The Case for SRPT Scheduling in Web Servers

Mor Harchol-Balter*
Laboratory for Computer Science
MIT, NE43-340
Cambridge, MA 02139
harchol@theory.lcs.mit.edu

Mark E. Crovella†
Department of Computer Science
Boston University
Boston, MA 02215
crovella@bu.edu

SungSim Park†
Laboratory for Computer Science
MIT
Cambridge, MA 02139
kanadala@mit.edu

October 23, 1998

Abstract

The Shortest-Remaining-Processing-Time (SRPT) scheduling policy is known to be the optimal policy for minimizing mean response time, but it is rarely employed in computing systems for a number of reasons. These reasons include: lack of knowledge of task size, fear of starvation of the large tasks, concern over pre-emption overhead, and lack of empirical evidence on the performance benefits of switching to SRPT. In this paper we argue that the special characteristics of Web servers and Web workloads make the usual objections to SRPT less persuasive.

We start by arguing that it is possible for Web servers to extract task sizes for a large fraction of tasks. We then compare SRPT to an alternative policy – processor sharing (PS) – which we use as an idealization of typical scheduling policies currently used in Web servers. Our comparisons are made both analytically (assuming Poisson arrivals and an empirically-derived file size distribution) and on trace-driven simulations using logs from operating Web servers. With respect to performance, we show that at high server utilization, the SRPT policy can reduce mean waiting time and mean slowdown over PS by well over an order of magnitude.

*Supported by the NSF Postdoctoral Fellowship in the Mathematical Sciences.

†Supported in part by NSF Grants CCR-9501822 and CCR-9706685.

‡Supported by DARPA N00014-95-1-1246.

With respect to our main concern, starvation, we show that although starvation is in fact a problem under SRPT for many workloads, in the particular case of *Web* workloads, starvation even for the largest one percent of all tasks is far lower under SRPT than under PS. Lastly, we argue that pre-emption overhead is lower under SRPT than under currently used policies. Although our model is a simplified model of real Web servers, our results suggest that SRPT is an attractive alternative to current scheduling policies in Web servers.

1 Introduction

We consider a Web server consisting of a single machine which receives and processes HTTP requests arriving on-line. We ask the question: In what order should the machine schedule the HTTP requests so as to maximize performance?

Such Web servers today are typically Unix or Windows NT machines and the scheduling is performed in the operating system (rather than in the Web server). Under the assumption that the processes or threads used by the Web server require approximately similar balances of CPU demand and I/O, the scheduling on these servers is approximately Round-Robin, which we model in this paper with Processor-Sharing (PS). [2]

However, it is well known that the on-line scheduling policy which minimizes mean flow time is (preemptive) Shortest-Remaining-Processing-Time-First (SRPT) [12]. In fact SRPT is optimal for any sequence of task arrival times and service demands.

This begs the question: Why isn't SRPT the scheduling policy being used in Web servers? The immediate answer is that SRPT requires knowledge of each task's service requirement, and this information is not currently available. However, it is in fact easy to estimate a task's service requirement: First, observe that the amount of work represented by a Web request is (at least approximately) proportional to the size of the file requested. Next, note that in the majority of cases, file size (and therefore, task size) can be determined by the server at the time the request arrives. This is the case when the request is for a *static* file, *i.e.,* one that is served without modification from the filesystem. Typical studies indicate that over 90-95% of Web requests are for static files [1]. In fact this average statistic may reflect a situation in which, for many servers, essentially all requests are for static files — while the non-static files are served mainly by a relatively few servers [13].

Given that the task size (service demand) is known, there are three issues which need to be addressed in using SRPT:

1. Does the performance gain really justify switching to SRPT?
2. What about starvation of the large tasks? This is the principal objection to using SRPT, and thus is a primary focus of this paper.
3. Does SRPT cause too many pre-emptions?

The organization of this paper is in two parts. The first part of the paper (Sections 2, 3, and 4)

[2] Processor Sharing is defined as Round-Robin in the limit where the quantum size shrinks to zero.

is purely analytical. In the analytic model we assume that the task size distribution is a particular hybrid distribution consisting of a body which is lognormal and a tail that declines via a power-law. This distribution has been shown to be a good model of measured Web file sizes. It has the characteristics of having very high variance and a heavy tail. Although our task size distribution is very realistic, for purposes of analysis we're also forced to assume Poisson arrivals. The second part of paper (Section 5) uses a trace driven simulation of a Web server which allows us to evaluate the effects of realistic arrival processes. We consider 4 different traces of Web server HTTP requests.

The first question above asks whether the performance improvement obtained from using SRPT justifies the expense of rewriting the scheduler. We compare the performance of SRPT with that of PS on two performance metrics: mean *flow time* (time from task arrival to task departure) and mean *slowdown*, where the slowdown of a task is defined to be the task's flow time divided by its size.

Our analytical comparison of SRPT versus PS shows that with respect to mean flow time the performance of SRPT is significantly better than that of PS at high loads (*e.g.,* mean flow time under PS is three times that under SRPT at $\rho = 0.9$). With respect to mean slowdown, the advantage of SRPT over PS is even greater; SRPT improves on PS by as much as a factor of 10 at high load ($\rho = 0.9$).

The performance differences between SRPT and PS are even more dramatic in trace-based simulation. In simulation, mean flow time under SRPT improves over that under PS by an order of magnitude at high load. Furthermore, in simulation mean slowdown is two orders of magnitude smaller under SRPT than under PS at high load.

The second question above is probably the most cited reason for not using SRPT: starvation of the large jobs. We use a job's slowdown as the measure of whether the job is being starved. We consider slowdown as a function of the percentile of the job size distribution (for example, we examine the slowdown of jobs in the 99th percentile of the job size distribution – only 1% of all jobs are bigger than this job). Again, we address this question both analytically and using a trace-driven simulation.

In the analysis section we show that starvation can be a legitimate concern for many task size distributions; however this is *not* the case for the kind of task size distributions that characterize Web requests. In analysis, we find that under the analytical model of the Web task size distribution, the slowdown under SRPT across *all* job size percentiles is significantly lower than under PS – an order of magnitude lower under high loads. The slowdown of jobs in all size percentiles remains close to 1 even under high loads. Thus, starvation is in fact not an issue under the heavy-tailed task

size distribution which characterizes Web file sizes. In contrast, we show that starvation of large jobs can be a problem when using SRPT for less variable task size distributions, and we explain why this is the case. We also examine starvation in the context of the trace-driven simulation and again show that slowdown under SRPT across *all* percentiles is far below that under PS. In particular, slowdown under PS is two orders of magnitude larger than that under SRPT for all jobs, except the largest 1% (and still one order of magnitude larger for the largest 1%).

