

SAP: An Efficient Scheduling Protocol for Web Servers

Eric J. Friedman*

Shane G. Henderson†

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

January 15, 2002

Abstract

We present a new scheduling protocol that we call the “starvation avoidance protocol” (SAP). SAP is designed for situations in which users primarily care about the completion time of a job, such as web serving. SAP is similar to the shortest remaining processing time (SRPT) protocol in that small jobs typically receive priority. However, unlike SRPT, SAP has a safeguarding feature that ensures that large jobs cannot be starved indefinitely. In particular, SAP guarantees that every job completes at least as early as it would under processor sharing (PS), a common web serving protocol. Indeed, SAP is strictly better than PS for every nonterminal job during a busy period. Thus, SAP provides an attractive protocol for web servers.

1 Introduction

Processor sharing (PS) is a common protocol in many settings. In particular, many web servers use PS. Recently Bansal and Harchol-Balter [BHB01]

*Work supported in part by National Science Foundation Grant No. ANI-9730162. Email: friedman@orie.cornell.edu, [www:http://www.orie.cornell.edu/~friedman](http://www.orie.cornell.edu/~friedman)

†Work supported in part by National Science Foundation Grant No. DMI-0085165. Email: shane@orie.cornell.edu, [www:http://www.orie.cornell.edu/~shane](http://www.orie.cornell.edu/~shane)

(denoted BH01 in the remainder of the paper) have proposed that the shortest remaining processing time protocol (SRPT) might be superior to PS for web serving. They present both analytical and simulation based evidence for the superiority, on average, of SRPT over PS in this setting. However, the main objection to SRPT is that large requests might be starved [Tan92, Sta95, BCM98]. BH01 provide some suggestive evidence that this might not be a significant problem; in fact, in certain cases, even large jobs receive better service, on average, under SRPT. However, starvation can arise under SRPT and even when SRPT is better on average for large jobs it can still be significantly worse for specific jobs. We believe that this drawback to SRPT provides an impediment to its wide-scale implementation.

In this paper, we present a protocol which is similar in spirit to SRPT but avoids these problems. Our starvation avoidance protocol (SAP) is guaranteed to perform at least as well as PS for every job on any sample path. In fact, it strictly outperforms PS for every job except the terminal one during any busy period. Thus, starvation with SAP can never be worse than with PS. This is possible since, as we show, PS is not efficient – it is often quite wasteful in this setting. (In fact, our analysis helps to explain the advantage SRPT has over PS as analyzed in BH01.)

Other protocols have been proposed based on heuristic arguments. In particular [Mod97, Che98] both propose families of protocols which are meant to bridge the gap between SRPT and starvation free protocols. However, there are no formal guarantees for these protocols.

The key aspect of web serving which drives these results is the observation (noted in BH01) that users only care about the completion time of a job.

Thus the intermediate processing progress is irrelevant. It is precisely in this setting, which occurs in web servers, that PS is so inefficient. When intermediate stages in processing are useful (such as during telnet sessions or time-sharing in general), PS retains its value.

2 Model

We consider a single server queue with an arbitrary arrival process and arbitrary distribution of jobs 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 arrivals and job lengths, i.e., $((t_1, l_1), (t_2, l_2), \dots, (t_n, l_n))$. (The reason that we consider such a general model is that our main results involve sample-path arguments, and as such do not rely on distributional assumptions.)

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). Another important measure is the slowdown ratio $sl(p; l) = so(p; l)/l$. For example, HB01 analyze starvation (or fairness) in terms of the slowdown ratio.

As we will demonstrate later, SAP is strictly preferred over 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. As is well known, SRPT is always better than PS in terms of average sojourn times (since it is optimal for that metric [Sch68, Smi76]) and as shown in BH01 for specific examples, it may 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 starvation, leading to larger average slowdowns for large jobs under SRPT than under PS. In addition, as our simulation results show (for a specific example), SAP has a smaller slowdown ratio for large jobs than SRPT, and had better worst case slowdown behavior for all job sizes. Thus, if slowdown and starvation are considered important criteria we expect that SAP would be preferred to both SRPT and PS.

3 SAP

3.1 Intuition

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. Thus, we see that PS can be very inefficient. In fact, in this example, PS is the worst nonstalling protocol possible.

