NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
NWU-CS-04-33
April 19, 2004

Effects and Implications of File Size/Service Time Correlation on Web Server Scheduling Policies

Dong Lu **Huanyuan Sheng** **Peter A. Dinda**

Abstract

Recently, size-based policies such as SRPT and FSP have been proposed for scheduling requests in web servers. SRPT and FSP are superior to policies that ignore request size, such as PS, in both efficiency and fairness given heavy-tailed service times. However, a central assumption that is usually made in implementing size-based policies in a web server is that the service time of a request is strongly correlated with the size of the file it serves. This paper shows how the performance of SRPT and FSP are affected by the degree of this correlation. We developed a simulator that supports both M/G/1/m and G/G/n/m queuing models. The simulator can be driven with trace data, which can be taken from the logs of modified Apache servers, or which can be produced by a workload generator we have developed that allows us to control the correlation. Using both trace data and generated data, we find that the degree of correlation has a dramatic effect on the performance of SRPT and FSP. In response, we propose and evaluate domain-based scheduling, a simple technique that better estimates connection times by making use of the source IP address of the request. Domain-based scheduling improves SRPT and FSP performance on web servers, particularly in regimes where correlation is low, thus making size-based policies such as these more broadly deployable.

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, ANI-0301108, EIA-0130869, and EIA-0224449. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

Keywords: Web server scheduling; Queuing models; Simulations; Correlation; SRPT, FSP

Effects and Implications of File Size/Service Time Correlation on Web Server Scheduling Policies

November 6, 2003

Dong Lu
Dept. of Computer Science
Northwestern University
Evanston, IL 60201
{donglu}@cs.northwestern.edu

Huanyuan Sheng
Dept. of IEMS
Northwestern University
Evanston, IL 60208
{h-sheng}@northwestern.edu

Peter A. Dinda
Dept. of Computer Science
Northwestern University
Evanston, IL 60201
{pdinda}@cs.northwestern.edu

ABSTRACT

Recently, size-based policies such as SRPT and FSP have been proposed for scheduling requests in web servers. SRPT and FSP are superior to policies that ignore request size, such as PS, in both efficiency and fairness given heavy-tailed service times. However, a central assumption that is usually made in implementing size-based policies in a web server is that the service time of a request is strongly correlated with the size of the file it serves. This paper shows how the performance of SRPT and FSP are affected by the degree of this correlation. We developed a simulator that supports both M/G/1/m and G/G/n/m queuing models. The simulator can be driven with trace data, which can be taken from the logs of modified Apache servers, or which can be produced by a workload generator we have developed that allows us to control the correlation. Using both trace data and generated data, we find that the degree of correlation has a dramatic effect on the performance of SRPT and FSP. In response, we propose and evaluate domain-based scheduling, a simple technique that better estimates connection times by making use of the source IP address of the request. Domain-based scheduling improves SRPT and FSP performance on web servers, particularly in regimes where correlation is low, thus making size-based policies such as these more broadly deployable.

1. INTRODUCTION

In a web server, requests continuously arrive to be serviced. A request requires a certain service time to be completed, time whose components include the CPU, the disk, and the network path. A request is queued when it arrives and remains in the system until it is complete, the total time from arrival to completion being the sojourn time or response time. Scheduling policies determine which requests in the queue are serviced at any point in time, how much time is spent on each, and what happens when a new request arrives. Common goals of the scheduling policy are to minimize the mean sojourn time (response time of the request), the average slowdown (the ratio of its response time to its size), and to behave fairly to all requests.

Many policies are possible. First Come First Served (FCFS) is a non-preemptive policy in which the requests are run to completion in the order in which they were received. A more common policy is Processor Sharing (PS), which is preemptive. In PS all requests in the queue are given an equal share of the web server's attention. Generalized Processor Sharing (GPS) generalizes PS with priorities. Often, FCFS can be combined with PS or GPS, with FCFS dispatching of requests from the queue to a pool of processes or threads that are collectively scheduled using PS or GPS. These polices ignore the service time of the request.

Recently, size-based scheduling policies, those that incorporate the service time of the request into their decisions, have been proposed for use in web servers. Harchol-Balter, et al, have proposed the use of the Shortest Remaining Processing Time (SRPT) scheduling policy in web servers [21, 22], showed how to incorporate it into actual implementations [20, 22], and studied how SRPT can help a web server to gain performance under both persistent and transient overload [38]. The Fair Sojourn Protocol (FSP) is a modified version of SRPT that has been proven to be more efficient and fair than PS given any arrival sequence and service time distribution [18].

SRPT has been studied since the 1960s. Shrage first derived the expression for the response time in an M/G/1 queue [36]. For a general queuing system (G/G/1) Schrage proved in 1968 that SRPT is optimal in the sense that it yields—compared to any other conceivable strategy—the smallest mean value of occupancy and therefore also of waiting and delay time [35]. Schassberger obtained the steady state appearance of the M/G/1 queue with SRPT policy in 1990. Perera studied the variance

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, ANI-0301108, EIA-0130869, and EIA-0224449. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

of delay time in $M/G/1/SRPT$ queuing systems and concluded that the variance is lower than FIFO and LIFO [33]. Bux introduced the SRPT principle into packet networks [12] in 1983, using the message size as the service time.

An objection to SRPT is that it is possible to design an adversarial workload in which SRPT leads to the starvation of large jobs [40]. In other words, SRPT can behave unfairly. However, under the workload models that are believed to be correct for web servers (A Poisson process modeling requests and a bounded Pareto distribution modeling the heavy-tailed file size distribution, M/G/1) SRPT performs very well [8].

In the *implementation* of size-based polices such as SRPT and FSP on a web server, the service time of the request is needed. The common assumption is that the service time is the size of the file being served, as this is very easy to discover when the request enters the system. More broadly, the assumption is that the service time is strongly correlated to the file size. In this paper, we examine the validity of this assumption, and the impact that the degree of correlation between file size and service time has on the performance of SRPT and FSP.

To evaluate this impact, we developed a simulator that can support PS, SRPT, and FSP in both M/G/1/m and G/G/n/m. The simulator operates on a trace of request arrivals, which can come either from an augmented Apache [1] web server log, or from a trace generator. The trace contains the request arrivals, the file sizes, and the actual service times in microseconds. We use traces that we have captured on our department-level web server, and traces captured by others on web caches. Our trace generator allows us to control the correlation coefficient between file size and service time in a trace.

We study G/G/n/m in addition to M/G/1/m because previous research [32, 16] has shown that Poisson processes are valid only for modeling the arrival of user-initiated TCP sessions such as the arrival of TELNET connections and FTP connections. HTTP arrivals are not Poisson. That is because HTTP document transmissions are not entirely initiated by the user: the HTTP client will automatically generate a series of additional requests to download embedded files, thus resulting in a more bursty process. Previous work [16] pointed out that the aggregated interarrival times of HTTP requests can be modeled with a heavy-tailed Weibull distribution.

Crovella, et al found that WWW traffic showed self-similarity and proposed possible explanations for the phenomenon [15]. This work also pointed out that many characteristics of web can be modeled using heavy-tailed distributions, including the distribution of transfer times, the distribution of user requests for documents, the underlying distribution of documents sizes available in the web, and the interarrival time of requests. Barford, et al built the discovered web server workload characteristics into SURGE [10], a representative synthetic analytic workload generator.

There has been significant work on the G/G/n queuing model. Tabet-Aouel, et al gave analytic approximations for the mean sojourn time of P ($P \geq 2$) priority classes in a stable G/G/c/PR queue with general class interarrival and service time distributions and c ($c \geq 2$) parallel servers under pre-emptive resume (PR) scheduling [29]. Boxma, et al considered a GI/G/1 queue in which the service time distribution and/or the interarrival time distribution has a heavy tail, i.e., a tail behavior like t^{-v} with $1 \leq v \leq 2$, such that the mean is finite but the variance is infinite. Depending on whether the service time distribution is heavier than that of the interarrival time distribution, they concluded that the stationary waiting time can be modeled as either a Kovalenko distribution or a negative exponential distribution [11]. Xia, et al analyzed the asymptotic tail distribution of stationary virtual waiting times in a single-server queue with long-range dependent arrival process and subexponential service times [44]. However, we are aware of no analytical results on G/G/n/m for SRPT or FSP scheduling in regimes where interarrival times and service times are heavy-tailed. The work we describe in this paper is based on measurement and simulation.

