Fairness and Efficiency in Web Server Protocols

Eric J. Friedman^{*}
friedman@orie.cornell.edu

Shane G. Henderson[†]
shane@orie.cornell.edu

School of Operations Research and Industrial Engineering
Cornell University
Ithaca, NY 14853

ABSTRACT

We consider the problem of designing a preemptive protocol that is both fair and efficient when one is only concerned with the sojourn time of the job and not intermediate results. Our Fair Sojourn Protocol (FSP) is both efficient, in a strong sense (similar to the shortest remaining processing time protocol: SRPT), and fair, in the sense of guaranteeing that it weakly outperforms processor sharing (PS) for every job on any sample path.

Our primary motivation is web serving in which the standard protocol is PS, while recent work proposes using SRPT or variants. Our work suggests both a framework in which to evaluate proposed protocols and an attractive new protocol, FSP.

Categories and Subject Descriptors

F.2.2 [Nonnumerical Algorithms and Problems]: Sequencing and Scheduling; C.4 [Performance of Systems]: Performance Attributes

General Terms

Algorithms, Performance

Keywords

Processor sharing, starvation avoidance

1. INTRODUCTION AND MOTIVATION

Many queuing problems arising on computers and the Internet allow processor time to be shared among multiple jobs simultaneously. These include data transmission, batch job

^{*}Work supported in part by National Science Foundation Grant No. ANI-9730162.

[†]Work supported in part by National Science Foundation Grant No. DMI 0224884

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMETRICS'03, June 10–14, 2003, San Diego, California, USA.
Copyright 2003 ACM 1-58113-664-1/03/0006 ...\$5.00.

processing and web serving. In all three of these examples, some version of processor sharing is standard. For example, data transmission with FIFO queuing in which file transfers are broken into multiple packets, essentially shares a link among multiple processes simultaneously, while both time-sharing on computers and processor sharing on web servers are the norm. Aside from the simplicity of these processor sharing (PS) protocols, the typical justification for PS is “fairness”: current jobs receive equal processing resources.¹

We study such problems in a general setting. Our main insights apply to problems for which one only cares about the sojourn time of jobs (the time between arrival and service completion of a job) and intermediate processing stages are irrelevant. For example, in data transmission, our analysis applies to bulk file transfer, such as in web serving, but not to telnet sessions, in which intermediate processing is important.

Recently, non-sharing protocols, such as shortest-remaining-processing-time (SRPT) have been recommended for both data transmission [7, 8] and web serving [1].² While such protocols obviously improve the sojourn times for small jobs, [1] present intriguing results that show that, in many cases, even the sojourn times for large jobs are improved under SRPT, in comparison with PS. In many cases these improvements can be dramatic. In fact, understanding these fascinating results is what motivated this paper.

However, improvement is not guaranteed, and in some settings, large jobs can do much worse under SRPT compared to PS. For example, it is well known that under SRPT, long jobs can be starved [15, 14, 2], where one can interpret starvation strictly in the sense of never receiving service, or in the sense that a large job has an extremely long (but finite) sojourn time. We believe that potential starvation provides an impediment to the wide scale implementation of protocols such as SRPT.

In this paper we devise notions of fairness and efficiency

¹Technically, almost all of these examples do not truly exhibit processor sharing, since they are typically implemented via time division multiplexing; however, as long as the time slices are sufficiently small it is reasonable to treat this as true processor sharing.

²Various heuristic protocols that are combinations of PS and SRPT have also been studied. In particular [7, 8, 3] propose families of protocols which are meant to bridge the gap between SRPT and “fair” protocols. Simulation analysis suggest that these protocols are promising, but unfortunately, as yet, there are no formal guarantees of their performance.

that can be used to assist in the development of fair and efficient protocols³. Specifically, the contributions of this paper are as follows.

- We explain the “counterintuitive” result in [1] that under SRPT the expected sojourn times for *all* job sizes can be less than they would be under PS. Basically, this is possible because PS is not an efficient protocol.
- We develop a partial ordering on protocols based on a notion of domination. This partial ordering provides a framework for studying protocols so that, for example, one can classify certain protocols (like PS) as being dominated and therefore poor candidates in practice. This framework then allows one to study and design efficient (nondominated) protocols.
- We show that PS is fair, but inefficient, and so its performance may be significantly worse than an efficient protocol such as SRPT, for *all* classes of jobs. SRPT, while efficient, is not fair.
- We present a new protocol, FSP, that is both fair and efficient. FSP reduces the sojourn times of jobs from those that would be encountered under PS, sometimes dramatically, while still guaranteeing that no job will ever wait longer than it would have done under PS.

FSP can be viewed as a form of Fair Queuing [5, 11]; however, the standard analysis of Fair Queuing produces bounds which are too weak for our purposes.

