served.

We know that $w_d > w_a$, as otherwise PFSP would have chosen $d$. (In the event of a tie, PFSP chooses the job of lower $v$-index; thus had we had $w_d = w_a$, it would have chosen $d$ also.) Interchanging $d$ and $a$ maintains protectiveness as all jobs served between $d$ and $a$ have lower $v$-index than $d$ and their slacks are reduced by a smaller amount ($w_a$). This interchange clearly gives a lower total flow time, contradicting the claim that we had the optimal order in which to serve jobs. This completes the proof. ■

For the next lemma we define a set of algorithms intermediate between FSP and PFSP. Let PFSP(k) be the algorithm which runs PFSP until $r_k$, the time of the $k$-th job release and then switches to FSP. Note that PFSP($\infty$)=PFSP and PFSP(0)=FSP.

**Lemma 6**
*Consider any slack system $(v, w, s)$ arising at time $t$ from any scheduling algorithm. If the system satisfies the property that the $s$ vector is non-negative then PFSP(k) will have no greater average flow time than PFSP($k-1$) for all $k \geq 1$.*

**Sketch of proof:** Both PFSP($k-1$) and PFSP ($k$) behave the same way until the $r_{k-1}$ Suppose that after this job is released, $m$ entries of the $w$ vector are greater than 0 (i.e. $m$ jobs have processing time remaining). From this point PFSP($k-1$) will follow the FSP algorithm; denote this ordering by $(j_1, j_2, \ldots, j_m)$. Absent further jobs being released the PFSP($k$) schedule would be a simple permutation of the PFSP($k-1$) schedule where smaller jobs are moved to earlier locations, clearly giving a lower total flow time.

Now consider the release of job $k$. Both PFSP($k-1$) and PFSP($k$) have the same $v$-vector at all times (indeed all protective algorithms do) and thus this job will be inserted in the same place. If job $k$ does not have the lowest $v$-index among jobs remaining in the PFSP($k-1$)-system, then the permutation argument above still applies.

The other case to consider is when job $k$ does have the lowest $v$-index among jobs remaining under PFSP($k-1$). Here, PFSP($k-1$) will serve job $k$ immediately. However, because of the reordering, PFSP($k$) may have jobs of lower $v$-index than $k$ still in the system. However, we can apply a similar argument to show that the total flow time is no greater.

**Theorem 7** *PFSP strictly dominates FSP.*

**Proof:** We apply the previous lemma inductively, recalling that FSP = PFSP(0) and PFSP = PFSP($\infty$). ■

Both PFSP and OFSP satisfy a strong property - there is no algorithm that strictly dominates them.

**Theorem 8**
*No online protective algorithm strictly dominates PFSP.*

**Proof:** Suppose not, namely that there is an online protective algorithm that strictly dominates PFSP; call it $A$. Let $((r_1, p_1), (r_2, p_2), \ldots, (r_n, p_n))$ be the input on which $A$ has strictly lower average flow time.

Thus there must be a point at which PFSP and $A$ serve a different job. Let $\tau$ be the first such time. Let $k = \max\{i : r_i \leq \tau\}$ and consider the input $((r_1, p_1), (r_2, p_2), \ldots, (r_k, p_k))$, i.e. the original input with all jobs released after $\tau$ removed. As both PFSP and $A$ are online, they will

behave the same way up to $\tau$ on this new input. From Lemma 5, we know that PFSP is the optimal way to continue after $\tau$. At time $\tau$, the algorithm $A$ does something different; specifically it chooses a job with strictly greater remaining processing time than the job chosen by PFSP (we can just relabel jobs if it chooses one of the same length). From the lemma we know therefore that $A$ has greater total flow time. This gives the required contradiction. ■

**Theorem 9**
*No online protective algorithm strictly dominates OFSP.*

**Sketch of proof:** This theorem is proved in an analagous way, except that rather than cutting off jobs released later, we add jobs. ■

### 4.2 The SFSP algorithm

In a practical setting where we might have some statistical information on release times, one could define protective scheduling algorithms which use this information. One simple example is $\gamma$-Stochastic FSP (SFSP($\gamma$)) which chooses the shortest job which is both serviceable and which will be able to be completed with probability greater than $\gamma$. Thus, SFSP($\gamma$) smoothly interpolates between PFSP and OFSP. We are currently analyzing the behavior of SFSP.

## 5 Competitive analysis

In this section we consider the optimality of our scheduling algorithms. First we note that protectiveness is a fairly weak requirement, since PS can do quite badly.