The third question above asks whether SRPT is practical to implement. Our concern is the number of preemptions required by SRPT, since a preemption takes place in SRPT every time a shorter job arrives. We address this issue in Section 4 and show that this too is not a problem.

Throughout this paper we assume a very simple model of a Web server – namely a single-resource machine. In reality most HTTP requests alternately demand service from multiple devices including disk and the CPU. For ease of analysis, we have compressed these two devices into one with respect to scheduling. Although our Web server model is a simplified one, we feel that the results in this paper are dramatic enough to make a case for considering changing the scheduling of Web HTTP requests to an SRPT-based policy.

2 Performance of SRPT for Web Server Task Sizes

In this section we attempt to evaluate the potential performance improvements that are possible if SRPT scheduling is used in the context of our model of a Web server.

Our analytical results throughout this paper are based on the following formulas for the mean flow time for a task of size x, for an M/G/1 queue with load ρ under SRPT[15] and under PS[12]:

$$\begin{aligned} & E\{\text{Flowtime for a task of size } x \text{ under SRPT}\} \\ &= E\{\text{Waiting time for task of size } x\} + E\{\text{Residence time of a task of size } x\} \\ &= \frac{\lambda \int_0^x t^2 dF(t) + \lambda x^2 (1 - F(x))}{2\left(1 - \lambda \int_0^x t dF(t)\right)^2} + \int_0^x \frac{1}{\left(1 - \left(\lambda \int_0^t z dF(z)\right)\right)} dt \end{aligned}$$

$$E\{\text{Flowtime for a task of size } x \text{ under PS}\} = \frac{x}{1-\rho}$$

where $F(\cdot)$ is the cumulative distribution function of the service time distribution and λ is the arrival rate. From the above two formulas, we easily derive all the other metrics of interest in the paper.

We adopt the assumption that the amount of work represented by a Web request is proportional to the size of the file requested. This is reasonable as a approximation; a more accurate model including a fixed startup cost for each task would not affect our results significantly. As discussed in Section 1, we assume that file sizes, and therefore task sizes, can be determined by the Web server when the task arrives [1].

A number of previous studies have developed empirical models for the distribution of file sizes seen on Web servers [2, 7, 4]. An important property of Web file size distributions is that they typically exhibit *heavy tails.* By heavy tails we mean that the tail of the empirical distribution function declines like a power law. That is, if a random variable X follows a heavy-tailed distribution then

$$P[X > x] \sim x^{-\alpha}, \quad 0 < \alpha < 2$$

where $f(x) \sim a(x)$ means that $\lim_{x\to\infty} f(x)/a(x) = c$ for some positive constant c.

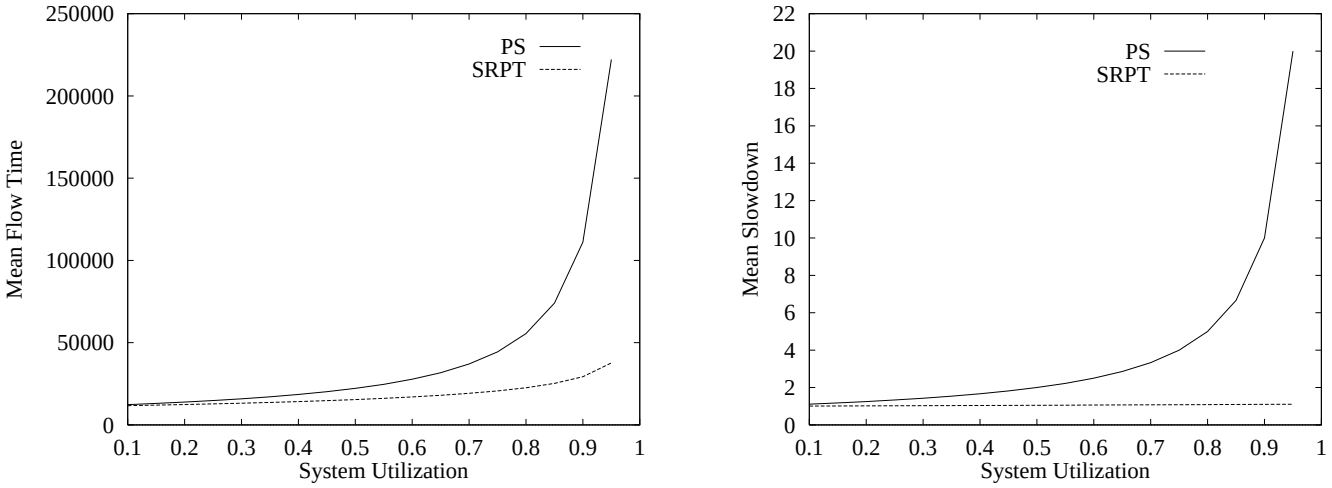
Random variables that follow heavy tailed distributions typically show extremely high variability in size. This is exhibited as many small observations mixed with a small number of very large observations. The implication for Web files is that a small fraction of the largest files makes up most of the load on a Web server. We refer to this as the *heavy-tailed property* of Web task sizes; it is central to the discussion in this paper.

Although Web files typically show heavy tails, the body of the distribution is usually best described using another distribution. Recent work has found that a hybrid distribution, consisting of a body following a lognormal distribution and a tail that declines via a power-law, seems to fit well some Web file size measurements [4, 3]. As a result we initially show results using such a model for task sizes, which we call the *Empirical* model; parameters of the Empirical model are shown in Table 1.

	Distribution	PMF	Range	Parameters
Body	Lognormal	$\frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x-\mu)^2/2\sigma^2}$	$0 \le x < 9020$	$\mu = 7.630$; $\sigma = 1.001$
Tail	Bounded Pareto	$\frac{\alpha k^\alpha}{1-(k/p)^\alpha} x^{-\alpha-1}$	$9020 \le x \le 10^{10}$	$k = 631.44$; $\alpha = 1.0$; $p = 10^{10}$

Table 1: Empirical Task Size Model

We show the potential benefits of the SRPT policy under this task size distribution in Figure 1. The mean of this distribution is 11108, which is therefore the smallest possible value of mean flow time. In Figure 1(a) we show mean flow time; Figure 1(b) shows mean slowdown.