Our results suggest that, for example, in web-serving, PS is not an ideal choice of protocol, and that FSP is an attractive alternative. The implementation of FSP basically involves simulating the operation of PS in the background, and so it should not involve a great computational burden.

Harchol-Balter et al. [6] point out that implementation of web-serving protocols is difficult using current web-serving software. In particular, it is difficult to achieve precise scheduling of jobs. Nonetheless, they provide an approximate implementation of SRPT that shows significant improvements over the default implementation of PS. Similarly, one could implement an approximation of FSP using currently available tools. Given the results in this paper and those of [1], there is now strong motivation for the development of web-serving software that allows for more precise job scheduling than is currently possible.

2. MODEL

We consider a single server queue with an arbitrary arrival process and arbitrary distribution of job sizes. We allow these to be correlated in any manner. In particular, any of the standard queuing models are allowed in our model as is any arbitrary deterministic sequence of arrivals and service times. Our key assumption is that when a job arrives, its processing time is known with certainty. We define a sample path as the sequence of arrival times and job lengths, i.e., $((t_1, l_1), (t_2, l_2), \dots, (t_n, l_n))$. (The reason that we consider

³Fair and efficient mechanisms are the focus of much of cooperative game theory and cost sharing research, see e.g., [9, 10] for introductions and overviews. For a specific application of these ideas in a related model, see Cres and Moulin [4], who have studied a scheduling model in which all jobs arrive at the same time.

such a general model is that our main results involve sample-path arguments, and as such do not rely on distributional assumptions.)

We consider preemptive scheduling protocols under which the processor may be shared. Formally, such a protocol can be defined as a function ω where $\omega(i; t)$ is the instantaneous work rate on job i at time t . We assume that $\omega(i; t)$ is non-negative and piecewise continuous. Furthermore, we assume that at all times t at which there is work present in the system, $\sum_i \omega(i; t) = \mu$ where μ is the processing rate, i.e., the server allocates its maximum possible effort. Of course, one does not devote effort to jobs that are no longer present, and so the summation is only over jobs that are still present in the system. (For simplicity, and without loss of generality, in the following we will set $\mu = 1$.) Of course, jobs cannot be processed before they arrive, so if $t < t_i$ then $\omega(i; t) = 0$. The completion time of job i is then given by the value of t such that $\int_{t_i}^t \omega(i; s) ds = l_i$. Lastly, we are interested in online protocols, where $\omega(i; t)$ cannot depend on arrivals after time t .⁴ The protocol is the mapping from sample paths to a function ω and we will typically denote a protocol as $p \in P$ where P is the set of all allowed protocols.

Suppose, for now, that one makes sufficient distributional and other assumptions to ensure that expected steady-state quantities can be computed. Given a scheduling protocol p let $so(p; l)$ be the expected steady-state sojourn time of jobs of length l under protocol p (assuming it exists). One reasonable goal is to minimize some weighted expectation of the sojourn times, such as the average sojourn time, $E_l[so(p; l)]$. It is well known that the protocol SRPT minimizes this function over the set of all protocols [12, 13]. Another important measure is the slowdown ratio $sl(p; l) = so(p; l)/l$. For example, [1] analyze starvation (or fairness) in terms of the slowdown ratio.

As we will demonstrate later, FSP is strictly preferred to PS for all jobs except those that terminate a busy period, and thus is guaranteed to have a better average sojourn and slowdown than PS (if such measures exist). As is well known, SRPT is always better than PS in terms of average sojourn times, since it is optimal for that metric [12, 13]. As shown in [1] for specific examples, it may even be better in terms of average slowdown; in fact, they show that in some cases, every class receives a smaller expected slowdown under SRPT than PS. However, in many cases, under SRPT large jobs may occasionally experience extremely large delays, leading to larger average slowdowns for large jobs under SRPT than under PS. In addition, as our simulation results for a specific example show, FSP can have a smaller slowdown ratio than SRPT for large jobs, and have better worst case slowdown behavior for all job sizes. Thus, if slowdown and starvation are considered important criteria then FSP may be quite attractive.

3. EFFICIENCY AND FAIRNESS

The key assumption in our analysis is that only the sojourn time of a job is relevant for performance evaluation. Thus, we will consider the values of sojourn times in developing notions of efficiency and fairness.

First, we present a simple baseline definition of efficiency based on the notion of dominance.

⁴Essentially, Cres and Moulin [4] study an offline version of this problem.

Definition 1. The protocol p' *weakly dominates* protocol p if no job completes later under p' than under p on any sample path. The protocol p' *dominates* p if p' weakly dominates p and there is at least one job on at least one sample path that completes strictly earlier under p' than under p .

Definition 2. A protocol p is efficient if there is no other protocol p' that weakly dominates it.