More generally, for any sample path under PS, consider two concurrent jobs a and b where a will complete first at time t . If we were to trade some of b 's processing time before t to a for the same amount of a 's processing time after t (and before the completion time of b) then a would complete earlier

and b 's completion time would be unchanged. Thus it is easy to see that since we only care about sojourn times, it is always possible to improve on PS. In fact, for any busy period, we can strictly improve the service on every single job except for the terminal one, i.e., the job that completes last under PS. This is the motivation for SAP, which guarantees that all jobs except the terminal one in any busy period are completed earlier than they would have under PS. Note that there are many other protocols that also achieve this goal. We plan to examine this class of protocols in the future.

Thus, 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.¹

3.2 Explaining SAP

SAP attempts to mimic PS as much as possible while obeying the following rule. In SAP we compute the time at which jobs would complete under PS and then order the jobs in terms of earliest (PS) completion times. SAP then devotes its full attention to the job with the earliest (PS) completion time. Such a definition is, in fact, well defined as the following lemma and discussion show.²

Lemma 1 *Consider any two sample paths of arrivals that are identical up*

¹There 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. We expect situations of this type to be rare.

²Note that we 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.

to time t and two jobs a, b that both arrive before t . Then under PS they 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. (Note however, that at any point in time the remaining processing times under PS can differ from those under SRPT.) ■

The importance of this lemma is that given that several jobs remain to be completed at some time t , they will complete in the same order under PS *regardless* of the arrival times and sizes of jobs that arrive after time t . Therefore, the ordering of processing under SAP is well-defined.

Thus, there are no reversals under PS, where job A begins processing and then we switch to job B even though job B had entered the system before we started job A.³

Note that there are many nontrivial sample paths on which SAP and SRPT are identical. The simplest example of this occurs when all jobs arrive at the same time, in which case SAP schedules identically to SRPT. See also Examples I and III in the next section. The main difference between SAP and PS is that SAP 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.

³Note that in the common case where processor sharing is actually implemented via time division multiplexing then one can view the inefficiency of PS as arising from its numerous alterations between jobs being processed.

3.3 Examples

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

3.3.1 Example I

Our first example demonstrates the common case when SAP 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. The following table summarizes this behavior, where a * indicates that the specified job is in service from the current time to the next indicated one.

SRPT and SAP: Example I					
Time	0	1	2	4	5
Remaining work (1)	*3	2	2	*2	0
Remaining work (2)	x	*1	0	0	0
Remaining work (3)	x	x	*1	0	0

Thus, we see that first job 2 completes at $t = 2$, then job 3 completes at $t = 3$ and finally job 1 completes at $t = 5$. Note that in this case SAP is identical to SRPT.

Next consider the behavior of PS as tabulated here:

PS: Example I						
Time	0	1	2	3.5	4.5	5
Remaining work (1)	*3	*2	*1.5	*1	*0.5	0
Remaining work (2)	x	*1	*0.5	0	0	0
Remaining work (3)	x	x	*1	*0.5	0	0

In this case, first job 2 completes at $t = 3.5$, then job 3 completes at $t = 4.5$ and finally job 1 completes at $t = 5$.

Thus we see that, in this case, both SAP and SRPT strictly path dominate PS as jobs 2 and 3 have significantly shorter sojourn times under either of these than under PS while job 1, the terminal job, completes at the same time for both.

3.3.2 Example II

In this example we show that SAP may differ from SRPT and while SAP path dominates PS, SRPT does not. Consider the modification of the above example where job 3 is of length 1.9. In this case we have the following:

For SRPT:

SRPT: Example II					
Time	0	1	2	3.9	5.9
Remaining work (1)	*3	2	2	*2	0
Remaining work (2)	x	*1	0	0	0
Remaining work (3)	x	x	*1.9	0	0

Here, first job 2 completes at $t = 2$, then job 3 completes at $t = 3.9$ and finally job 1 completes at $t = 5.9$.

Next consider the behavior of PS:

PS: Example II						
Time	0	1	2	3.5	5.5	5.9
Remaining work (1)	*3	*2	*1.5	*1	0	0
Remaining work (2)	x	*1	*0.5	0	0	0
Remaining work (3)	x	x	*1.9	*1.4	*0.4	0

We see that first job 2 completes at $t = 3.5$, then job 1 completes at $t = 5.5$ and finally job 3 completes at $t = 5.9$. Thus, job 1, the largest job, completes earlier under PS than under SRPT.