(a) (b)

Figure 1: Performance of SRPT and PS for Empirical Task Size Distribution

These figures demonstrate the impressive performance improvements possible under SRPT scheduling as compared to PS. While both policies yield mean flow times close to the minimum possible at low load, at high load the performance of PS degrades severly according to the $1/(1-\rho)$ relation. On the other hand, SRPT is remarkably resistant to breakdown at high loads. Even when load reaches 0.95, mean flow time under SRPT is only about three times its minimum possible value; in this region, mean flow time under PS is 20 times the same minimum possible value.

Even more striking is the mean slowdown under SPRT as a function of load. Even when load reaches 0.95, mean slowdown is only 1.1 (as compared to 20 under PS). This indicates that almost all tasks will have predictable flow time.

In examining the behavior of SRPT under the Empirical model, we find that the important performance effects are determined by the tail of the distribution. In fact, for the analysis used in this paper we can approximate the Empirical distribution with a much simpler one that has the same tail, but is power-law over its entire range. This is the Bounded Pareto (BP) distribution. This distribution has probability mass function

$$p(x) = \frac{\alpha k^\alpha}{1-(k/p)^\alpha} x^{-\alpha-1} \quad k \leq x \leq p.$$

The reason that we can approximate the Empirical distribution with the Bounded Pareto is that the performance effects are dominated by the tail of the distribution. This effect is shown in Figure 2. In this figure, we plot the mean flow time for SRPT under the Empirical distribution (which has an α value of 1.0) and under BP distributions with α values of 0.9 and 1.1. We do not plot the BP distributions with α value of 1.0 because its curve is indistinguishable from that of the

Empirical. This plot shows that over the entire range of system utilizations, the BP distribution is a good approximation for the Empirical distribution in terms of performance under SRPT.

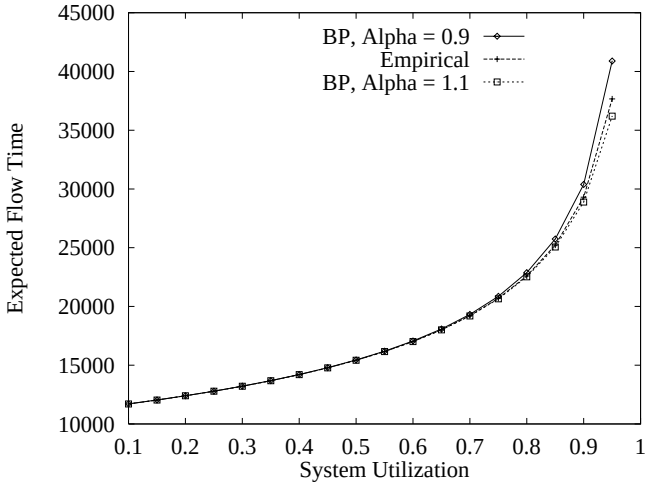


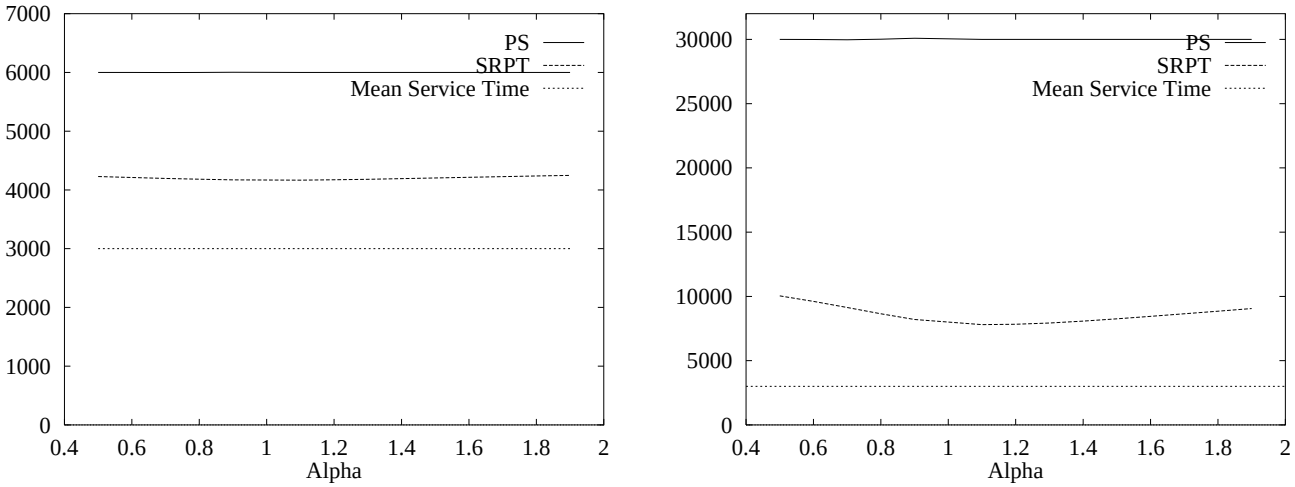
Figure 2: Performance of SRPT and PS for Empirical and BP Distributions

Thus, to simplify analysis in the remainder of the paper we use the BP distribution as a substitute for the Empirical distribution. To allow uniform treatment throughout we adopt the convention that the distributional mean is set at 3000; this is in approximate agreement with most empirical measurements. In addition, we fix the upper bound of the distribution at 10^{10} to reflect a condition of high variance; this value (10 GB) represents the a reasonable estimate of the practical upper limit for file sizes in the current Web. Note that recent results show that the performance of an $M/G/1$ queue in which service times follow a heavy-tailed distribution is not significantly affected (at practical timescales) when the distribution is truncated at a sufficiently large value [10].

The important free parameter in the BP distribution is α. The particular value of α determines the weight of the tail. For small values of α, the heavy-tailed property is more pronounced. Empirical measurements of α vary; typical values are in the range 1.0 to 1.5, but values outside this range are possible as well. Thus it is important to examine the performance of the SRPT policy over a range of α values.

Plot 3 shows BP for SRPT and PS, over a range of α values. This figure shows that, surprisingly, the relative performance of PS vs. SRPT is fairly independent of the particular value of alpha. Thus we find that SRPT is a fairly robust policy that should be successful over a range of file size distributions.

However, this does not mean that the value of the α parameter is unimportant. In fact the particular value of α has a significant impact on the likelihood of starvation, as we show in the next



(a) $\rho = 0.5$ (b) $\rho = 0.9$

Figure 3: Mean Flow Time of SRPT and PS for range of α values in BP Distribution.

section.