Inefficient protocols are clearly suboptimal for any reasonable loss measure that only depends on the sojourn times of jobs. Similarly, if some protocol is dominated by another protocol, then the dominated protocol will be no better under any reasonable loss measure.

Definition 3. A reasonable loss measure is a mapping

$$\pi : P \rightarrow \mathbb{R}$$

such that if p' dominates p then $\pi(p') \leq \pi(p)$.

For example, expected sojourn time and expected slowdown are reasonable loss measures, as is any weighted average of sojourn times.

One can develop strong (also called strict in the sequel) versions of these notions. First we define a busy period as a sequence of arrivals that all overlap, i.e., a busy period ends whenever a job completes and there are no jobs waiting for service.⁵ Note that busy periods consist of the same jobs for any nonstalling protocol (i.e., for any protocol that works at the maximum possible rate whenever work is present).

Definition 4. A protocol p' *strongly dominates* protocol p if, for every busy period on every sample path, every job except the terminal one completes strictly earlier under p' than under p .

Definition 5. A strong loss measure is a mapping $\pi : P \rightarrow \mathbb{R}$ such that if p' strongly dominates p then $\pi(p') < \pi(p)$.

Define an arrival process to be trivial (steady-state trivial) if the probability (probability in steady-state) of a busy period consisting of more than one job occurring is 0. For example, if the arrival process is Poisson and job sizes are nonzero, then it is neither trivial nor steady-state trivial. Over finite time intervals, weighted averages of sojourn times in which every job is given nonzero weight are strict loss measures, such as average sojourn time or average slowdown. If we consider limiting averages, then these same measures will be strict loss measures for any nontrivial steady state arrival process.

Next we consider fairness. Following the literature we define PS as the standard of fairness.⁶

Definition 6. A protocol p is fair if it weakly dominates PS.

It is important to emphasize that we define fairness by comparing a protocol with PS. If it does no worse than PS on any sample path then we say it is fair. Another notion of fairness is that expected slowdown is the same for all job sizes, which is a property that PS enjoys under certain assumptions on the model. We prefer to work with minimal assumptions and use sample-path arguments, and so the above definition is appropriate.

⁵For example, if there are no waiting jobs and the current job completes at time t , then we say that the busy period is over, even if a new job arrives at time t .

⁶See, e.g., [1] for further discussion and references.

4. FSP

In this section we present the Fair Scheduling Protocol (FSP), providing an intuitive introduction, the formal protocol and a simple implementation. FSP is essentially equivalent to Fair Queuing in which each job is treated as its own packet stream; however, for completeness and to describe our analysis (for which the standard analysis of FQ is not sufficient [11]) we will present a self-contained description of FSP.

4.1 Motivation

The intuition behind FSP is similar to that in Fair Queuing: Consider a sample path in which n jobs of length 1 arrive at the same time. Under PS the sojourn time for every job is n . However, if we simply order the jobs randomly and then schedule them in order, we get an average sojourn time of $(n+1)/2$ and a longest sojourn time of n . Thus, all jobs receive better service under this ordering than under PS.

Clearly, sharing processor time is always suboptimal. In general, once a job has started processing, that job should continue to completion. Perhaps the only exception to this rule is when a new job arrives. If the new job is small enough, it might be reasonable to switch to processing the new job.⁷

4.2 Explaining FSP

FSP computes the times at which jobs would complete under PS and then orders the jobs in terms of earliest (PS) completion times. FSP then devotes its full attention to the (uncompleted) job with the earliest PS completion time.

This definition only makes sense if the ordering of jobs does not change as new jobs are introduced. The invariance of the ordering follows from the following lemma.⁸

LEMMA 1. *Consider any two sample paths of arrivals that are identical up to time t and two jobs A, B that both arrive before t . Then, under PS, jobs A and B complete in the same order on both sample paths.*

PROOF. If one or both jobs complete by time t then there is nothing to prove. If not, then both jobs are still being processed at time t . The result follows immediately since, under PS, jobs complete in the order of shortest remaining processing time. $\square$

Remark 1. An immediate consequence of this lemma is that there are no “reversals” under FSP. A reversal occurs when job B enters the system before job A , and the processing of job A is interrupted to perform some processing on job B .

There are many nontrivial sample paths on which FSP and SRPT are identical. The simplest example of this occurs when all jobs arrive at the same time, in which case FSP schedules identically to SRPT. See also Examples I and III

⁷There may be situations with correlated arrival times where the non-arrival of a job might provide information that could be used to change the job in service. However, we expect situations of this type to be rare, and do not consider them for this informal discussion.

⁸We have not specified the protocol’s behavior in the event of ties. Although any tie breaking rule would work, for concreteness we assume that in the case of a tie we choose the job with the earliest arrival time.