Lastly, consider the behavior of SAP:

SAP: Example II					
Time	0	1	2	4	5.9
Remaining work (1)	*3	2	*2	0	0
Remaining work (2)	x	*1	0	0	0
Remaining work (3)	x	x	1.9	*1.9	0

In this case first job 2 completes at $t = 2$, then job 1 completes at $t = 4$ and finally job 3 completes at $t = 5.9$. Thus, job 1, the largest job, completes much earlier under SAP than under either PS or SRPT while job 3 completes later under SAP than under SRPT.

This example is easily modified to show that SAP can obtain arbitrarily large improvements (for large jobs) over SRPT and PS.

3.3.3 Example III

In this last example we show a path for which jobs complete under SAP in a different order than they do under PS, something which might appear at first to contradict the definition of SAP. Consider the following example: Jobs of length 3 and 10 arrive at $t = 0$ and then a job of length 0.9 arrives at $t = 4$.

The following table summarizes the behavior of SRPT:

SRPT: Example III					
Time	0	3	4	4.9	13.9
Remaining work (1)	*3	0	0	0	0
Remaining work (2)	10	*10	9	*9	0
Remaining work (3)	x	x	*0.9	0	0

Here, first job 1 completes at $t = 3$, then job 3 completes at $t = 4.9$ and finally job 2 completes at $t = 13.9$.

Next consider the behavior of PS:

PS: Example III					
Time	0	4	6.7	6.9	13.9
Remaining work (1)	*3	*1	*0.1	0	0
Remaining work (2)	*10	*8	*7.1	*7	0
Remaining work (3)	x	*0.9	0	0	0

Under PS, first job 3 completes at $t = 6.7$, then job 1 completes at $t = 6.9$ and finally job 2 completes at $t = 13.9$.

Lastly, consider the behavior of SAP:

SAP: Example II					
Time	0	3	4	4.9	13.9
Remaining work (1)	*3	0	0	0	0
Remaining work (2)	10	*10	9	*9	0
Remaining work (3)	x	x	*0.9	*0	0

Under SAP, first job 1 completes at $t = 3$, then job 3 completes at $t = 4.9$ and finally job 2 completes at $t = 13.9$.

Thus, in this example jobs complete in a different order under SAP (1,3,2) than under PS (3,1,2). This is because under SAP, job 1 completes before job 3 arrives, so even though job 3 would get higher priority than job 1 under SAP, this is irrelevant. Finally, note that both jobs 1 and 3 complete earlier

under SAP than under PS while job 2 completes at the same time for both.

3.3.4 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 2.1 under SAP. (Note that 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 Implementation

Although SAP appears to be significantly more complex to implement than either PS or SRPT we believe that compared to the computation performed by a typical webserver the additional overhead is not significant. Below we present a simple implementation that is not computationally intensive.

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 SAP.

The basis for our implementation is the ordered linked list v , which has elements $v_i = (j_i, r_i, c_i)$. The job j_i has remaining processing time r_i (under the virtual PS), $c_i = 0$ implies that the job has completed under SAP (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 r_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 5 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 5 routines may be defined as follows.

NextVirtualCompletionTime(v,t): [Returns the next ‘‘virtual’’ completion time]

If $|v| = 0$ then return \emptyset

else return $t + r_1 * |v|$

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 $r_i = r_i - (s - t)/|v|$.

Remove v_1 from v

Set $t = s$.

RealJobCompletion(v,t,s,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 $r_i = r_i - (s - t)/|v|$

Insert $(j, l, 1)$ into v . (maintaining the sort⁴)

Set $t = s$.

Now, the implementation of SAP is straightforward. At $t = 0$ 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 $\text{RealJobCompletion}(v, t, s, j)$ and then call $\text{ProcessJob}(v)$ to find the next job to go into service.

Note that the computation required of this algorithm upon an arrival or a completion (virtual or real) is $O(|v|)$ and that $|v|$ 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 SAP deserve further study. (Harchol-Balter et. al. [HBBSA01] have tested an implementation of SRPT and shown that the computational costs are small.)

5 Properties of SAP

Our main result will show that SAP is clearly superior to PS as for any sample path, almost every job completes earlier under SAP than under PS. This also shows that PS is clearly wasteful in a very strong sense.