Using our infrastructure, and measured and synthesized trace data, we address the following questions:

1. What is the actual degree of correlation between file size and service time in practice? (Section 2)
2. How does the performance of SRPT, FSP, and PS vary with the degree of correlation between file size and service time under G/G/n/m and M/G/1/m? (Section 3)
3. Is there a simple and low-overhead estimator for service time that would make SRPT and FSP on M/G/1/m and G/G/n/m perform better? (Section 4)

It is important to point out that our results in addressing questions 2 and 3 are largely independent of our results for question 1, and the algorithm we develop in response to question 4 provides benefits to SRPT and FSP over a wide range of possible answers to question 1.

Our measurements show that the assumption that file size and service time are strongly correlated is unwarranted—the correlation is, in fact, often rather weak. We speculate that the reason for this phenomenon is that the bottleneck for file transfer is in the network and different clients have different available bandwidth to the web server.

Our simulation experiments with generated traces show that the performance of file size-based SRPT and FSP are strongly related to the degree of correlation (R) between file size and service time. For low values of R, these scheduling policies perform *worse* than PS. For our web server traces, R is indeed low enough that both file-size based SRPT and FSP perform worse than PS. However, we find that SRPT can perform better than PS once R crosses a rather low threshold of about $R = [0.13, 0.18]$. In other words, SRPT needs some degree of correlation, but not much. As R increases, its performance continues to improve.

These results led us to believe that a better estimator for service time was needed. We refer to our estimator as a domain estimator, and the use of our domain-based estimator with a

Queuing Model	Description
$M/G/1/\mathrm{m}$	Poisson arrival process; General service time distribution; Single server ; Limited queue capacity m.
$G/G/\mathrm{n}/\mathrm{m}$	General arrival process (Pareto and Weibull); General service time distribution; n servers ; Limited queue capacity m.

Figure 2: Queuing models used in the paper. Both Pareto and Weibull service time distributions are considered.

size-based scheduling policy such as SRPT or FSP as *domain-based scheduling*. The basic idea is to use the high order k bits of the source IP address to assign the request to one of 2^k domains. For each domain, we estimate the service rate (file size divided by service time) based on all previous completed transfers to the domain. The service rate is then used to estimate the service time of a new request based on its file's size. Based on our traces, there is a strong relationship between the correlation of these estimates and the actual service time, which grows with the number of bits k used. In short, by choosing k appropriately, we can create enough correlation to make SRPT and FSP perform well. Surprisingly, k can be kept relatively small, making the implementation of domain-based scheduling feasible and fast. Throughout the paper, we refer to the scheduling policies as listed in Figure 1, and refer to the queuing models used as listed in Figure 2.

2. IS FILE SIZE A GOOD INDICATOR OF SERVICE TIME?

Size-based SRPT scheduling appeared in digital communication networks research in 1983 [12]. In this context, the service time was taken to be equal to the transmission time of a message, which is proportional to the length of the message stored in the node buffers. A web server serving static requests appears superficially similar in that it transmits files to the client. However, there are differences. First, in the digital communication network context, the work represented by the service time is pushing the bits of the message onto the wire, while for the web server context, the work involves end-to-end cooperation along an entire shared heterogeneous path. Although most transfers are likely to be dominated by the bottleneck bandwidth in the path and the latency of the path, there are multiple possible bottlenecks along the path and they can vary with time due to packet losses and congestion. Second, the disk(s), memory system(s), and CPU(s) of the web server and the client are also potential bottlenecks. These complexities suggest that the service time of a request may not be proportional or even well correlated with the size of the file it serves.

There are several possible definitions for service time in the web server context. For example, we could focus on a bottleneck resource on the server, such as the CPU, and define the service time as the total CPU time needed to execute the request. Alternatively, we could treat the CPU, disk, and network link of the server as a single resource and consider the total non-blocked time of a request on it. We could also take a holistic view and consider it the time spent on the bottleneck resource on the path from server to client. We take the position that the service time of a request is the time that the combination of server, client, and network would take to finish the request given no other requests in the whole end-to-end system (no load on any resource). In the following sections, we use this definition and argue that our measurement methodology measures it by verifying that the loads on the resources of the end-to-end system that we measure are miniscule.

To measure correlation between file size and service time we use the correlation coefficient (Pierson's R) [6]. The correlation between two random variables X (file size) and Y (service time) is

$$R(X,Y) = \frac{Cov[X,Y]}{\sqrt{Var[X] \times Var[Y]}} \quad (1)$$

where $Cov[X,Y]$ is the covariance of X and Y, $Var[X]$ is the variance of X and $Var[Y]$ is the variance of Y. R is in the range $[-1,1]$. Its absolute value denotes the strength of correlation. $abs[R] = 0.5$ indicates that $R^2 = 25\%$ of the variance of Y can be explained by X. Our results in this paper show only positive correlation $R > 0$. We estimate R using sample variance and covariance. A confidence interval can be computed for R based on the sample. For our results, the sample size is quite large, leading to intervals that are tiny compared to the differences we point out for quite small p-values. Hence, we do not plot them.

To answer the question posed by this section, we examine R values for a large trace acquired by us from a typical web server, as well as 70 traces collected from web cache servers. The main conclusion is that R can vary considerably from server to server, and can be quite small. $R = 0.14$ for our web server trace, while the web caches have R evenly distributed in the range $[0.12, 0.61]$. In subsequent sections, we use our web server trace to drive our simulation. However, we also use synthetically generated traces in which we can control R directly. While many web server traces are available, none that we could find record the actual service time of the request and thus are not useful for the purposes of our study.

2.1 Measurement on a typical web server

We modified the code of the Apache log module so that it records the *response time* of each request with microsecond granularity (using the IA32 cycle counter to measure time). Under extremely low load conditions, as we document below, this time is *equivalent* to the service time according to our definition above.

We deployed the module on our department-level web site. We collected data from September 15, 2003 to October 19, 2003. This trace includes approximately 1.5 million HTTP requests, among which less than 2% are dynamic PHP requests that collectively took less than 1% of the total service time recorded. $> 98\%$ of our requests and $> 99\%$ of the service time in the trace are for static pages. Hence, our web sever is dominated by static web content. Others claim that static content dominates web traffic [25, 24, 22] and thus our results are comparable to theirs. The requests originated from 110 "/8" IP networks, 7220 "/16" IP networks and 31250 "/24" IP networks spread over the world. We claim that this server is typical. However, the conclusions of this paper are also supported by other measured traces and generated traces.

Scheduling Policy	Description
PS	Processor Sharing scheduling policy.
FSP	Ideal Fair Sojourn Protocol, service times are known exactly.
SRPT	Ideal Shortest Remaining Processing Time, service times are known exactly.
FSP-FS	File size-based Fair Sojourn Protocol, file sizes are used as service times.
SRPT-FS	File size-based Shortest Remaining Processing Time, file sizes are used as service times.
FSP-D	Domain-based Fair Sojourn Protocol, estimated service times are used as service times.
SRPT-D	Domain-based Shortest Remaining Processing Time, estimated service times are used as service times.

Figure 1: Scheduling policies used in the paper.

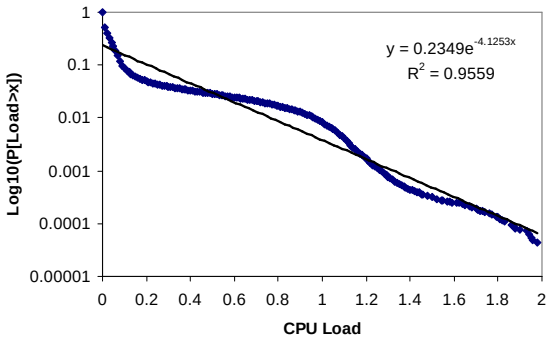


Figure 3: Complementary distribution of CPU load on the web server.