Figure 1: The common performance of both SRPT and FSP in Example I. First job 2 completes at $t = 2$, then job 3 completes at $t = 3$ and finally job 1 completes at $t = 5$.

Figure 2: The performance of PS in Example I. First job 2 completes at $t = 3.5$, then job 3 completes at $t = 4.5$ and finally job 1 completes at $t = 5$.

in the next section. The main difference between FSP and SRPT is that FSP protects long jobs that have been in the system for a long time against the possibility of starvation, perhaps due to a string of arriving short jobs.

4.3 Examples

We now present several detailed examples of the FSP protocol in order to explain its behavior and provide counter-examples for our later analysis.

4.3.1 Example I

Our first example demonstrates the common case when FSP behaves identically to SRPT. Consider the following example: A job of length 3 arrives at $t = 0$ then a job of length 1 arrives at $t = 1$ and then a job of length 1 arrives at $t = 2$.

Under SRPT we see that job 1 begins in service, then when job 2 arrives it preempts job 1 and goes into service. It then completes at the same time that job 3 arrives and begins service. Then job 3 completes and job 1 returns to service. Figure 1 summarizes this behavior, where the graph gives the remaining processing time required for each job. When the graph of work remaining for a particular job has negative slope it means that it is receiving some processing effort. The behavior of PS is given in Figure 2.

In this example, both FSP and SRPT strictly dominate

Figure 3: The performance of SRPT in Example II. First job 2 completes at $t = 2$, then job 3 completes at $t = 3.9$ and finally job 1 completes at $t = 5.9$.

Figure 4: The performance of PS in Example II. First job 2 completes at $t = 3.5$, then job 1 completes at $t = 5.5$ and finally job 3 completes at $t = 5.9$.

PS, since jobs 2 and 3 have significantly shorter sojourn times under FSP or SRPT than under PS while job 1, the terminal job, completes at the same time for all protocols.

4.3.2 Example II

In this example we show that FSP may differ from SRPT and while FSP dominates PS, SRPT does not. Consider the modification of the above example where job 3 is of length 1.9. The performance of SRPT, PS and FSP is given in Figures 3, 4 and 5 respectively.

We see that job 1, the largest job, completes earlier under FSP than under PS, and earlier under PS than under SRPT. Job 3 completes later under FSP than under SRPT. This example is easily modified to show that FSP can obtain arbitrarily large improvements (for large jobs) over SRPT and PS.

4.3.3 Example III

In this example we give a path on which jobs complete under FSP in a different order than they do under PS, something that might appear at first to contradict the definition of FSP. Consider the following example: Jobs of length 3 and 10 arrive at $t = 0$ and then a job of length 0.9 arrives at $t = 4$. Figures 6 and 7 give the performance of PS and FSP respectively.

Thus, in this example jobs complete in a different order under FSP (1,3,2) than under PS (3,1,2). This is because

Figure 5: The performance of FSP in Example II. First job 2 completes at $t = 2$, then job 1 completes at $t = 4$ and finally job 3 completes at $t = 5.9$.

Figure 6: The performance of PS in Example III. First job 3 completes at $t = 6.7$, then job 1 completes at $t = 6.9$ and finally job 2 completes at $t = 13.9$.

Figure 7: The common performance of both SRPT and FSP in Example III. First job 1 completes at $t = 3$, then job 3 completes at $t = 4.9$ and finally job 2 completes at $t = 13.9$.

under FSP, job 1 completes before job 3 arrives, so even though job 3 would get higher priority than job 1 under FSP, this is irrelevant. Finally, note that both jobs 1 and 3 complete earlier under FSP than under PS while job 2 completes at the same time for both.

4.3.4 Example IV

Now we present a simple example demonstrating starvation. Consider a system in which a new job of length 1 arrives at unit intervals, $t = 0, 1, \dots, 99$ and a single job of length 1.1 also arrives at $t = 0$. Under SRPT the “long” job completes at $t = 101.1$ while it completes at time 2.1 under FSP. (It completes at (approximately) $t = 2.97$ under PS.)

Thus, as is well known, when there is a steady stream of short jobs, a single long job can receive extremely poor service under SRPT.

4.4 Implementation

Although FSP appears to be significantly more complex to implement than either PS or SRPT we suggest that the additional overhead may not be significant. Below we present a simple implementation that is not computationally intensive, and which will prove useful in our formal analysis. For example, compared to the computation performed by a typical webserver to serve a webpage, the computation of the protocol seems insignificant.

The underlying notion of this implementation is that we track the performance of PS as if it were applied “in the background.” This allows us to determine the order in which jobs should be served under FSP.⁹

The basis for our implementation is the ordered linked list v , which has elements $v_i = (j_i, w_i, c_i)$. The job j_i has remaining processing time w_i (under the virtual PS), $c_i = 0$ implies that the job has completed under FSP (although it may not yet have completed under PS which is why it remains in the list), and $c_i = 1$ means it has not. The list is ordered in increasing order of w_i . Associated with v is the time t at which it was last updated, which for simplicity is considered an external variable. Let $|v|$ denote the number of elements in v and if $|v| > 0$ then v_1 is the first element in the list.