⁴In case of ties place the new job last among equals.

First we define a busy period as a sequence of arrivals which 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.

Theorem 2 *During any busy period all jobs except the terminal job have a strictly shorter sojourn time under SAP than under PS.*

Proof: To prove the theorem we follow the working of the algorithm we presented in the previous section. Let v and w be vectors indicating the remaining work for each job in the system under both PS and SAP respectively. The vectors are ordered in exactly the same way, and so that $v_1 \leq v_2 \leq \dots \leq v_n$. (Hence elements i in both vectors correspond to the remaining work for the *same* job under both disciplines.) It may be the case that $w_i = 0$ while $v_i > 0$ for several values of i and the vector v need not be ordered.

In the following discussion, an event is either a new job arriving, a job completing under PS (a virtual service completion), or a job completing under SAP. We will prove by induction on the sequence of events in a busy period that for all i , $\sum_{j=1}^i w_j \leq \sum_{j=1}^i v_j$ at (immediately after) all event times. (We will write this as $w \ll v$.) 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), $0 = v_1 \geq w_1$, so that $w_1 = 0$, i.e., the job in position 1 completed either at the same instant, or earlier.

Suppose that the first job to arrive in a busy period is of size y . At the time of this arrival, $|v| = 1$ and $v_1 = w_1 = y$. Thus, the base case is true.

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

Suppose the result is true at the time of event m , where $m \geq 1$. Let v and w be the vectors at the time of this event.

Let x be the time between the occurrence of event m and event $m + 1$. Let v' and w' be the vectors v and w 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 w . Then,

$$w'_j = w_j = 0, \quad j = 1, \dots, i - 1$$

$$w'_i = w_i - x$$

$$w'_j = w_j, \quad j = i + 1, \dots, n$$

$$v'_j = v_j - x/|v| \quad \forall j.$$

It is then easy to see, using the fact that $w \ll v$, that immediately before the occurrence of event $m + 1$, the ordering $w' \ll v'$ holds and that all $v'_j \geq 0$ (otherwise a virtual completion would have occurred in the interval). In fact, we get strict inequality for all components from i onwards except for the last component.

Now consider the different types of events that may occur, and their impacts on the vectors v and w .

Case 1: Virtual service completion (completion of job under PS). Since v is maintained in increasing order, we may take the completing job to correspond to v_1 . We have that $0 = v'_1 \geq w'_1$, so $w'_1 = 0$, and it follows that this job completes under SAP either at the same time, or earlier, than it would under PS. Set v and w equal to v' and w' respectively, with the first component deleted. Then $w \ll v$, and the inductive step for this case is complete.

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

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

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

Thus, we see that SAP “dominates” PS in a very strong sense. This immediately implies “domination for every size job” and also “domination on average.”

Corollary 3 *When the appropriate steady-state averages are defined, $so(SAP;l) \leq so(PS;l)$ and $sl(SAP;l) \leq sl(PS;l)$ for all l .*

In fact, the above inequalities are strict for any “sufficiently random” arrival processes, since all nonterminal jobs in a busy period finish strictly earlier under SAP than under PS.

Thus, PS is inefficient in a very strong sense – there is another protocol that “dominates” it on every sample path. However, neither SAP nor SRPT are inefficient in this sense. In fact, they are efficient in the following strong sense.

Definition 1 *A protocol p is strongly efficient if there is no other protocol p' which is better on any sample path, in the sense that no job completes later*

under p' and at least one job completes earlier under p' .

Theorem 4 *Both SAP and SRPT are strongly efficient.*

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

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

Before proceeding, we note the following. Consider the k 'th job to complete under SAP. Then during that job's time in the system, from its arrival to its completion, no job that completes later than the k 'th job can receive service since this would contradict Lemma 1. From this it is easy to see that the first job to complete under SAP does so at the earliest possible time, since it is uninterrupted.

We proceed inductively. Consider the first job to complete under SAP. By the discussion in the previous paragraph it cannot complete sooner under any protocol. Thus this job completes at the same time under both p' and SAP. Assume that the first n jobs that complete under p' and SAP do so at the same time. Now consider the $(n + 1)$ st job to complete under SAP. By the discussion in the previous paragraph the $(n + 1)$ st job to complete must do so as early as possible, conditional upon the servicing of the n previously completed jobs. These n jobs completed at the same times under both p' and SAP, so that under SAP, 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 SAP, completing the proof. ■

Thus, we see that depending on the arrival process either SRPT or SAP (or any other strongly efficient protocol) can do better on average. In addition, it is possible, and maybe even common (see the next section) that SRPT will be superior to SAP under average sojourn time, but worse under average slowdown, a measure that BH01 use to examine “fairness.”