**Theorem 10** *[Motwani et. al.] PS is $\Omega(n/\log n)$ competitive to the optimal nonprotective algorithm (SRPT).*

**Proof:** We repeat their proof here as we will use a modification of it below. Let $H_k$ be the k'th harmonic number. Consider a sample path in which 2 jobs of length 1 are released at time 0, and for all $k \in \{1, \ldots, n-2\}$ a job of length $1/(k+1)$ is released at time $H_k$. It is easy to see that the optimal schedule saves one of the length 1 jobs until the end and thus has total flow time ($n$ times the average flow time) of $2H_{n-1} + 1$. Under PS, note that just before the release of the job of size $1/k$ at time $H_{k-1}$, there are already $k$ jobs of size $1/k$ in the system. Thus all $n$ jobs complete at the same time of $t = H_{n-1} + 1$, giving a total flow time of $2n + H_{n-1} - 1$. Since $H_n = \Theta(\log n)$ this gives the stated competitive ratio. ■

However, as we now show, these losses are in some sense unavoidable by any protective scheduling algorithm.

**Theorem 11** *The FSP algorithm is $\Omega(n/\log n)$-competitive to the optimal nonprotective algorithm*

**Proof:** We modify the previous example by assuming that both of the length 1 jobs are now of length $1-1/n$. Thus, they now finish at $t = H_{n-1}$, whereas the other jobs finish at $t = H_{n-1}+1-\frac{2}{n}$. Thus, FSP will serve the two jobs that are released at time 0 first, and the serve the other jobs in the order in which they are released. Thus the $n-2$ jobs that are released at $H_1, H_2, \ldots, H_{n-2}$ will wait $1-\frac{2}{n}$ before even entering service, giving a $\Theta(n)$ total flow time for this schedule. Analogously to the previous proof SRPT has total flow time of $2H_{n-1} + 1 - 2/k$. The result follows. ■

**Theorem 12** *All protective scheduling protocols are $\Omega(n^{1/2})$ competitive to the optimal nonprotective algorithm (SRPT).*

**Proof:** Consider a input with $n-1$ jobs released at times $0, 1, \ldots, n-2$ each with a processing requirement of one unit. Suppose that another job with processing requirement $X(n)$ is released at time 0; call this job $A$.

Suppose that we have choosen $X(n)$ so that job $A$ finishes at $\frac{n}{2}$ under PS. Consider the optimal protective schedule. We will only enforce protectiveness in a very loose way here; specifically we require that job $A$ is served before time $\frac{n}{2}$ and do not care about all other jobs violating protectiveness. Under these constraints, the optimal algorithm will start serving job $A$ at time $\frac{n}{2} - X(n)$. For all of the length 1 jobs released before this time, it will serve them immediately. At time $\frac{n}{2}$ we will have $X(n)$ jobs that have been backed up. At this point we also no longer have any protectiveness requirement, so SRPT is the optimal algorithm. As all jobs have equal processing requirements it doesn't actually matter which order we serve them in. Suppose we do so in the order in which they are released. Then each job is delayed by $X(n)$. There are at least $\frac{n}{2}$ such jobs and so the total flow time for this schedule will be $\Omega(nX(n))$.

On the other hand, the pure SRPT algorithm will hold job $A$ off until the end and always serve the length 1 jobs the moment they are released. This gives a total flow time of $2n - 2 + X(n)$. Comparing these two gives the competitive ratio of $\Omega(X(n))$.

Computing $X(n)$ is somewhat more involved. It is easy to see, from the harmonic series, that if $X(n) = O(log(n))$ this job will complete by $n/2$ for sufficiently large $n$. However, as sketched in the appendix, $X(n) = n^{1/2}$ also works and gives the stated bound. ■

Note that in the proof above, the cost of protectiveness arises directly from the need to avoid starving the large job, and is therefore unavoidable. However, this still leaves us with the problem of finding the best protective algorithm. First we note that within the realm of protective algorithms FSP can be quite bad.

**Theorem 13** *The FSP algorithm is $\Omega(n/\log n)$-competitive to the optimal protective algorithm*

**Proof:** Consider the same input used in the proof of Theorem 10. As the time 0 jobs will the first elements in the slack system, FSP will serve them before serving the jobs that are released at times $H_1, H_2, \ldots, H_{n-2}$. On the other hand, the SRPT ordering is protective and so, just as in Theorem 10 we get the stated competitive ratio.