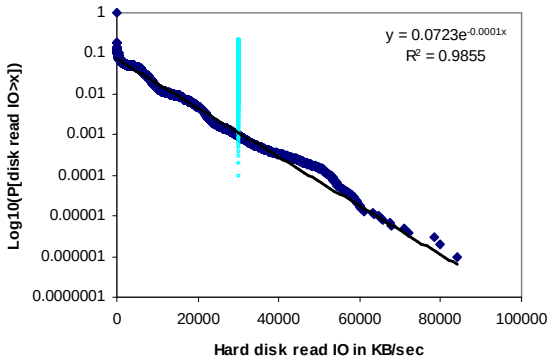


Figure 4: Complementary distribution of hard disk read I/O on the web server.

The bottleneck resource of a request in this trace is hardly ever the CPU of the server. The web server is a dual processor Pentium IV Xeon machine running Red Hat Linux 7.3. CPU load is defined as the exponentially averaged number of jobs in the run queue of the OS kernel scheduler (the Unix load average), The machine can serve two CPU intensive applications with full CPU cycles. Figure 3 plots the complementary distribution of the CPU load during the period of the traces with the vertical axis in log scale to better show details. This distribution can be modeled with a exponential distribution with $R^2 \approx 0.96$. Figure 3 shows that the probability $P[CPUload > 2]$ is minuscule. The memory system is also clearly not a bottleneck based on these results as significant cache stalls would show up as increased load.

The bottleneck resource of a request in this trace is hardly ever

Char read	Block read	WebRead	Char write	Block write
23604.2	1399254.2	29879.3	16777.9	50355.8

Figure 5: Hard disk to memory bandwidth, KB/sec.

the disk system of the server. The machine's file systems reside on a NFS-mounted (over private gigabit Ethernet SAN) RAID 5 storage server. Figure 4 shows the complementary distribution of the storage system reads during the period of the trace. The vertical axis is log scale to show details. The distribution can be modeled with a exponential distribution with R^2 close to 0.99.

We benchmarked the storage system using Bonnie, which is a widely used utility that measures the performance of Unix file system operations that an application sees [2]. Bonnie reads and writes a 100 MB file (marked uncacheable) by character or by block. Both sequential and random access are tested. Random block and character throughput give us upper and lower bounds on the performance of file system I/O that Apache sees. We also wrote our own benchmark (WebRead) to get a sense of the typical read performance that Apache sees. WebRead simply reads the files in our access log, in order, as fast as possible. Not surprisingly, the WebRead performance is in between the character read and block read benchmark given by Bonnie. WebRead's performance is shown Figure 4 as a vertical line, while all the results are shown Figure 5. We can see that probability of read throughput being larger than the throughput measured in the WebRead benchmark is < 0.001, while no recorded read throughput was larger than Bonnie's block read benchmark. Notice also that the highest throughputs seen are lower than the 125 MB/s throughput limit of the Ethernet SAN, hence the SAN is also not a bottleneck.

As it is clear that the CPU, memory, and disk systems are not bottlenecks, if there is any bottleneck it is in the network or the client. Based on many earlier measurements of load behavior on clients that indicate their resources spend much of their time idle [27, 17], it is extremely unlikely for a client to be the bottleneck. If there is any bottleneck, it is in the network path to the client, which agrees with earlier work [22]. Given the low rate of requests, it is highly likely that a single request would perform similarly to the requests in our trace. Hence, the high-resolution response time that we record in the Apache log is a close approximation of the service time as defined above. Obviously, there are situations where CPU or disk can become bottlenecks, such as in virtual server configuration in which one physical server hosts several web sites, or on a web server that hosts database-based dynamic web content.

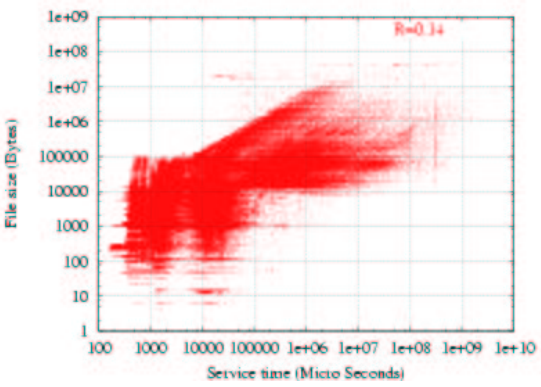


Figure 6: File size versus service time in web trace.

File Size	R
$X < 30$ KB	0.0616
$30 \leq X < 500$ KB	0.1121
$X > 500$ KB	0.1033

Figure 7: R depends on file size.

Given the provenance of the trace, we can now use it to answer our question. Figure 6 is a log-log scatter plot of file size versus service time. Visually, we can see hardly any correlation between file size and service time. File transfer times vary over several orders of magnitude with same file size. *Over the entire 1.5 million requests in the trace, we find that R is a very weak 0.14.* R varies slightly with file size, as can be seen in Figure 7.

Within a domain, R is larger. We define precisely what we mean by a domain and connect it with CIDR in Section 4. Here, simply consider it as a single network that may be recursively decomposed into subnetworks. For example, Figure 8 is a log-log scatter plot of file size versus service time for requests originating with a single "/16" IP network, where the network address is 16 bits. $R = 0.25$ for this situation. As the domain grows smaller (has fewer IP addresses, or more bits representing its network address), R grows larger. For example, if we focus on a particular "/24" LAN subnet (24 bit network address) that is contained within the previous network, $R = 0.39$. We speculate that the reason for this behavior is that network bandwidth heterogeneity from the server to the clients of a domain decreases as the size of the domain decreases. This provides a different, but compatible, explanation for earlier findings [8] that file size-based SRPT scheduling can decrease mean sojourn time by a factor of 3-8 over PS in a LAN for load higher than 0.5, but can only decrease the mean sojourn time by 25% on the WAN. In Section 3, our simulations show that when $R \approx 0.4$, as on the example LAN, file size-based SRPT outperforms PS by a factor of about 3, but when the $R < 0.2$ (recall that our web trace showed $R = 0.14$) file size-based SRPT performs similar to PS and can perform worse than PS if R goes down further, when file sizes are hardly any indicator of service times at all.

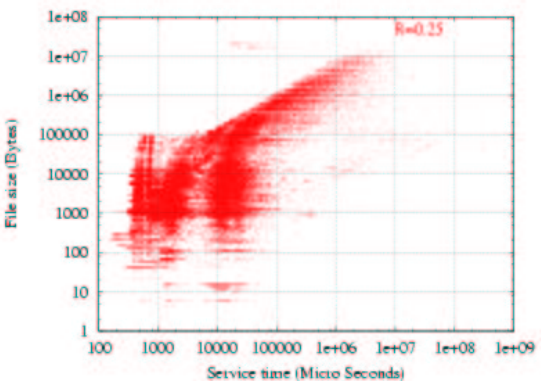


Figure 8: File size versus service time for particular /16 network.

We are actively acquiring additional traces, but this is difficult because web server modifications are necessary to acquire fine grain service times. Many available traces, such as those from the Internet Traffic Archive [3], our institution's other web servers, and others provide only file size, not service time and thus are unsuitable for our work. We have, however, acquired many traces from web caches, described next, and built a trace generator that allows us to control R as well as the distributions of service time and interarrival time, described later.

2.2 Measurement on web caches

We examined 70 sanitized access logs from Squid web caches, made available through the ircache site [4]. These traces contain fine grain service times in addition to file sizes. Internet object caching stores frequently requested Internet objects (i.e., pages, images, and other data) on caches closer to the requester than to the source. Clients can then use a local cache as an HTTP proxy, reducing access time as well as bandwidth consumption.

Squid is a high-performance proxy caching server for web clients. Unlike traditional caching software, Squid handles all requests in a single, non-blocking, I/O-driven process [5], making it very easy to determine the service time of a request. Squid is similar to a web server in that it also accepts HTTP requests and sends back requested files, but it is different in that the Squid servers form a overlay network that uses the Internet Cache Protocol (ICP) to perform server selection for web clients and load balancing among the cache servers [42, 41]. A client sees that it typically receives a reply from the nearest cache server, while from the Squid cache servers' points of view, the Internet is divided into several regions with each cache server typically serving requests for a specific region.