Our implementation will rely upon five main routines:

- NextVirtualCompletionTime,
- ProcessJob,
- VirtualJobCompletion,
- RealJobCompletion, and
- JobArrival.

The routines VirtualJobCompletion and JobArrival are used to track the progress that PS would make were it applied to the observed jobs. These five routines may be defined as follows.

⁹Note that we are not following the standard analysis (for Fair Queuing) using “virtual time” in order to be able to prove our main result. In addition, in our setting, we believe that our approach is somewhat more intuitive and perhaps easier to implement.

NextVirtualCompletionTime(v,t):
 [Returns the next "virtual" completion time]
 If $|v| = 0$ then return $\emptyset$
 else return $t + w_1 * |v|$

ProcessJob(v):
 [Returns the job number to process]
 If $|v| = 0$ then return $\emptyset$
 else $[k = \min\{i \mid c_i = 1\}]$
 return j_k .

VirtualJobCompletion(v,t,s):
 [Update when a job completes]
 For all elements of v set $w_i = w_i - (s - t)/|v|$.
 Remove v_1 from v
 Set $t = s$.

RealJobCompletion(v,j):
 [Update when a job completes]
 Find i s.t. $j_i = j$.
 Set $c_i = 0$.

JobArrival(v,t,s,l,j):
 [Update when a new job arrives]
 For all elements of v set $w_i = w_i - (s - t)/|v|$
 Insert $(j, l, 1)$ into v . (maintaining the sort¹⁰)
 Set $t = s$.

Now the implementation of FSP is straightforward. At time 0 let $v = \emptyset$ and $t = 0$. During the running of the algorithm, we only need to do computation when a job arrives, virtually completes or completes. When a job j of length l arrives at time s call $\text{JobArrival}(v, t, s, l, j)$, call $\text{ProcessJob}(v)$ to find the next job to go into service and reset the virtual completion time by calling $\text{NextVirtualCompletionTime}(v, t)$. When we reach a virtual completion time at time s call $\text{VirtualJobCompletion}(v, t, s)$. When the current job j completes, call $\text{RealJobCompletion}(v, j)$ and then call $\text{ProcessJob}(v)$ to find the next job to go into service.

The computation required of this algorithm upon an arrival or a completion (virtual or real) is $O(|v|)$ and $|v|$ is less than or equal to the number of jobs that would have been in service at this time under PS for this sample path. This does not seem to be significantly computationally burdensome compared to the typical complexity of the jobs being serviced. Nonetheless the precise computational costs of FSP deserve further study.

5. EVALUATING EFFICIENCY AND FAIRNESS

First we show that both FSP and SRPT are efficient.

THEOREM 2. *Both FSP and SRPT are efficient.*

Proof: The proof for SRPT follows easily from the fact that it minimizes expected sojourn time. For FSP we use the following argument.

Consider an arbitrary protocol p' and assume that there is a sample path under which p' improves upon FSP, so that all jobs complete either at the same time as in FSP, or earlier. We will show that no job can complete strictly earlier under p' than under FSP.

¹⁰In case of ties place the new job last among equals.

Consider the first job to complete under FSP. By the remark following Lemma 1 on no reversals, the service of this job is uninterrupted. Hence, it completes at the earliest possible time, and so it completes at the same time under both p' and FSP.

Assume that the first n jobs that complete under p' and FSP do so at the same time. Now consider the $(n+1)$ st job to complete under FSP. Again by the "no reversals" property, the only jobs that can have received service while the $(n+1)$ st job was present in the system are those that have already completed under FSP (and p'). Thus, the $(n+1)$ st job to complete must do so as early as possible, conditional upon the serving of the n previously completed jobs. These n jobs completed at the same times under both p' and FSP, so that under FSP, the time at which the $(n+1)$ st job completes is at least as early as the time at which the $(n+1)$ st job completes under p' . Therefore, the $(n+1)$ st job completes at the same time under both p' and FSP, completing the proof. $\square$

Although FSP and SRPT are both efficient, PS is not. This follows from our next result which shows that FSP strongly dominates PS. Thus, PS is wasteful in a very strong sense.

THEOREM 3. *FSP strongly dominates PS.*

PROOF. The proof follows the working of the algorithm presented in the previous section. Let w and x be vectors indicating the remaining work for each job in the system under both PS and FSP respectively. The vectors are ordered in the same way, and so that $w_1 \leq w_2 \leq \dots \leq w_n$. (Hence elements i in both vectors correspond to the remaining work for the *same* job under both disciplines.) It may be the case that $x_i = 0$ while $w_i > 0$ for several values of i .