However, even if we changed the convention used by FSP and allowed newly released jobs to be inserted in the slack system before jobs of the same $v$-value, we can modify the proof of Theorem 11 to get the same bound. We do this by setting the initial jobs length to be $1 - 1/n^2$, in which case only the job released at time $H_{n-2}$ needs to be delayed compared to the SRPT schedule and thus the total sojourn time of the optimal protective schedule is only $O(1)$ larger than the SRPT schedule, proving the result. ■

However, we conjecture that, when compared to the optimal protective algorithm, OFSP and PFSP are $O(1)$-competitive.

# 6 Acknowledgements

We would like to thank Shane Henderson, Gennady Samorodnitsky, David Shmoys and Jeremy Staum for helpful conversations.

## References


[BCM98] M. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continous job streams. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[BHB01] Nikhil Bansal and Mor Harchol-Balter. Analysis of srpt scheduling: Investigating unfairness. In *Proc. ACM Sigmetrics '01*, 2001.

[FH02] E. Friedman and S. Henderson. Fairness and efficiency in processor sharing protocols to minimize sojourn times. mimeo, Cornell University, 2002.

[FM99] E. J. Friedman and H. Moulin. Three methods to share joint costs. *Journal of Economic Theory*, 87(2):275–312, 1999.

[HBBSA01] Mor Harchol-Balter, Nikhil Bansal, Bianca Schroeder, and Mukesh Agrawal. Size-based scheduling to improve web performance. mimeo, CMU, 2001.

[MS92] H. Moulin and S. Shenker. Serial cost sharing. *Econometrica*, 60:1009–1037, 1992.

[PG93] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking (TON)*, 1(3):344–357, 1993.

[PSW95] C. Phillips, C. Stein, and J. Wein. Scheduling jobs that arrive over time. In *Proc. of Fourth Workshop on Algorithms and Data Structures, LNCS*, pages 86–97. Springer-Verlag, 1995.

[Sch68] L.E. Schrage. A proof of the optimality of the shortest processing remaining time discipline. *Operations Research*, 16:687–690, 1968.

[She90] S. Shenker. Efficient network allocations with selfish users. In P. J. B. King, I. Mitrani, and R. J. Pooley, editors, *Performance '90*, pages 279–285. North-Holland, New York, 1990.

[Sta95] W. Stallings. *Operating Systems*. Prentice Hall, 2nd edition, 1995.

[Tan92] A.S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1992.

# A Appendix

**Lemma 14** *In the proof of Theorem 12, $X(m) = n^{1/2}$ satisfies the assumptions.*

**Sketch of proof:** Let $n(t)$ be the number of jobs in the in the slack system at time $t$. Then, for any positive integers $t, \Delta$,

$$n(t+\Delta) - n(t^+) = \Delta - d \qquad (*)$$

where d is the number of jobs that depart during the interval $(t, t+\Delta]$, where $t^+$ is the limit as time approaches $t$ from above. Now, a job will depart at time $t$ if it was released at time $r$ and $\int_r^t n(t)^{-1} = 1$. Thus, the number of jobs that will complete during the interval is equal to the number that were released during the interval $(a_-, a_+]$, where $\int_{a_-}^t n(t)^{-1} = 1$ and $\int_{a_+}^{t+\Delta} n(t)^{-1} = 1$.

Now define a smoothed version of $n(t)$ as $n_\delta(t) = \int_0^\infty n(s)\phi_\delta(s,t)ds$, where $\phi_\delta(s)$ is a $C^\infty$ bump function centered at $t$ with width $\delta$. Since $n(t) \leq t$ this function is well defined and continuously differentiable.

Now by taking the appropriate limits as $t, T, \delta, \Delta \to \infty$ and $\Delta/t, \delta/T, \delta/\Delta, \to 0$ we can define $n_\delta(t/T) \to \hat{n}(\tau)$. Then (*) becomes a differential equation for $\hat{n}(\tau)$. We can then show that $\hat{n}(\tau) = c\tau^{1/2}$ is a solution for $t$ sufficiently large for some $c > 0$, implying that $n(t) = O(t^{1/2})$ for $t$ sufficiently large.

Now, since $n(t) = O(t^{1/2})$ then for a job of size $X$ to complete by $n/2$ implies that $\int_0^{n/2} n(t)^{-1} > X$, but $\int_0^{n/2} n(t)^{-1} = O(n^{1/2})$, completing the sketch of the proof. ■