Because a single Squid cache serves clients largely from one region of the Internet, the bandwidth heterogeneity to the clients is likely to be less than that seen by a web server, which services clients regardless of region. This, we believe, should lead to larger R being measured on Squid caches than on web servers. The partitioning of the network as seen from the web server into domains that we describe in Section 4 builds on this observation.

While we cannot (and do not) use web cache traces as prox-

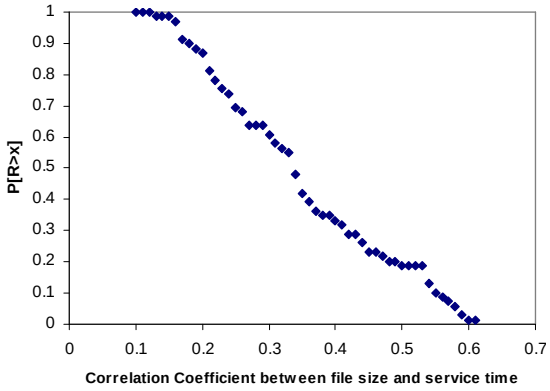


Figure 9: Complementary distribution of R in web cache traces.

ies for web server traces, it is instructive that R on the caches is also rather weak. Figure 9 shows a complementary distribution plot of the R values in the 70 traces. The traces were collected from 10 squid web cache servers over 7 days Each trace contains from 0.1 to 1.1 million requests. The smallest $R = 0.12$, while the largest $R = 0.61$. The mean is 0.34 with standard deviation 0.13. Given that we expect that R for web servers will be lower than R for web caches by the reasoning in the previous paragraph, that measured Rs on web caches are low suggests that R on web servers is likely to be low as well.

In combination with the low R seen on our web server trace, we believe that we can now answer the question posed by this section in the negative: *The correlation between request file size and service time on web servers is weak.*

3. HOW DOES PERFORMANCE DEPEND ON CORRELATION?

We have seen that request service time on web servers and caches is not strongly correlated with request file size. Here, we investigate, via simulation, how the degree of this correlation (R) affects the performance of size-based scheduling policies (SRPT and FSP, where actual service time is known a priori, and SRPT-FS and FSP-FS, where the file size is used as the service time) and compare with a size-oblivious policy (PS). Our metrics are the mean sojourn time and mean queue length. We find that for these metrics, the performance of SRPT-FS and FSP-FS is dramatically affected by R, falling below that of PS for low R values. Furthermore, for a fixed low R, as we measured on our web server, increasing load causes increasing divergence of performance of the file-size based policies (SRPT-FS, FSP-FS) from their ideal versions (SRPT, FSP).

3.1 Simulator and traces

Our simulator supports both M/G/1/m and G/G/n/m queuing systems. It is driven by a trace in which each request contains the arrival time, file size, and service time. In addition to the web server trace described in the previous section, we also use synthetic traces generated with interarrival times from exponential, bounded Pareto, and Weibull distributions, and file sizes from bounded Pareto and Weibull distributions, and service times from bounded Pareto. In the synthetic traces, we directly control the correlation, R, between file size and service time, as described later. Each simulation throughout the rest of the paper is repeated 20 times.

We validated our simulator by (a) checking stability and assuring that Little's law is never violated on each run, using effective arrival rate is appropriate for limited queue capacity, (b) repeating the simulations described in Friedman and Henderson's FSP paper [18] with nearly identical results, and (c) comparing our simulation results with the analytic results of Bansal and Harchol-Balter's SRPT fairness paper [8].

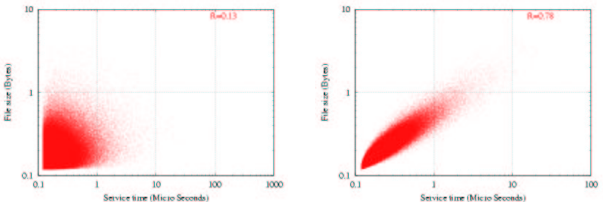
3.2 Controlling R in synthetic traces

Given some parametric distribution (exponential, bounded Pareto or Weibull here) and a target correlation coefficient R, we generate pairs of random numbers where each number of the pair is chosen from its required distribution and where the two numbers of the pair are correlated to degree R. To do this, we use a simplified Normal-To-Anything (NORTA) method. The basic ideas and proofs behind NORTA were developed by Cario and Nelson [13]. Given the distributions $dis_{filesize}$ and $dis_{servicetime}$, our target correlation coefficient R and our sample size N, the following algorithm generates N pairs:

1 Set $\rho = R$
2 Generate two independent random numbers $x_1, x_2 \sim N(0,1)$.
3 let $y_1 = x_1, y_2 = \rho \times x_1 + (1-\rho^2) \times x_2$
4 let $u_1 = NormCDF(y_1, 0, 1)$, $u_2 = NormCDF(y_2, 0, 1)$ where $NormCDF(y_i, 0, 1)$ is the CDF value of a standard normal distribution at y_i for $i = 1, 2$. It can be shown that $u_i \sim U[0,1], i = 1, 2$
5 let $filesize = F^{-1}_{dis_{filesize}}(u_1)$, $servicetime = F^{-1}_{dis_{servicetime}}(u_2)$ where $F_{dis_{filesize}}$, $F_{dis_{servicetime}}$ are the CDFs of our desired distributions for file size and service time respectively. F^{-1}_{dis} is the inverse of F_{dis}.
6 Repeat steps 2-5 N times generating N pairs $\{(filesize_j, servicetime_j)\}$. $\{filesize_j, j = 1, \ldots, N\}$ and $\{servicetime_j, j = 1, \ldots, N\}$ are two correlated random numbers each following their own distributions.
7 Compute the correlation coefficient of $\{filesize_j\}$, $\{servicetime_j\}$ and call it ρ_{temp}. If $\rho_{temp} > R$, then decrease ρ and go to step 2. If $\rho_{temp} < R$, then increase ρ and go to step 2. If $\rho_{temp} \approx R$ then stop.

Figure 10 gives some examples of file size/service time pairs generated for different values of R.

To show the correctness of this algorithm, we can try following analysis: First, it is easy to see that $y_1, y_2 \sim N(0,1)$ and $u_1, u_2 \sim U[0,1]$, thus $filesize \sim dis_{filesize}$ and $servicetime \sim dis_{servicetime}$. Second, it can be shown that y_1 is correlated with y_2 and thus so is u_1 with u_2. Intuitively it follows that $\{filesize_j\}$ and $\{servicetime_j\}$ are correlated as well. Cario and Nelson showed that (1) ρ_{temp} is a nondecreasing continuous function of ρ, and (2) ρ_{temp} and ρ share the same sign. These properties guarantee the termination of the above simplified NORTA algorithm and let us bound the values of R that can be achieved by NORTA. If we sample ρ from 0 to 1,



(a) R=0.13 (b) R=0.78

Figure 10: Examples of generated file size/service time pairs.

	α	Lower bound	Upper bound
Interarrival	1.32	0.07177	5×10^5
Service time	2.0	0.1188	10^5

Figure 11: Bounded Pareto Distribution Parameters.

we can estimate the range of ρ_{temp}, producing a set of sets of pairs, ordered with increasing R as a side effect. This is exactly how we generated correlated random pairs of file size and service time. Depending on the structures of different distributions, ρ_{temp} may not always take a full range of $[0, 1]$, which is why some of the results we show here have a restricted range of R.

3.3 Simulation with synthetic traces

To study the effects of the correlation R between file size and service time on the performance of SRPT-FS and FSP-FS, we generated traces with controlled correlation as described in the previous section. We used bounded Pareto distributions for both file size and service time. For the arrival process, we consider Poisson arrivals (exponential interarrival times), heavy tailed Pareto arrivals, and heavy tailed Weibull arrivals. For all the simulations of this section, the load (mean arrival rate divided by mean service rate) is 0.9, and queue capacity is 5000. A single server is assumed. Multiple servers are considered in the next section. Figure 11 shows the parameters of the bounded Pareto distributions used for the simulations shown in Figure 12 and Figure 13. We used identical Bounded Pareto distribution for both file size and service time distributions.

The scheduling policies used (SRPT, SRPT-FS, FSP, FSP-FS, and PS) are described in Figure 1. Each graph data point we show data point represents several simulations, each of 0.5 million requests.