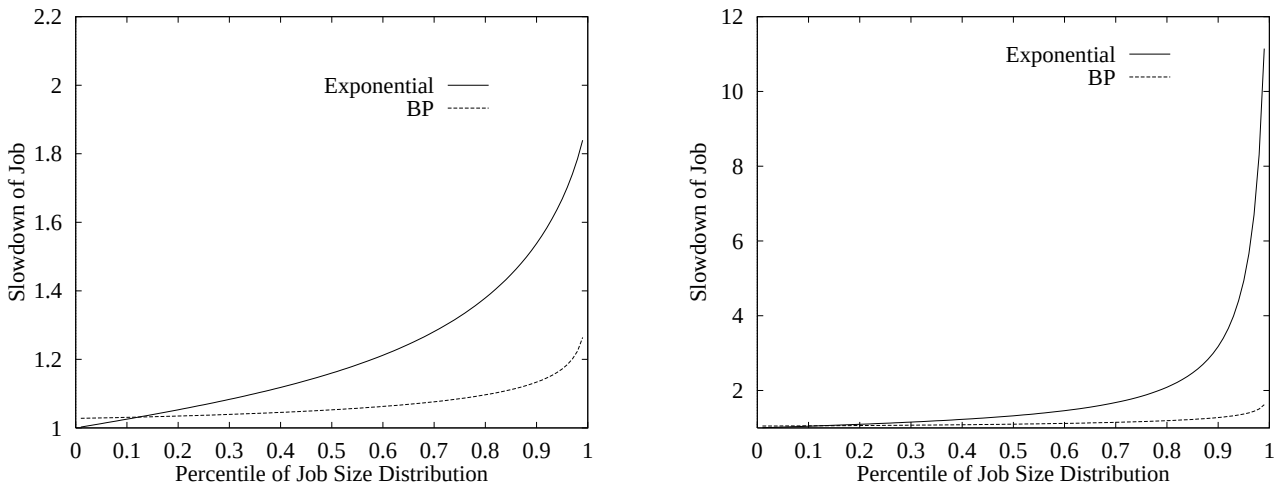
3 Does SRPT Starve Large Jobs?

A common concern with the SRPT discipline is that, by giving preference to small tasks, large tasks may starve. The SRPT discipline has in the past been rejected for use in Web servers specifically for this reason [5].

In this section we'll show that while the fear of starvation is well-founded for some task size distributions (such as the exponential task size distribution), starvation is not a significant concern when task sizes follow heavy-tailed distributions like those that model Web file size requests.

As long as the system under study is in steady-state, every task that enters the system eventually leaves; thus we can use slowdown as a measure of a task's starvation. So to evaluate the potential for starvation of large tasks we plot the mean slowdown of a task of a given size, as a function of the task size. Task size is plotted in percentiles of the task size distribution, which allows us to assess the fraction of largest tasks that will achieve mean slowdown greater than a given threshold value. All results are analytically-derived for an $M/G/1$ queue under SRPT.

Figure 4 shows the mean slowdown as a function of task size under the SRPT discipline. The two curves represent the case of an exponential task size distribution and a Bounded Pareto task size distribution with $\alpha = 1.1$. The two distributions have the same mean. Figure 4(a) shows the situation under low load, $\rho = 0.5$, and Figure 4(b) is the same plot for high load, $\rho = 0.9$.



(a) $\rho = 0.5$ (b) $\rho = 0.9$

Figure 4: Mean slowdown under SRPT as a function of task size.

Figure 4 shows that large mean slowdowns do not occur at low load in either case. However, under high load, there can be starvation of tasks, but *only for the Exponential distribution.* For example, the largest 5% of tasks under the Exponential distribution all experience mean slowdowns of 5.6 or more, with a non-negligible fraction of task sizes experiencing mean slowdowns as high as 10 to 11. In contrast, *no* task size in the BP distribution experiences a mean slowdown of greater than 1.6. Thus, when the task size distribution has low variability (Exponential), SRPT can tend to starve a significant fraction of tasks; however when task size distributions show high variability (BP distribution), SRPT does not lead to starvation.

To understand why SRPT does not tend to starve tasks under the BP distribution, we plot mean slowdown as a function of task size over a range of BP task size distributions with constant mean (in this case, 3000) and varying α. This plot is shown in Figure 5. The high α cases represent low variability, whereas the low α cases represent high variability in the task size distribution.

This figure shows how the likelihood of starvation under SRPT increases as the variability of the task size distribution decreases. When α is less than about 1.5, there is very little tendency for SRPT to starve large tasks (curves for $\alpha = 0.5$ and $\alpha = 0.7$ stay so close to 1 as to be invisible on the plot). Only as α gets close to 2.0 (*e.g.,* 1.7 or 1.9) is there any significant fraction of tasks that experience high mean slowdowns.

The surprising resistance of the high variance task size distributions to starvation under SRPT can be understood by considering how work arrives at the server under such distributions. For a Bounded Pareto distribution with $\alpha = 1.1$, the largest 1% of all tasks account for more than half

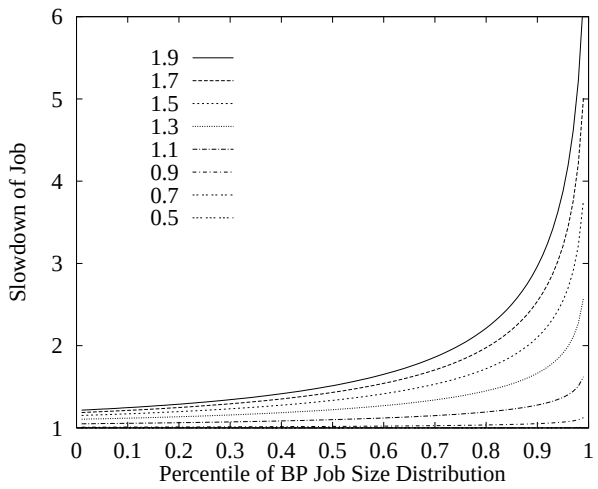


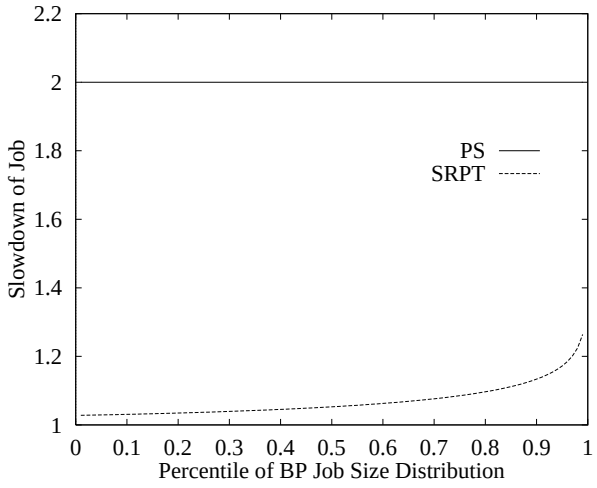
Figure 5: Mean slowdown under SRPT as a function of task size, varying α of task size distribution.