6 Simulation

The path-by-path results given earlier establish that SAP strictly path dominates PS, but that neither SAP nor SRPT dominate the other. Thus, if one restricts attention to these three protocols, it is clear that one should implement either SAP or SRPT. We have shown through examples of sample paths that SAP can prevent starvation of large jobs as compared to SRPT. But how much starvation is avoided? This question is important to consider, given that SAP 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 SAP 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 SAP 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 BH01, 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 BH01 ($\alpha = 1.1$) to simplify some of the analysis⁶. (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 SAP. 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, SAP 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 SAP. Note that in all cases except one, the confidence intervals for the difference do not contain zero, so that the differences are statistically significant.

⁶Choosing $\alpha = 1.1$ would require us to determine b above numerically

Protocol	(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
SAP	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 - SAP	-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 SAP mean slowdown.

Protocol	(0, 2]	(2, 4]	(4, 7]	(7, 10]	(10, 20]	(20, ∞)
SRPT	31 ± 3	48 ± 4	58 ± 9	54 ± 8	52 ± 10	35 ± 7
SAP	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 - SAP	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 SAP maximum slowdown.

We see from these results that, as expected, PS is not competitive with either SRPT and SAP. The well-known result that PS gives a uniform slowdown over all job sizes is also apparent. The difference between SAP and SRPT is harder to discern. SRPT performs better than SAP 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, SAP and PS. It also gives a confidence interval for the difference in performance (in each bin) between SRPT and SAP. 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 SAP and SRPT. We also see that SAP 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 SAP is approximately 38. The confidence interval for the difference in performance shows that with 95% confidence the 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 SAP in this example.

7 Conclusions

The key to the application of our analysis to web serving is the assumption that performance of web serving only depends on sojourn times and not intermediate results. If one accepts this then perhaps the only argument for PS over SAP is based on the complexity of SAP. However, given the amount of computing power available in even the slowest web server, we believe that the overhead from SAP is negligible compared to the processing needed to create all but the simplest web pages. The simulation results reported in the previous section suggest that the protective aspects of SAP compared to SRPT are also significant, although the comparison here is not as clearcut. Thus, it would seem natural to implement SAP on many web servers and other queuing systems where PS has been used.

Although our analysis of SAP compared to PS is clearcut, and we believe

compelling, the comparison of SAP to SRPT is not. Since neither path dominates the other this is not surprising. Additionally, the comparison between SAP and other heuristic protocols [Mod97, Che98] 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.

Note that we believe that sample path arguments, as used in this paper, are quite important in the analysis of web serving, since the arrival processes may be extremely complex and simple arrival processes are likely to be insufficient models of reality. For example, one might expect arrivals to be highly correlated.

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 SAP to use expected processing times. However, we have no results for that protocol.

References

- [BCM98] 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.

- [BHB01] Nikhil Bansal and Mor Harchol-Balter. Analysis of srpt scheduling: Investigating unfairness. To appear in Proceedings of ACM Sigmetrics 2001 Conference on Measurement and Modeling of Computer Systems, 2001.
- [Che98] L. Cherkasova. Scheduling strategies to improve response time for web applications. In *High-performance computing and network-ing: international conference and exhibition*, page 305-314, 1998.
- [HBBSA01] Mor Harchol-Balter, Nikhil Bansal, Bianca Schroeder, and Mukesh Agrawal. Size-based scheduling to improve web performance. mimeo, CMU, 2001.
- [Mod97] E. Modiano. Scheduling algorithms for message transmission over a satellite broadcast system. In *Proceedings of IEEE MIL-COM'97*, page 628-634, 1997.
- [Sch68] L.E. Schrage. A proof of the optimality of the shortest processing remaining time discipline. *Operations Research*, 16:678-690, 1968.
- [Smi76] D.R. Smith. A new proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 26:197-199, 1976.
- [Sta95] W. Stallings. *Operating Systems*. Prentice Hall, 2nd edition, 1995.

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