Figure 12 shows the effects of R on the mean sojourn time of different scheduling polices with a Poisson arrival process, corresponding to M/G/1/m queuing model, the interarrival mean used to generate Poisson process is 0.278. Figure 13 shows the effects of R on the mean sojourn times of different scheduling polices with a heavy tailed Pareto arrival process, corresponding to G/G/1/m queuing model. The mean sojourn time of the policies with a heavy tailed Weibull arrival process is similar to that of Pareto arrival except that both SRPT-FS and FSP-FS need a slightly bigger R to work better than PS.

FSP-FS is very sensitive to R. For Poisson arrivals, its performance is not stable until $R > 0.8$, when its performance converges with that of PS, and slowly overtakes it. For Pareto arrivals, its performance exceeds that of PS only when $R > 0.2$.

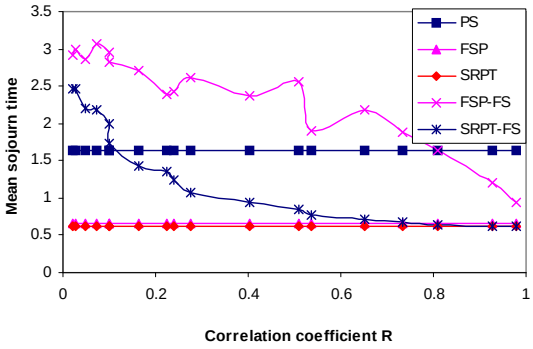


Figure 12: Mean sojourn time versus R, synthetic traces, M/G/1/m, Pareto service times, Poisson arrivals.

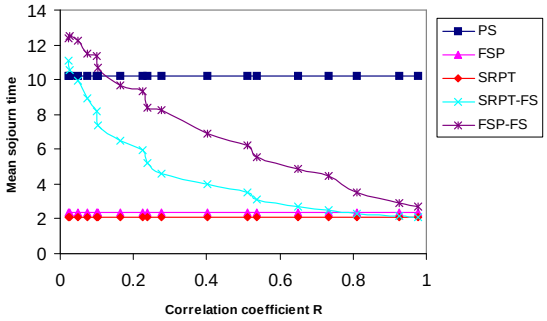


Figure 13: Mean Sojourn Time versus R, synthetic traces, G/G/1/m, Pareto service times, Pareto arrivals.

The performance of SRPT-FS increases much more quickly with increasing R. When R is very small, SRPT-FS and FSP-FS essentially behave like a random scheduling policy, with difficulty to predict performance. When R exceeds a low threshold, SRPT-FS performance exceeds that of PS in both M/G/1/m and G/G/1/m. The threshold is in the range $[0.13, 0.18]$. Beyond this point, SRPT-FS's performance increases geometrically with increasing R. Recall that our web trace shows $R = 0.14$, which suggests that SRPT-FS performance will be similar to PS performance for the trace. The figures clearly show that SRPT performance is strongly tied to R, even at high values of R. Improvements in estimating actual service time can dramatically improve SRPT for a wide range of R.

The lack of accurate service time information has been an important reason why SRPT is not widely deployed [37, 8]. However, SRPT appears to be better suited than FSP to working under conditions where it is difficult to estimate service time, and its performance scales nicely as more accurate service time estimates are available. We show in Section 4 that such improvements are possible for web servers using the simple, efficient techniques.

3.4 Simulation with web server trace

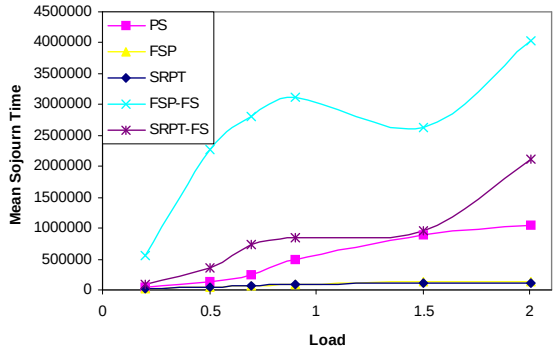


Figure 14: Mean sojourn time versus queue load for web trace, M/G/1/m, Poisson arrivals

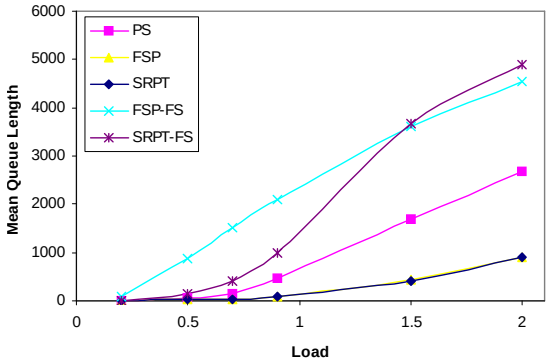


Figure 15: Mean queue length versus queue load for web trace, M/G/1/m, Poisson arrivals.

Here we consider the performance of SRPT, SRPT-FS, FSP, FSP-FS, and PS on the measured web server trace ($R = 0.14$) described in Section 2.1. The mean service time is 1250 microseconds. The scheduling policies are described in Figure 1. Note that although our web server trace represents very low load, here we vary the load in the system by controlling the arrival process of the requests represented in the trace. We make use of Poisson arrivals, Pareto arrivals, and Weibull arrivals and control their mean rate in order to control the load. Load control is important, because, as we discussed in Section 2.1, the load captured in the trace is rather low. The time units are microseconds throughout the rest of the paper.

We begin with M/G/1/m (Poisson arrivals, file sizes and service times as in the trace). Figure 14 shows the mean sojourn times of different scheduling policies with increasing load, while Figure 15 shows the mean queue lengths. In both figures, ideal SRPT and FSP perform very well and identically. However, SRPT-FS and FSP-FS both perform quite poorly, and their performance diverges dramatically from their ideal performance with increasing load. SRPT-FS and FSP-FS perform worse than SRPT and FSP in all our simulations.

Next, we consider G/G/1/m (Pareto arrivals, file sizes and service times as in the trace). Figure 16 shows the mean sojourn times of different scheduling policies with increasing

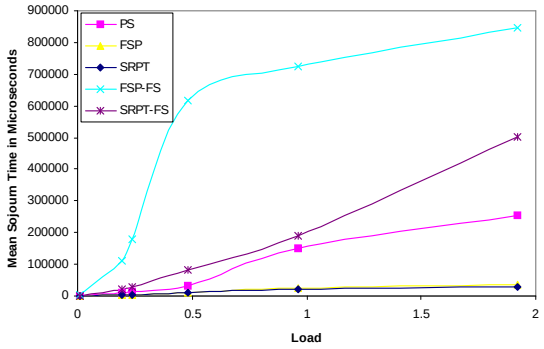


Figure 16: Mean sojourn time versus load, G/G/n/m, Pareto arrivals.

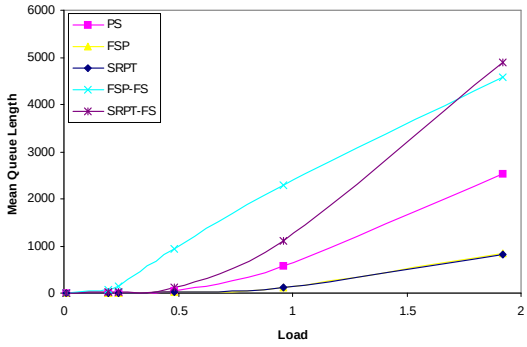


Figure 17: Mean queue length versus load, G/G/n/m, Pareto arrivals.

load, while Figure 17 shows the mean queue length of different scheduling policies with increasing load on the queue. In both figures, ideal SRPT and FSP perform very well, and identically. However, again, SRPT-FS and FSP-FS perform worse, and their behavior diverges from the ideal with increasing load.