the total service demand arriving at the server. For comparison, for an Exponential distribution with the same mean, the largest 1% of all tasks make up only 5% of the total demand. Thus, under the Bounded Pareto distribution, large tasks (for example, the largest 1%) are interrupted much less (by less than 50% of the total work arriving) than are the same fraction of tasks under the Exponential distribution (interrupted by about 95% of the total work arriving).

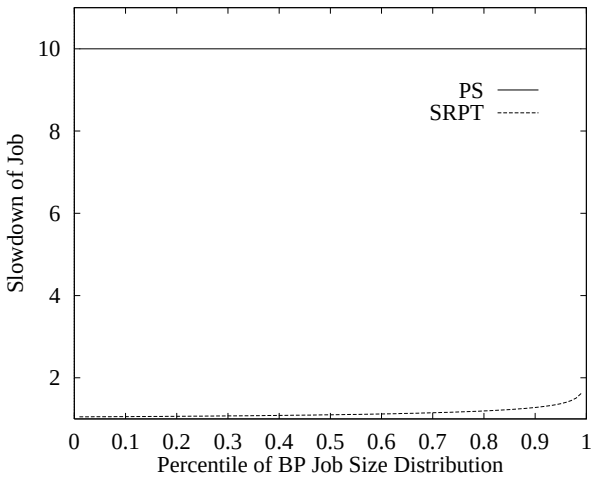
So far we have shown that starvation under the BP task size distribution is not as severe as it would be under the exponential task size distribution. However SRPT may still not be a desirable policy if the Processor-Sharing (PS) discipline shows even lower levels of starvation. Now we show that when the task size distribution is BP with $\alpha = 1.1$, starvation under PS is higher than it is under SRPT, over the entire range of task size percentiles.

Figure 6 plots mean slowdown for PS and SRPT versus percentile of task size under the BP task size distribution with $\alpha = 1.1$. Figure 6(a) considers the low-load situation ($\rho = 0.5$) and Figure 6(b) considers the high-load case ($\rho = 0.9$). The Figure 6 shows the PS discipline consistently yields much larger mean slowdowns for tasks of all sizes. For example, in the heavy-load case, the PS discipline results in mean slowdown of 10 for all task sizes, whereas under the SRPT discipline, tasks of all sizes experience mean slowdowns under 2.

Thus we've seen that while starvation appears to be a concern for low-variability task size distributions like the Exponential, under the BP task size distribution with low α, SRPT is much more attractive. In addition when the task size distribution is BP, SRPT is more attractive than PS with respect to starvation.



(a) $\rho = 0.5$



(b) $\rho = 0.9$

Figure 6: Slowdown as a Function of Job Size, Bounded Pareto Distribution ($\alpha = 1.1$), For SRPT and PS Policies.

4 Does SRPT cause too many pre-emptions?

Another possible cause for concern with SRPT is the number of pre-emptions it requires, because a larger task is always pre-empted when a smaller task arrives.

In this section we point out that under typical conditions, SRPT should result in fewer pre-emptions than Round Robin (a practical implementation of Processor Sharing). We consider a pre-emption to be either the suspension of one task to run another task, or the entry of a task to an empty system, or the departure of a task that leaves the system empty. In each of these cases, significant work must be done by the scheduler to manage task or process contexts.

First, note that under SRPT there are twice as many pre-emptions as there are task arrivals. To see this, consider that SRPT can be implemented using a sorted list of tasks, with the property that two tasks never reverse their relative order in the list. The current executing task is always at the head of the list. When a task arrives it finds its place in the ordered list based on its remaining time, possibly displacing the task at the head of the list. Now each task reaches the head of the list only once. This, along with its departure, are the two pre-emptions it generates.

Second, for the Round-Robin system, we can lower bound the number of pre-emptions by the total busy time divided by the quantum length. In order for the system's performance under Round-Robin to approximate that of Processor Sharing, the quantum length should be small relative to the mean task size. Thus the number of pre-emptions per task will typically be large (and not less than 2).

Thus it seems that the SRPT policy should generate much fewer pre-emptions in practice than the common alternative policy, Round Robin.

5 Effects of Realistic Arrival Processes

5.1 Why do we need a trace-driven simulation

Up until now, all results have been analytically-derived. To apply analysis, we required the assumption of a Poisson arrival process. We also required the assumption of a closed-form distribution for the task size distribution, which we derived from empirical data.

However, it is well known that realistic HTTP arrival processes are more bursty (the interarrival times have a higher coefficient of variation) than a Poisson process [8, 9]. The primary advantage of a trace-driven simulation is that it allows us to evaluate the effect of this increased bustiness.

A second advantage of the trace-driven simulation is that, although we have so far carefully modelled task size distribution analytically, the simulation employs actual measured task sizes.

5.2 The trace data

We ran our simulator on 4 different traces, taken from the Internet Traffic Archives.[3]

The only part of the trace data that we used in each case was the timestamp of the request and the size in bytes of the request.

The ClarkNet trace contains two week's worth of HTTP requests to the CLarkNet WWW server, from August 28, 1995 through September 10, 1997. ClarkNet is a full Internet access provider for the Metro Baltimore-Washington DC area.[4] The trace logs about 1.7 million HTTP requests.

The NASA trace consists of all HTTP requests to the NASA Kennedy Space Center WWW server in Florida during the months of July and August 1995.[5] The trace logs about 1.9 million HTTP requests.

The EPA trace contains all HTTP requests to the EPA WWW server located at Research Triangle Park, NC, during August 29, 1995.[6] The trace logs about 47,000 HTTP requests.

[3] `http://ita.ee.lbl.gov/traces.html`

[4] The ClarkNet log was collected by Stephen Balbach of ClarkNet, and contributed by Martin Arlitt (mfa126@cs.usask.ca) and Carey Williamson (carey@cs.usask.ca) of the University of Saskatchewan.