An event is either the arrival of a new job, the completion of service of a job under PS (a virtual service completion), or the completion of service of a job under FSP. We prove by induction on the sequence of events in a busy period that for all i , $\sum_{j=1}^i x_j \leq \sum_{j=1}^i w_j$ at (immediately after) all event times. We write this as $x \ll w$. This will prove the result, since the next job to complete under PS is always job 1, and at that job completion time (which is an event), $0 = w_1 \geq x_1$, so that $x_1 = 0$, i.e., the job in position 1 completed either at the same instant, or earlier, under FSP.

Suppose that the first job to arrive in a busy period is of size y . At the time of this arrival, $|w| = 1$ and $w_1 = x_1 = y$. Thus, the base case is true. Suppose the result is true at the time of event m , where $m \geq 1$. Let w and x be the vectors at the time of this event, and let y be the time between the occurrence of event m and event $m+1$. Let w' and x' be the vectors w and x updated for the period between event m and event $m+1$ (but not for event $m+1$). Let i be the index of the first nonzero value in x . Then

$$\begin{aligned} x'_j &= x_j = 0, \quad j = 1, \dots, i-1, \\ x'_i &= x_i - y, \\ x'_j &= x_j, \quad j = i+1, \dots, n, \text{ and} \\ w'_j &= w_j - y/|w| \quad \forall j. \end{aligned}$$

It is then easy to see, using the fact that $x \ll w$, that immediately before the occurrence of event $m+1$, the ordering $x' \ll w'$ holds and that all $w'_j \geq 0$ (otherwise a virtual completion would have occurred in the interval). This is sufficient to show weak domination (see below). Strong domi-

nation follows along similar lines since $\sum_{k=1}^j x'_k < \sum_{k=1}^j w'_k$ for all $j \geq i$, except for the last component.

Consider the different types of events that may occur, and their impacts on the vectors w and x .

Case 1: Virtual service completion (completion of job under PS). Since w is maintained in increasing order, we may take the completing job to correspond to w_1 . We have that $0 = w'_1 \geq x'_1$, so $x'_1 = 0$, and it follows that this job completes under FSP either at the same time, or earlier, than it would under PS. Set w and x equal to w' and x' respectively, with the first component deleted. Then $x \ll w$, and the inductive step for this case is complete.

Case 2: Real service completion. In this case, one of the jobs completes under FSP. The vectors w' and x' remain unchanged. We set $w = w'$, $x = x'$, and so $x \ll w$. The inductive step for this case is complete.

Case 3: A new job, of size y , arrives. Insert the value y into the vector w' maintaining the sort, to give w . In case of ties, place y at the last position possible. Insert the value y into the vector x' at the same position to give x . Then it is easy to see that since $x' \ll w'$, we also have that $x \ll w$. The inductive step for this case is complete.

In all 3 cases, the relation $x \ll w$ is preserved, and this completes the proof. $\square$

Thus, as discussed earlier, this implies that for any strict loss measure, FSP will be strictly better than PS. For example, for any nontrivial arrival process¹¹, both average sojourn time and average slowdown will be strictly less for FSP than for PS.

Note that the standard results from Fair Queuing only show that FSP is not worse than PS by the length of the longest job [11]. In the Fair Queuing setting where jobs are single packets, this excess is not significant, but in the web server setting it is.

Note that neither SRPT nor FSP dominates the other, as can be seen from the examples discussed earlier. Thus, depending on the arrival process either SRPT or FSP (or any other efficient protocol) can do better on different loss measures. For example, SRPT is optimal with respect to average sojourn time, while it is possible, and probably common (see the next section) that FSP will typically be superior to SRPT under average slowdown.

In summary we have seen that FSP is both fair and efficient, and SRPT is not fair (see Example II) but is efficient.

6. SIMULATIONS

The results given earlier establish that FSP strongly dominates PS, but that neither FSP nor SRPT dominate the other. Thus, if one restricts attention to these three protocols, it is clear that one should implement either FSP or SRPT. We have shown through examples of sample paths that FSP can provide significantly shorter sojourn times for large jobs as compared to SRPT, since FSP is guaranteed to be fair. But how significant is this? This question is important to consider, given that FSP demands more background computation than SRPT.

In this section we present a small simulation experiment that helps to shed light on the potential tradeoffs between FSP and SRPT. Our goal is not to present a comprehensive simulation study that explores all the intricacies of the

tradeoffs, but rather to show that there is indeed value in exploring FSP as an alternative to SRPT.