Finally, we consider what happens if we fix the mean interarrival time and increase the number of servers, G/G/n/m (Pareto

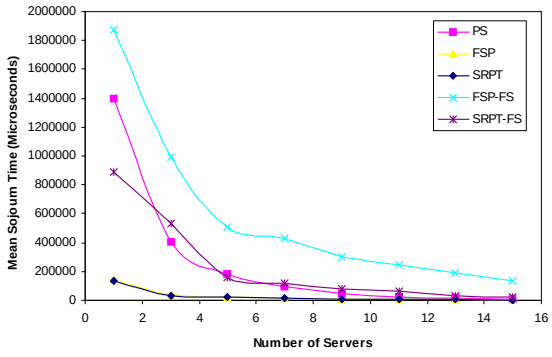


Figure 18: Mean sojourn time versus number of servers, G/G/n/m, Pareto arrivals, Mean interarrival time 162.5 microseconds.

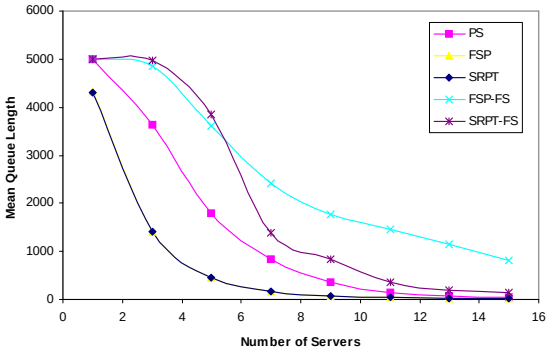


Figure 19: Mean queue length versus number of servers, G/G/n/m, Pareto arrivals, Mean interarrival time 162.5 microseconds.

arrivals, file sizes and service times as in the trace). We fixed the interarrival mean at 162.5 microseconds, and mean service time for each server at 1250 microseconds. Figure 18 shows the mean sojourn times of different scheduling policies with the increasing number of servers. Figure 19 shows the the mean queue length of different scheduling policies with increasing load on the queue. Again, we see that ideal SRPT and FSP perform very well and identically, but that SRPT-FS and FSP-FS perform much more poorly, especially with few servers. As the number of servers increases, the differentiation between the policies decreases. Not surprisingly, in what is effectively a low-load regime, there is not much difference between the policies.

For a queue with unlimited queue capacity, the mean sojourn time tends to be infinity if the load is over unity. But our simulator takes a limited queue capacity to reflict the reality, therefore, the server start to reject jobs when overloaded for some time (the queue is full) and both mean sojourn time and mean queue length are meanful. They represent the queue behavior of the server under transient overload.

We have also investigated a Weibull arrival process, where the interarrival times of requests in the trace are drawn from a Weibull distribution. The results are similar to those for the Pareto arrival process shown earlier.

Our simulations, using both synthetic traces and our measured web server trace have found that the performance of SRPT-FS and FSP-FS, SRPT and FSP where request file size is used as request service time, is highly dependent on the correlation R between file size and service time. With low enough R, performance can degrade so far that PS is preferable to either of these policies. Our trace shows such a low R. Furthermore, over wide range of R, which includes the range seen in the web cache traces we examined, increasing R dramatically increases performance for SRPT-FS. In the next Section, we describe and evaluate a better estimator for service time that uses file size, the network "domain" of the client, and past performance to the domain to produce more accurate service time estimates. Using these estimates, SRPT and FSP can be made to perform much better than simply using file size.

4. DOMAIN-BASED SCHEDULING

We have found that request file size and service time are weakly correlated and that the performance of size-based scheduling policies are strongly dependent on the degree of this correlation. Given these results, a natural question is whether there is a better service time estimator than file size, one whose estimates are more strongly correlated with actual service time. Such an estimator must also be lightweight, requiring little work per request. For this reason, we cannot use active probing techniques such as those used in tools like the Network Weather Service [43]. We also cannot use passive network-layer techniques, such as those used in Remos [26], because we do not have access to the network layer throughout the path. Instead, we use past web requests as our probes, similar in spirit to SPAND [39].

4.1 Statistical stability of the Internet

Domain-based scheduling relies on the Internet being statistically stable over periods of time, particularly from the point of view of the web server. Fortunately, there is significant evidence that this is the case. This evidence falls into two classes, routing stability and spatial and temporal locality of end-to-end TCP throughput.

Routing stability: Paxson [31, 30] proposed two metrics for route stability, prevalence and persistency. Prevalence, which is of particular interest to us here, is the probability of observing a given route over time. If a route is prevalent, than the observation of it allows us to predict that it will be used again. Persistency is the frequency of route changes. The two metrics are not closely correlated. Paxson's conclusions are that Internet paths are heavily dominated by a single route, but that the time periods over which routes persist show wide variation, ranging from seconds to days. However, 2/3 of the Internet paths Paxson studied had routes that persisted for days to weeks. Chinoy found that route changes tend to concentrate at the edges of the network, not in its "backbone" [14]. Barford, et al measured the web performance in the wide area network and found that the routes from/to the client to/from a web servers was asymmetric, but very stable [9].

Spatial locality and temporal locality of end-to-end TCP throughput: Balakrishnan, et al analyzed statistical models for the observed end-to-end network performance based on extensive packet-level traces collected from the primary web site for the Atlanta Summer Olympic Games in 1996. They concluded that nearby Internet hosts often have almost identical distributions of observed throughput. Although the size of the clusters for which the performance is identical varies as a function of their location on the Internet, cluster sizes in the range of 2 to 4 hops work well for many regions. They also found that end-to-end throughput to hosts often varied by less than a factor of two over timescales on the order of many tens of minutes, and that the throughput was piecewise stationary over timescales of similar magnitude [7]. Seshan, et al applied these findings in the development of the Shared Passive Network Performance Discovery (SPAND) system [39], which collected server performance information from the point of view of a pool of clients and used that history to predict the performance of new requests. Myers, et al examined performance from a wide range of clients to a wide range of servers

and found that bandwidth to the servers and server rankings from the point of view of a client were remarkably stable over time [28]. Yin Zhang, et al [45] found that three Internet path properties, loss rate, delay and TCP throughput show various degrees of constancy and concluded that one can generally count on constancy on at least the time scale of minutes.

4.2 Algorithm

Although the Internet, web servers, and clients form a highly dynamic system, the stability we pointed out in the previous section suggests that previous web requests (the web server's access log) are a rich history which can be used to better estimate the service time of a new request. We assume that after processing a request we know (1) its file size, (2) the actual service time, and (3) the IP address of the client. Collecting this information is simple and efficient. Our goal is to develop an efficient estimator that uses a history of such requests, combined with the file size and IP address of the current request to determine the likely service time of the current request. The correlation R between the estimated service time and the actual service time should be higher than the correlation between file size and actual service time. Recall that R must exceed a threshold in order for SRPT to perform better than PS, and as R increases, the performance of SRPT increases.

Classless Inter Domain Routing (CIDR) [23, 34, 19] was proposed in 1993 as "a strategy for address assignment of the existing IP address space with a view to conserve the address space and stem the explosive growth of routing tables in default-route-free routers". The CIDR strategy has been widely deployed since 1993. "One major goal of the CIDR addressing plan is to allocate Internet address space in such a manner as to allow aggregation of routing information along topological lines". Consider a *domain*, a neighborhood in the network topology. The broad use of CIDR implies that routes from machines in the domain to a server outside the domain will share many hops. Similarly, the routes from the server to different machines in the domain will also have considerably overlap. This also means that the routes will be likely to share the same bottleneck network link and therefore have similar throughput to/from the server. The smaller the domain, the more the sharing.

The aggregation of CIDR is along a hierarchy of increasingly larger networks and is reflected in IP addresses. The first k bits of an IP address gives the network of which the address is a part, the first $k-1$ bits give the broader network that contains the first network, and so on. We exploit this hierarchy in domain-based scheduling, the algorithm of which is given below.