[5] The log was collected by Jim Dumoulin of the Kennedy Space Center, and contributed by Martin Arlitt and Carey Williamson of the University of Saskatchewan.

[6] The logs were collected by Laura Bottomley (laurab@ee.duke.edu) of Duke University.

The NCSA trace contains HTTP requests made to the NCSA WWW server located at UIUC in Champaign-Urbana on December 20, 1995. The trace logs about 44,000 HTTP requests.

5.3 The trace-driven simulation

We simulated a single processor with a single resource (CPU). Job interarrival times and service requirements were taken from the traces. For each trace, we created a version of the trace representing system utilization ranging from .05 to .95 (.05, .10, .15, .20, etc.). To do this we simply scaled the interarrival times by the appropriate factor.

A simulation consisted of running the entire trace through the processor. Each arrival was sampled with probability 1/40, so that on average every 40th arrival was sampled, and its flow time and slowdown were recorded. This data was used to create the mean flow time and slowdown plots and the percentile plots. We chose not to sample every single arrival since there would be too strong a corrolation between subsequent arrivals. We acknowledge that there is still a corrolation even between what every 40th arrival sees, however no trace was long enough to allow us to perform many independent runs which each converge to steady-state.

5.4 Results

We show full results for one of the traces, NCSA and partial results for all the other traces.

Figure 7 shows the results for the NCSA trace.[7] Figure 7(a) shows mean flow time as a function of server utilization for the case of SRPT scheduling as compared with PS scheduling. Figure 7(b) is the corresponding figure for mean slowdown. Figures 7(a) and 7(b) corroborate the general result of Section 2, namely that the performance of SRPT is far better than the performance of PS on Web workloads. Observe however that the distinction between SRPT and PS is more exagerated in the trace-based results than the analytical formulas from Section 2 predicted. For example, at high load, $\rho = .9$, the mean flow time of PS for the trace-based data is a factor of 10 times greater than the mean flow time for SRPT. Furthermore, the mean slowdown of PS for the trace-based data is a factor of about 90 times greater than the mean slowdown of SRPT. Contrast this with the analytical results from Section 2 which, for the case of load $\rho = .9$, showed only a factors of 3 improvement with respect to mean flow time and a factor of 10 improvement with respect to mean slowdown.

[7] For many of these figures, we do not show the entire PS curve, since it would have dwarfed the shape of the SRPT curve. This is true throughout the paper. In the text, we describe the results of the high load case, $\rho = .9$, although they are not always visible on the plots.

The difference between the trace-based results and the analytical results can be explained by the following observation. A bursty arrival process harms the performance of PS because when tasks arrive together, they must *share* the processor longer, thus all slowing each other down. On the other hand, bursty arrival processes do not seem to harm the performance of SRPT since the SRPT algorithm will simply schedule the shortest one to run to completion. (This point might be clearer to understand when comparing PS and FCFS. Simultaneous arrivals have a more adverse affect on PS than they do on FCFS.)

A big advantage of simulation is that we have sample standard deviations. Figures 7(c) and 7(d) show the standard deviation of the flow time and slowdown. SRPT shows some improvement over PS with respect to the mean flow time, but the important effect is that the standard deviation of slowdown is 20 times better under high load ($\rho = 0.9$). This indicates that the system is performing much more predictably; when users submit small (large) tasks, the flow time is predictably small (large).

We next turn to the issue of starvation, discussed in Section 3. Figures 7(e) and 7(f) show mean flow time and mean slowdown as a function of task size under the heavy load case of $\rho = .9$. Figure 7(f) in particular indicates how SRPT performs as compared with PS with respect to starvation. The corresponding figure in the analytical section is Figure 6. Figure 7(f) shows the mean slowdown in increments of 5% of the task size distribution. Our results show that tasks below the 95th percentile in size have slowdowns under 2. Looking at the largest 5% of tasks we found that tasks in the 99 to 100 percentile (the largest 1% of tasks in the trace) had a mean slowdown of 75, however the tasks in the 98th to 99th percentile had mean slowdown of 8, tasks in the 97th to 98th percentile had a mean slowdown of 6, and tasks in the 96th to 97th percentile had a mean slowdown of only 4. These numbers might seem high, however they are far less than the slowdowns under PS, which averaged around 200 across all task sizes, as shown in Figure 7(f).

Figure 8 shows the performance for the remaining three traces: EPA, NASA and ClarkNet under the SRPT algorithm and under the PS algorithm. For each trace, we plot the mean flow time as a function of server utilization, mean slowdown as a function of server utilization and mean slowdown as a function of task size under the fixed utilization of $\rho = 0.9$. For all metrics, the performance under these three traces, are even more dramatic than under the NCSA trace.