Consider a single-server system where jobs arrive according to a Poisson process at rate λ . Successive job sizes are i.i.d. with mean 1. (This choice of mean job size is arbitrary, as one can always choose time units to ensure that this holds.) Following [1], job sizes are chosen to have a truncated Pareto distribution. In particular, the distribution function of job sizes is

$$F(x) = \frac{b^{-\alpha} - x^{-\alpha}}{b^{-\alpha} - c^{-\alpha}}$$

for $x \in [b, c]$, where $c > b > 0$. We chose $c = 10^7$, $\alpha = 2$, and $b = (2 - c^{-1})^{-1}$, which ensures that the mean job size is 1. The constant $\alpha = 2$ differs from that chosen in [1] ($\alpha = 1.1$) to simplify some of the analysis¹². (Choosing $\alpha \approx 1$ apparently reflects current opinion regarding the distribution of file sizes transmitted over the Internet.) Given the modest goals of this section, $\alpha = 2$ seems like a reasonable selection. We chose $\lambda = 0.95$, so that the system is very heavily loaded.

We simulated the completion of 10,000 jobs under PS, SRPT and FSP. Common random numbers were used, so that each protocol saw the same sample path of jobs. For reporting purposes, we grouped the jobs into 6 intervals (or bins) according to job size. The bins were $(0, 2]$, $(2, 4]$, $(4, 7]$, $(7, 10]$, $(10, 20]$ and $(20, \infty)$. For each bin, we recorded the mean of the slowdown ratios and the maximum of the slowdown ratios over all jobs that fell in that bin. This calculation was repeated 30 independent times to enable the construction of confidence intervals.

Table 1 gives the results for the expected *mean* slowdown over 10,000 jobs for SRPT, FSP and PS broken down by bin so that the effect of job size becomes apparent. It also gives a confidence interval for the difference in performance (in each bin) between SRPT and FSP. Note that in all cases except one, the confidence intervals for the difference do not contain zero, so that the differences are statistically significant.

We see from these results that, as expected, PS is not competitive with either SRPT and FSP. The well-known result that PS gives a uniform slowdown over all job sizes is also apparent. The difference between FSP and SRPT is harder to discern. SRPT performs better than FSP on small jobs, while the reverse is true for large jobs. However, the two methods perform quite similarly in terms of expected mean slowdown.

Table 2 gives the results for the expected *maximum* slowdown within each bin over 10,000 jobs for SRPT, FSP and PS. It also gives a confidence interval for the difference in performance (in each bin) between SRPT and FSP. Note that in all cases except one, the confidence intervals for the difference do not contain zero, so that the differences are statistically significant.

Again we see the poor performance of PS relative to both FSP and SRPT. We also see that FSP outperforms SRPT in terms of the expected maximum slowdown of jobs. For example, for job sizes in the interval $(7, 10]$, SRPT gives an expected maximum slowdown (over the first 10,000 jobs) of approximately 54, while the corresponding value for FSP is approximately 38. The confidence interval for the difference in performance shows that with 95% confidence the

¹¹Recall that an arrival process is trivial if the probability of two jobs overlapping is 0.

¹²Choosing $\alpha = 1.1$ would require us to determine b above numerically.

Protocol/Job Size	(0, 2]	(2, 4]	(4, 7]	(7, 10]	(10, 20]	(20, ∞)
SRPT	1.84 ± 0.01	6.1 ± 0.2	10.4 ± 0.7	14 ± 1	16 ± 2	17 ± 3
FSP	2.3 ± 0.4	8.2 ± 0.6	11 ± 1	13 ± 1	14 ± 2	15 ± 2
PS	18 ± 2	18 ± 2	18 ± 2	18 ± 2	18 ± 2	17 ± 3
SRPT - FSP	-0.43 ± 0.04	-2.1 ± 0.5	-1.0 ± 0.8	0.6 ± 0.9	2.1 ± 0.9	1.9 ± 0.7

Table 1: The first three rows give 95% confidence intervals for the expected mean slowdown over the first 10,000 jobs broken down by bin. The final row gives a 95% confidence interval for the expected difference between SRPT and FSP mean slowdown.

Protocol/Job Size	(0, 2]	(2, 4]	(4, 7]	(7, 10]	(10, 20]	(20, ∞)
SRPT	31 ± 3	48 ± 4	58 ± 9	54 ± 8	52 ± 10	35 ± 7
FSP	29 ± 2	37 ± 4	40 ± 5	38 ± 6	36 ± 6	28 ± 5
PS	69 ± 8	61 ± 7	55 ± 7	48 ± 7	42 ± 7	32 ± 5
SRPT - FSP	2 ± 3	12 ± 4	18 ± 7	16 ± 5	16 ± 6	7 ± 4

Table 2: The first three rows give 95% confidence intervals for the expected maximum slowdown over the first 10,000 jobs broken down by bin. The final row gives a 95% confidence interval for the expected difference between SRPT and FSP maximum slowdown.

difference between these 2 values is at least 11, which is a significant fraction of the absolute maximums (54 and 38) observed. This marked improvement in expected maximum slowdown is apparent in almost all bins, clearly showcasing the value of FSP in this example.