1. Use the high order k bits of the client IP address to classify the clients into 2^k domains, where the k bits are treated as the domain address.
2. Aggregate past requests to estimate the service rate (or representative bandwidth) for each domain. This can be done with several estimators, but our experiments show that the estimator $S_R = \frac{F_s}{S_t}$ performs the best. Here S_R is the representative service rate, F_s is the sum of the requested file sizes from the domain, and S_t is the sum of the service times for these requests. Notice that updating this estimate after a request has been processed is trivial: simply add the request's file size and service time to F_s and S_t, respectively (two reads, two adds, two writes). For each domain, we store F_s and S_t. An array of these pairs is kept, indexed by the domain address. The total state size is 2^{k+1} floating point numbers.
3. For each incoming client request, the web server first extracts the domain address, indexes the array and computes S_R for the domain. It then estimates the request's service time as $T_{estimate} = \frac{f_s}{S_R}$, where f_s is the request file size. The estimator requires a logical shift, two reads, a division, and a multiply. For a request from a heretofore unobserved domain, which occurs exactly once per domain, we simply use file size as the estimate.
4. Apply a size-based scheduling policy such as SRPT using the estimated service times. We suffix the scheduling policy with "-D": SRPT-D, FSP-D.

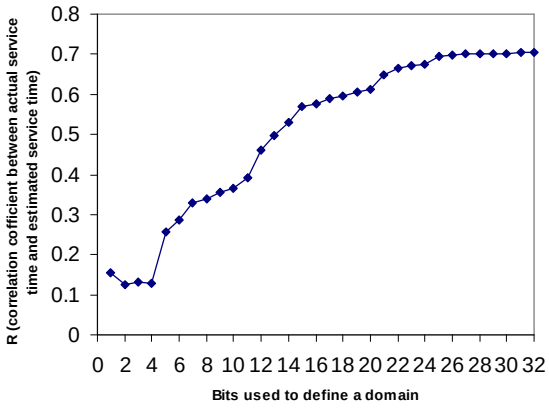


Figure 20: Correlation R versus number of bits used to define a domain k.

As we might expect, as domains become smaller (k gets larger), predictive performance increases, at the cost of memory to store the state. Figure 20 shows the relationship between k, the number of bits used to define a domain and the correlation R between the actual service time and estimated service time. The figure is derived from our web server trace. R jumps to 0.26 with $k = 5$ bits, beyond the threshold at which SRPT begins to perform better than PS. Notice that this is a mere 32 domains (state size of 256 bytes with 4 byte floats). After $k = 24$ bits, there are only very small increases of R, probably because at this point we have divided the Internet into LANs, where each machine on a LAN shares a common route to every other machine in the Internet, and thus shares the same bottlenecks. The maximum R we were able to achieve was 0.704.

4.3 Performance evaluation

To evaluate domain-based scheduling (SRPT-D and FSP-D, also see Figure 1), we use the methodology of Section 3.4. We replay our web trace with Poisson, Pareto, and Weibull arrivals to control load. We vary k, the number of high-order bits

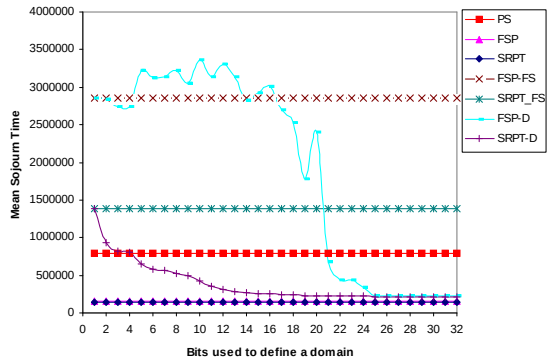


Figure 21: Mean sojourn time versus k for web trace, domain-based scheduling, M/G/1/m, Poisson arrivals with Mean Interarrival time 1269, load 0.90.

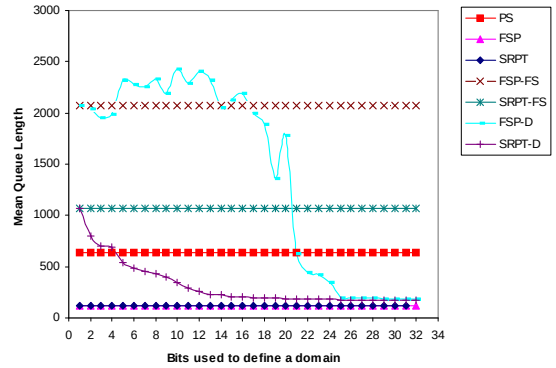


Figure 22: Mean queue length versus k for web trace, domain-based scheduling, M/G/1/m, Poisson arrivals with Mean Interarrival time 1269, load 0.90.

we use to define a domain. In this section, we used a longer trace than in the previous sections. The R between file size and service time remains unchanged, but the mean service time is slightly lower, which is 1145 instead of 1250 microseconds.

Figures 21 and 22 show the mean sojourn time and mean queue length of all the scheduling policies with Poisson arrivals. Notice that PS, FSP, SRPT, FSP-FS, and SRPT-FS are flat lines. PS ignores service time. FSP and SRPT have exact knowledge of the service times (they represent the ideal performance of these policies). FSP-FS and SRPT-FS use file size as a proxy for service time (representing current practice). Notice that as we increase the number of bits k used to define a domain, the performance of SRPT-D and FSP-D first exceeds that of PS and finally converges to near the ideal performance.

While SRPT-D's performance increases continuously, with diminishing returns, with increasing k, FSP-D is rather insensitive until $k = 16$ to 24 bits, at which point its performance jumps dramatically and comes very close to SRPT-D's. Since R doesn't increase much beyond $k = 24$ bits, as we might expect, the performance of SRPT-D and FSP-D plateaus.

Figures 23 and 24 show the mean sojourn time and mean

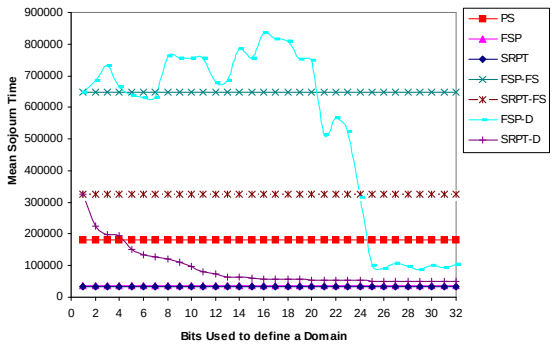


Figure 23: Mean sojourn time versus k for web trace, domain-based scheduling, G/G/1/m, Pareto arrivals with $\alpha = 1.32$, Lower bound 84, Upper bound 5×10^5, load 0.88.

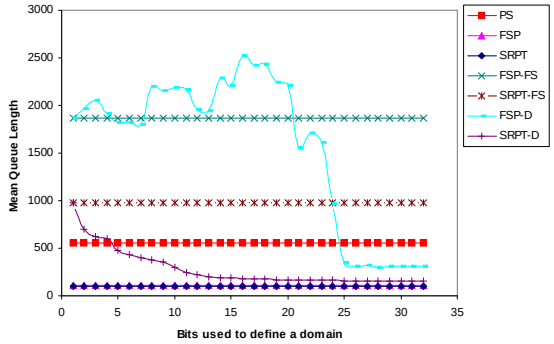


Figure 24: Mean queue length versus k for web trace, domain-based scheduling, G/G/1/m; Pareto arrivals with $\alpha = 1.32$, Lower bound 84, Upper bound 5×10^5, load 0.88.

queue length of all the scheduling policies with Pareto arrivals as a function of k. The effects here are very similar to those discussed previously, as are the results for Weibull arrivals.

Our performance evaluation of SPRT-D and FSP-D demonstrates that better, practical estimators of service time are possible and that they can dramatically improve the performance of size-based scheduling policies on web servers.

5. CONCLUSIONS AND FUTURE WORK

This paper has made the following contributions:

- We have demonstrated that the assumption that file size is a good indicator of service time for web servers is unwarranted. File size and service time are only weakly correlated. The implication is that size-based scheduling policies such as SRPT and FSP are likely to perform worse than expected.
- We have evaluated the performance of SRPT-FS and FSP-FS, SRPT and FSP using the assumption, with varying correlation between file size and service time. We found that their performance does indeed vary dramati-

cally with correlation. In some cases SRPT-FS and FSP-FS can actually perform worse than PS.

- We have proposed, implemented, and evaluated a better service time estimator that makes use of the hierarchical nature of routing on the Internet and the history of past requests available on the web server. We refer to SRPT and FSP augmented with our domain-based estimator as SRPT-D and FSP-D. The state size of our estimator is a parameter.