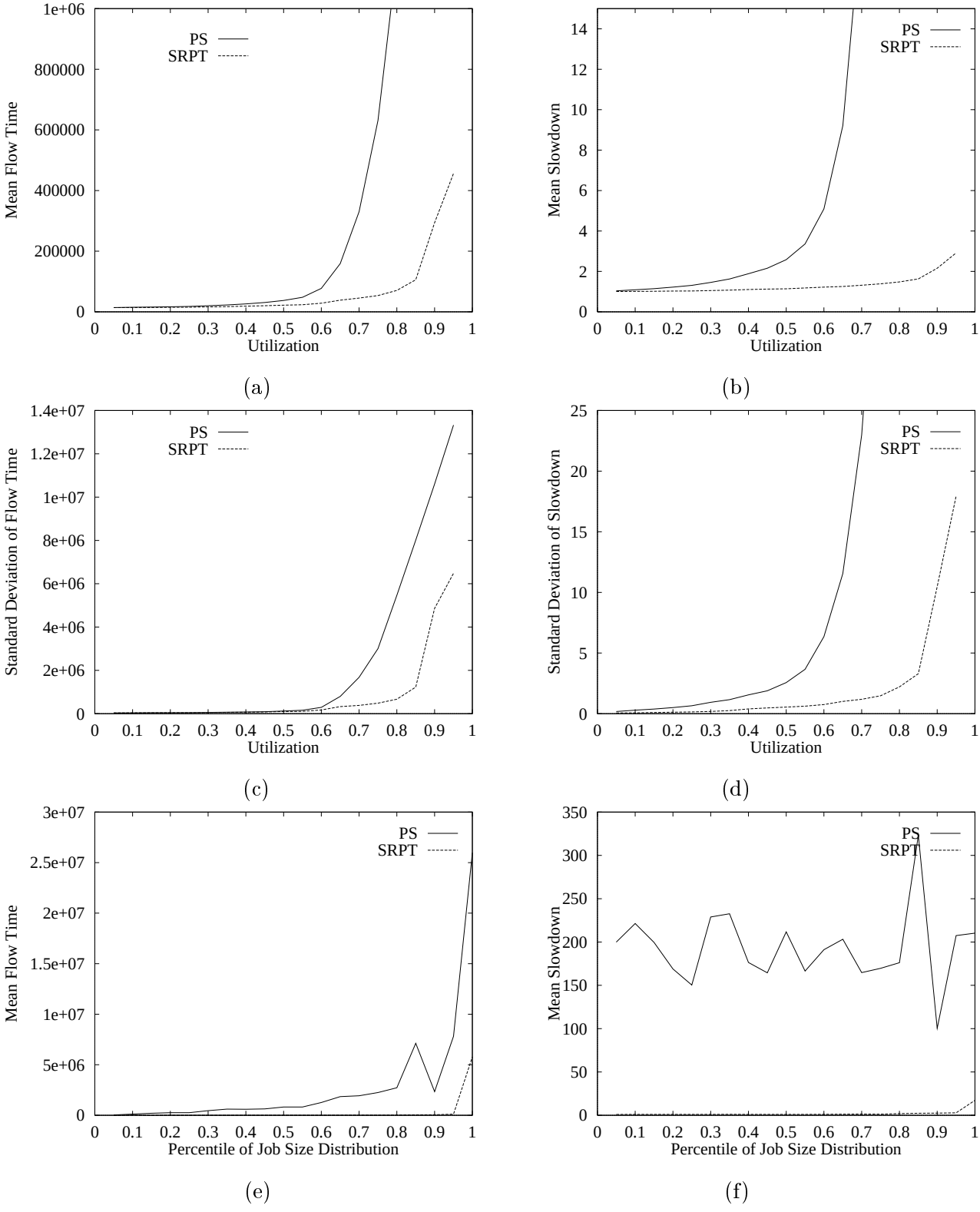


Figure 7: Performance Metrics for NCSA Trace. (a) Mean Flow Time as a function of server load; (b) Mean Slowdown as a function of server load; (c) Standard Devaiation of Flow Time as a function of server load; (d) Standard deviation of slowdown as a function of server load; (e) Mean flow time as a function of task size, assuming a server load of $\rho = 0.9$; (f) Mean slowdown as a function of task size, assuming a server load of $\rho = 0.9$.

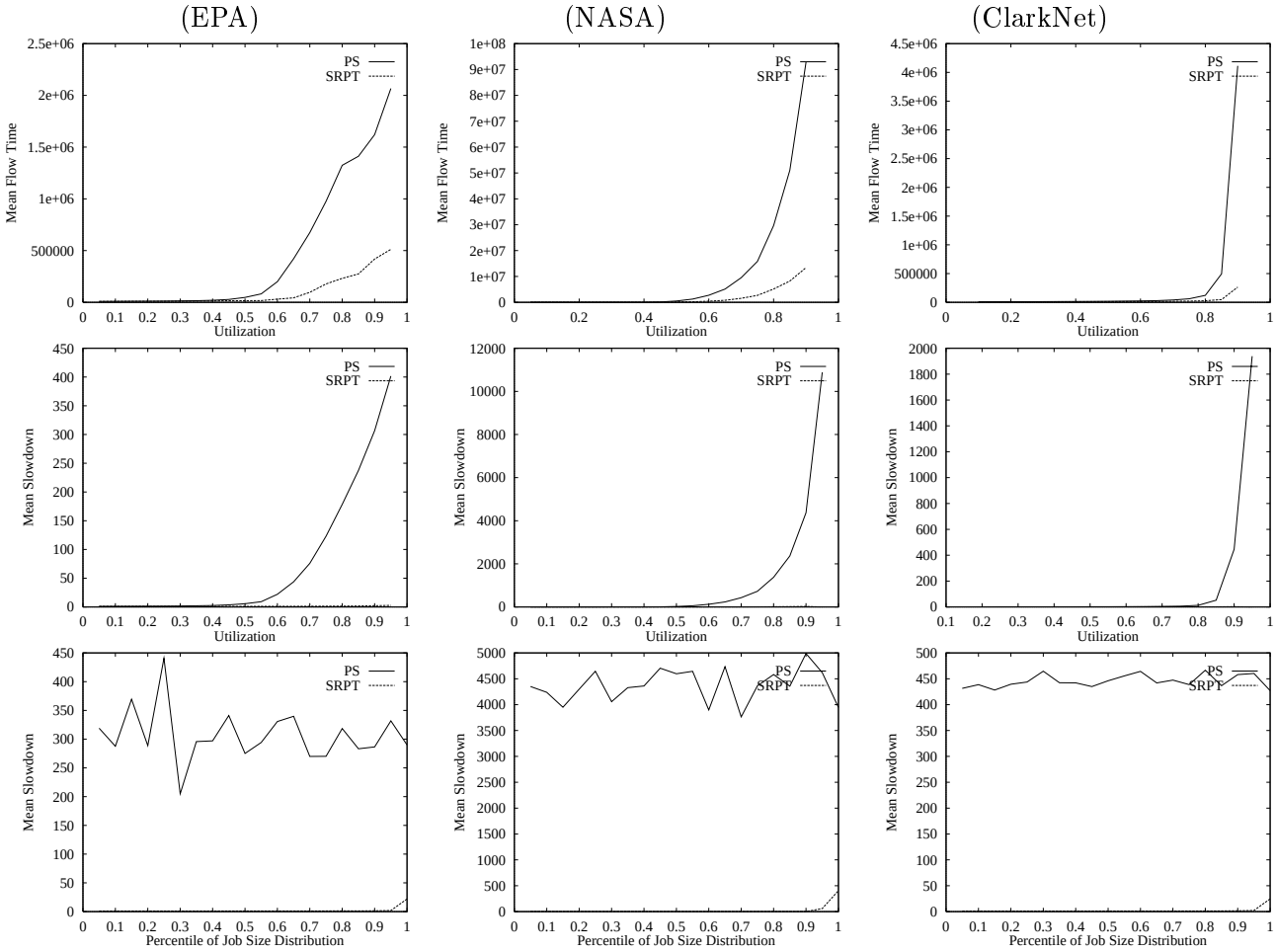


Figure 8: Peformance Metrics: left column: EPA Trace; center column: NASA Trace; right column: ClarkNet Trace. For each trace we show the mean flow time as a function of server load (top row), mean slowdown as a function of server load (middle row), and mean slowdown as a function of task size assuming server load of 0.9 (bottom row.)