Nonetheless, the key difference between FSP and SRPT is the tradeoff between fairness and expected sojourn time, a topic which merits further study.

7. CONCLUDING COMMENTS

We believe that the formal analysis developed in this paper provides a useful framework for analyzing protocols. However, it is merely a first cut. There are many other notions of efficiency and fairness that could be used. Our analysis was motivated by the general theory of cost allocation [16, 10] and its recent application to scheduling protocols [4]. We believe that the combination of ideas from this literature with those from scheduling and queuing theory is useful in understanding these issues.

There remain many interesting open problems relating to other efficient protocols, of which there are many. Additionally, one important direction for further research is the extension of our analysis and protocols to cases where job times are not completely known upon arrival but can only be estimated (perhaps with improving accuracy) during processing. This would seem to be important for dynamic web pages, such as those created by a database query in which the processing time is unknown, but can often be estimated with some degree of accuracy. In this setting one could modify FSP to use expected processing times. However, we have no formal results in that setting.

It would be of some interest to attempt to derive closed-form formulae characterizing FSP’s performance for simple models. While such models do not often capture reality well, they provide some insight into the factors that influence the performance of the algorithm. Of course, the sample-path analysis given in this paper must retain an important, and we believe prominent, role in the analysis of protocols, particularly because the assumptions in queueing models are typically quite unrealistic. For example, in web serving the arrival processes may be extremely complex and simple ar-

rival processes are likely to be insufficient models of reality. One might expect arrivals to be highly correlated (due to the nature of html), or arrival rates might be endogenous, since faster serving encourages users to increase their request rate.

The key to our analysis is the assumption that performance of a protocol only depends on sojourn times and not intermediate results. If one accepts this then perhaps the only argument for PS over FSP is based on the complexity of FSP. Given the amount of computing power available in most applications, we believe that the overhead from FSP is negligible compared to the processing requirements of typical jobs, such as those in web serving, but this belief needs to be investigated. The simulation results reported in the previous section suggest that the protective aspects of FSP compared to SRPT are also significant, although the comparison here is not as clear-cut.

Although our analysis of FSP compared to PS is clear-cut, and we believe compelling, the comparison of FSP to SRPT is not. Since neither protocol dominates the other this is not surprising. Our simulation results suggest that while large jobs tend to fare better under FSP than under SRPT, the reverse is true for small jobs. Quantifying the magnitude of these effects is therefore of great interest. Additionally, the comparison between FSP and other heuristic protocols [7, 8, 3] designed for these environments remains wide open. Since there are few analytic results, clearly a careful simulation-based study is called for. Nonetheless, we believe that the analytic (sample path) aspects of our analysis might be applicable to understanding and refining these protocols.

8. REFERENCES

- [1] N. Bansal and M. Harchol-Balter. Analysis of SRPT scheduling: Investigating unfairness. In *Proc. ACM Sigmetrics ’01*, 2001.
- [2] M. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.
- [3] L. Cherkasova. Scheduling strategies to improve response time for web applications. In

- High-Performance Computing and Networking: International Conference and Exhibition*, pages 305–314, 1998.
- [4] H. Cres and H. Moulin. Scheduling with opting out: Improving upon random priority. *Operations Research*, 49(4):565–577, 2001.
 - [5] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking*, 1(1):3–26, Jan. 1990.
 - [6] M. Harchol-Balter, N. Bansal, B. Schroeder, and M. Agrawal. Size-based scheduling to improve web performance. Mimeo, CMU, 2001.
 - [7] E. Modiano. Scheduling algorithms for message transmission over a satellite broadcast system. In *Proceedings of IEEE MILCOM97*, pages 628–634, 1997.
 - [8] E. Modiano. Scheduling packet transmissions in a multi-hop packet switched network based on message length. In *International Conference on Computer Communications and Networks*, 1997.
 - [9] H. Moulin. *Axioms of Cooperative Decision Making*. Cambridge University Press, New York, 1991.
 - [10] H. Moulin. Axiomatic cost and surplus-sharing. In Arrow, Sen, and Suzumura, editors, *Handbook of Social Choice and Welfare*. North-Holland, New York, 2002.
 - [11] 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.
 - [12] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16:687–690, 1968.
 - [13] D. Smith. A new proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 26:197–199, 1976.
 - [14] W. Stallings. *Operating Systems*. Prentice Hall, 2nd edition, 1995.
 - [15] A. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1992.
 - [16] H. P. Young. *Cost Allocation: Methods, Principles and Applications*. Elsevier Science, New York, 1985.