- We have found that, with a small state size, SRPT-D can outperform PS. With a practical state size, SRPT-D can exhibit close to ideal performance. FSP-D requires a significantly larger state size to perform close to its ideal. SRPT reacts very quickly to increasingly accurate service time estimates.

Fairness is an important concern in the deployment of domain-based scheduling. Slowdown has been used in previous research work [8, 22] as the fairness metric, but slowdown has two possible interpretations. Slowdown can be defined as sojourn time over service time or sojourn time over file size. We have studied fairness using both interpretations and our initial results show that SRPT-D outperforms PS in fairness under most conditions using both. We are working to extend these initial results.

A limitation of this work is that we have focused on web servers that provide static content. We speculate that service time estimators for web servers that provide dynamic content may also be possible. Small improvements in any such estimators would lead to significant improvements in the performance of algorithms such as SRPT. We are exploring this possibility. We are also considering hierarchical domain-based estimators in which a request would match a series of concentric domains defined by the high order $k, k-1, \ldots, 0$ bits of the source address. The request would then use the service time estimate provided by smallest domain which has sufficient samples, or the domain for which past estimates have been most accurate.

6. REFERENCES

[1] The apache software foundation. http://www.apache.org/.

[2] Bonnie, a unix file system benchmark. http://www.textuality.com/bonnie/.

[3] The internet traffic archive. http://ita.ee.lbl.gov/.

[4] The ircache project. http://www.ircache.net/.

[5] The squid web proxy cache project. http://www.squid-cache.org/.

[6] ALLEN, A. O. *Probability, statistics, and queueing theory with computer science applications*. Academic press, Inc., 1990.

[7] BALAKRISHNAN, H., SESHAN, S., STEMM, M., AND KATZ, R. H. Analyzing Stability in Wide-Area Network Performance. In *ACM SIGMETRICS* (June 1997).

[8] BANSAL, N., AND HARCHOL-BALTER, M. Analysis of SRPT scheduling: investigating unfairness. In *SIGMETRICS/Performance* (2001), pp. 279–290.

[9] BARFORD, P., AND CROVELLA, M. Measuring web performance in the wide area. *Performance Evaluation Review 27*, 2 (1999), 37–48.

[10] BARFORD, P., AND CROVELLA, M. Generating representative web workloads for network and server performance evaluation. In *SIGMETRICS* (98).

[11] BOXMA, O., AND COHEN, J. Heavy-traffic analysis for the G/G/1 queue with heavy-tailed distributions. *Queueing Systems 33* (1999), 177–204.

[12] BUX, W. Analysis of a local-area bus system with controlled access. *IEEE Transactions on Computers 32*, 8 (1983), 760–763.

[13] CARIO, M. C., AND NELSON, B. L. Numerical Methods for Fitting and Simulating Autoregressive-to-Anything Processes. *INFORMS Journal on Computing 10*, 1 (1998), 72–81.

[14] CHINOY, B. Dynamics of internet routing information. In *SIGCOMM* (1993), pp. 45–52.

[15] CROVELLA, M., AND BESTAVROS, A. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. In *SIGMETRICS'96* (Philadelphia, Pennsylvania, May 1996). Also, in Performance evaluation review, May 1996, 24(1):160-169.

[16] DENG, S. Empirical model of WWW document arivals at access links. In *IEEE International Conference on Communication* (June 1996).

[17] DINDA, P., AND O'HALLARON, D. An evaluation of linear models for host load prediction. In *8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8)* (1999).

[18] FRIEDMAN, E. J., AND HENDERSON, S. G. fairness and efficiency in web server protocol. In *SIGMETRICS/Performance* (2003).

[19] FULLER, V., LI, T., YU, J., AND VARADHAN, K. (rfc1519) Classless Inter-Domain Routing (CIDR): an address assignment and aggregation strategy, September 1993. http://www.faqs.org/rfcs/rfc1519.txt.

[20] HARCHOL-BALTER, M., BANSAL, N., AND SCHROEDER, B. Implementation of srpt scheduling in web servers. Tech. Rep. CMU-CS-00-170, Carnegie Mellon School of Computer Science, October 2000.

[21] HARCHOL-BALTER, M., CROVELLA, M. E., AND PARK, S. The case for srpt scheduling in web servers. Tech. Rep. MIT-LCR-TR-767, MIT lab for computer science, October 1998.

[22] HARCHOL-BALTER, M., SCHRDER, B., BANSAL, N., AND AGRAWAL, M. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems (TOCS) 21*, 2 (May 2003).

[23] HINDEN, R. (rfc1517) Applicability statement for the implementation of Classes Inter-Domain Routing (CIDR), September 1993. http://www.faqs.org/rfcs/rfc1517.txt.

[24] KRISHNAMURTHY, B., AND REXFORD, J. *Web Protocols and Practice: HTTP1.1, Networking Protocols, Caching, and Traffic Measurements*. Addison-Wesley, 2001.

[25] MANLEY, S., AND SELTZER, M. Web Facts and Fantasy. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS97)* (Monterey, CA, 1997).

[26] MILLER, N., AND STEENKISTE, P. Network status information for network-aware applications. In *Proceedings of IEEE Infocom 2000* (March 2000). To Appear.

[27] MUTKA, M. W., AND LIVNY, M. The available capacity of a privately owned workstation environment. *Performance Evaluation 12*, 4 (July 1991), 269–284.

[28] MYERS, A., DINDA, P. A., AND ZHANG, H. Performance characteristics of mirror servers on the internet. In *INFOCOM (1)* (1999), pp. 304–312.

[29] N., T. A., AND D.D, K. On the approximation of the mean response times of priority classes in a stable G/G/C/PR queue. *Journal of the Operational Research Society 43* (1992), 227–239.

[30] PAXSON, V. End-to-end routing behavior in the Internet. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (New York, August 1996), vol. 26,4 of *ACM SIGCOMM Computer Communication Review*, ACM Press, pp. 25–38.

[31] PAXSON, V. End-to-end routing behavior in the Internet. *IEEE/ACM Transactions on Networking 5*, 5 (1997), 601–615.

[32] PAXSON, V., AND FLOYD, S. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking 3*, 3 (1995), 226–244.

[33] PERERA, R. The variance of delay time in queueing system M/G/1 with optimal strategy SRPT. *Archiv fur Elektronik und Uebertragungstechnik 47*, 2 (1993), 110–114.

[34] REKHTER, Y., AND LI, T. (rfc1518) An architecture for IP address allocation with CIDR, September 1993. http://www.faqs.org/rfcs/rfc1518.txt.

[35] SCHRAGE, L. E. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research 16* (1968), 678–690.

[36] SCHRAGE, L. E., AND MILLER, L. W. The queue M/G/1 with the shortest remaining processing time discipline. *Operations Research 14* (1966), 670–684.

[37] SCHREIBER, F. Properties and applications of the optimal queueing strategy srpt - a survey. *Archiv fur Elektronik und Uebertragungstechnik 47* (1993), 372–378.

[38] SCHROEDER, B., AND HARCHOL-BALTER, M. Web servers under overload: How scheduling can help, June 2002.

[39] SESHAN, S., STEMM, M., AND KATZ, R. H. SPAND: Shared passive network performance discovery. In *USENIX Symposium on Internet Technologies and Systems* (1997).

[40] SILBERSCHATZ, A., AND GALVIN, P. *Operating System Concepts, 5th Edition*. John Wiley Sons, 1998.

[41] WESSELS, D., AND CLAFFY, K. (rfc2186) Internet cache protocol (ICP), version 2, September 1997. http://www.faqs.org/rfcs/rfc2186.html.

[42] WESSELS, D., AND CLAFFY, K. ICP and the Squid Web cache. *IEEE Journal on Selected Areas in Communication 16*, 3 (1998), 345–357.

[43] WOLSKI, R. Dynamically forecasting network performance using the network weather service. *Cluster Computing 1*, 1 (1998), 119–132.

[44] XIA, C. H., AND LIU, Z. Queueing systems with long-range dependent input process and subexponential service times. In *Sigmetrics* (2003), pp. 25–36.

[45] ZHANG, Y., DU, N., E PAXSON, AND SHENKER, S. On the constancy of internet path properties. In *ACM SIGCOMM Internet Measurement Workshop* (2001).