6 Related Work

Currently, state-of-the-art Web servers do not explicitly make use of a task's size in scheduling the task. We have restricted our discussion to Web servers consisting of a single host. Such Web servers are often Unix or Windows NT machines whose scheduling policies can be approximated by Processor-Sharing (PS), as explained in Section 1.

There are many algorithms in the literature which are designed for the case where the task size is known. Good overviews of the single-node scheduling problem and its optimal solution are given in [14], [11], and [6]. Despite the fact that the file sizes are typically available to the Web server,

very little work has considered size-based scheduling in the Web.

One paper that does discuss size-based scheduling in the Web is that of Bender, Chakrabarti, and Muthukrishnan, [5]. This paper raises an important point: in choosing a scheduling policy it is important to consider not only the scheduling policy's performance, *e.g., mean slowdown*, but also whether the policy is fair, *i.e.* do some tasks *starve* (have particularly high slowdowns). That paper considers the metric *max slowdown* (the maximum slowdown over all tasks) as a measure of starvation. The paper proposes a new algorithm, *Dynamic Earliest Deadline First (DEDF)*, designed to perform well on both the mean slowdown and max slowdown metric. The DEDF algorithm is a *theoretical* algorithm which cannot be run within any reasonable amount of time (it requires looking at all previous arrivals), however it has significance in being the first algorithm designed to simultaneously minimize max slowdown and mean slowdown.

The Bender *et. al.* paper recommends against the use of SRPT, claiming that SRPT leads to starvation. They point out that there exist worst-case inputs on which SRPT will have unbounded max slowdown; however we show that this isn't representative of SRPT's performance in practice. The paper also considers a trace-driven simulation of a Web server and makes the point that DEDF has a lower starvation level than SRPT. The paper does consider a few heuristics based on DEDF which are implementable, however, the performance of those more practical algorithms at high load is about the same as SRPT with respect to max stretch and significantly worse than SRPT with respect to mean slowdown.

7 Conclusion

This paper proposes the idea of scheduling HTTP requests at a Web server in SRPT order. After justifying why the sizes (service demands) of the HTTP requests are in fact known in most cases, the paper goes on to defend SRPT in the area for which it has received the most criticism: starvation of large tasks. The paper shows that the starvation criticism is justified with respect to many task size distributions. However, the paper shows that in the case of Web workloads, which have a heavy-tailed highly-variable task size distribution, starvation is not an issue and in fact is more than an order of magnitude lower than starvation seen in typical scheduling policies currently used in Web servers.

All results in this paper are obtained both via analysis and via a trace-driven simulation. The analysis assumes an empirically-derived workload distribution and Poisson arrivals. The trace-driven simulation uses 4 different traces of HTTP requests arriving at Web servers. The results

from the traces are even more dramatic than those from analysis since the bursty arrival process negatively impacts performance under traditional time-sharing scheduling, however it does not have much of an effect on SRPT.

The results in our paper come with some caveats. First, using processor-sharing as our comparison case ignores some subtleties of scheduling in current Web servers. Second, modeling a Web server as a single resource is a simplification of real servers. For these reasons, the precise values of our numerical results are not likely to match measurements in practice although we believe that our overall conclusions will remain valid. In addition, there are issues which will come up in implementation which we have not modelled. In particular, our assumption that the service demand associated with an HTTP request is proportional to file size may be affected by features of the server such as caching.

The results of this paper suggest much interesting future research. On the practical level, the current paper models servers as having a single resource. The next step is proposing minor modifications to SRPT which are designed to run in multi-resource systems. The step after that is actually implementing these scheduling policies in a Web server. We are currently pusuing these directions.

On the theoretical level, this paper shows that on Web workloads, SRPT is far superior to PS both with respect to performance and with respect to starvation. However, there is still some room for improvement both with respect to mean slowdown and with respect to starvation. This suggests the question of whether the SRPT policy can be further improved upon on these fronts by some other practical policy.

References

[1] M. Arlitt, R. Friedrich, and T. Jin. Performance evaluation of Web proxy cache replacement policies. *Proceedings of Performance Tools '98. Lecture Notes in Computer Science*, 1469:193–206, 1998.

[2] Martin F. Arlitt and Carey L. Williamson. Web server workload characterization: The search for invariants. *IEEE/ACM Transactions on Networking*, 5(5):631–645, 1997.

[3] Paul Barford, Azer Bestavros, Adam Bradley, and Mark Crovella. Changes in web client access patterns: Characteristics and caching implications. Technical report, Boston University Department of Computer Science, 1998.

[4] Paul Barford and Mark E. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of Performance '98/SIGMETRICS '98*, pages 151–160, July 1998.

[5] Michael Bender, Soumen Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[6] Richard W. Conway, William L. Maxwell, and Louis W. Miller. *Theory of Scheduling*. Addison-Wesley Publishing Company, 1967.

[7] Mark E. Crovella, Murad S. Taqqu, and Azer Bestavros. Heavy-tailed probability distributions in the World Wide Web. In *A Practical Guide To Heavy Tails*, chapter 1, pages 3–26. Chapman & Hall, New York, 1998.

[8] Shuang Deng. Empirical model of WWW document arivals at access links. In *Proceedings of the 1996 IEEE International Conference on Communication*, June 1996.

[9] Anja Feldmann. Impact of non-Poisson arrival sequences for call admission algorithms with and without delay. In *Proceedings of Globecom '96*, 1996.

[10] Daniel P. Heyman. Performance implications of very large service-time variances. In *Proceedings of the 1998 SPIE Conference on Performance and Control of Network Systems*, Boston, MA, November 2-4 1998.

[11] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. In *CRC Handbook of Computer Science*. 1997.

[12] Leonard Kleinrock. *Queueing Systems*, volume II. Computer Applications. John Wiley & Sons, 1976.

[13] S. Manley and M. Seltzer. Web facts and fantasy. In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*, 1997.

[14] M. Pinedo. *On-line algorithms, Lecture Notes in Computer Science*. Prentice Hall, 1995.

[15] Linus E. Schrage and Louis W. Miller. The queue $m/g/1$ with the shortest remaining processing time discipline. *Operations Research*, 14:670–684, 1966.