Generating Representative Web Workloads for Network and Server Performance Evaluation

Paul Barford and Mark Crovella
Computer Science Department
Boston University
111 Cummington St, Boston, MA 02215

{barford,crovella}@cs.bu.edu

BU-CS-97-006

REVISED

December 31, 1997

Abstract

One role for workload generation is as a means for understanding how servers and networks respond to variation in load. This enables management and capacity planning based on current and projected usage. This paper applies a number of observations of Web server usage to create a realistic Web workload generation tool which mimics a set of real users accessing a server. The tool, called SURGE (Scalable URL Reference Generator) generates references matching empirical measurements of 1) server file size distribution; 2) request size distribution; 3) relative file popularity; 4) embedded file references; 5) temporal locality of reference; and 6) idle periods of individual users. This paper reviews the essential elements required in the generation of a representative Web workload. It also addresses the technical challenges to satisfying this large set of simultaneous constraints on the properties of the reference stream, the solutions we adopted, and their associated accuracy. Finally, we present evidence that SURGE exercises servers in a manner significantly different from other Web server benchmarks.

[1]

1 Introduction

With the growing importance of the Web as an Internet application comes an increasing need to model and reproduce typical Web workloads accurately. In particular, the ability to generate a stream of HTTP requests that mimics a population of real users is important for performance evaluation and capacity planning of servers, proxies, and networks.

Generating representative Web reference traces is difficult because Web workloads have a number of unusual features. First, empirical studies of operating Web servers have shown that they experience highly variable demands, which is exhibited as variability in CPU loads and number of

[1]Supported in part by NSF Grants CCR-9501822 and CCR-9706685 and by Hewlett-Packard Laboratories.

open connections (*e.g.,* see [14]). This indicates that it is important to pay attention to those properties of Web reference streams that have been shown to exhibit high variability, such as request interarrivals [8, 10] and file sizes [5, 2].

The second unusual feature of Web workloads is that network traffic due to Web requests can be *self-similar—i.e,* traffic can show significant variability over a wide range of scales. Self-similarity in traffic has been shown to have a significant negative impact on network performance [9, 16], so it is an important feature to capture in a synthethic workload.

To capture these properties in a workload generator, one of two approaches could be used: a trace-based approach or an analytic approach. Trace-based workload generation uses prerecorded records of past workloads and either samples or replays traces to generate workloads. In contrast, analytic workload generation starts with mathematical models for various workload characteristics and then generates outputs that adhere to the models.

These two approaches each have strengths and weaknesses. The trace-based approach has the advantage that it is easy to implement, and mimics activity of a known system. However, it treats the workload as a "black box." As a result, insight into the causes of system behavior is hard to obtain. Furthermore, it can be hard to adjust the workload to imitate future conditions or varying demands. Analytic workloads do not have these drawbacks, but they can be challenging to construct for at least three reasons. First, it is necessary to identify those characteristics of the workloads which are important to model. Second, the chosen characteristics must be empirically measured. Third, it can be difficult to create a single output workload that accurately exhibits a large number of different characteristics.

In this paper we describe methods we have developed for generating analytically-based Web reference streams. Our goal is to imitate closely a stream of HTTP requests originating from a fixed population of Web users. These methods are embodied in a tool called SURGE (Scalable URL Reference Generator). The methods used in SURGE have the advantages that come with the analytic approach. In particular, the models used in SURGE are explicit and so can be examined directly by the user. In addition, SURGE's models can be varied to explore expected future demands or other alternative conditions.

To construct an analytic Web workload, it is necessary to address the three challenges described above. In this paper we show how each challenge was addressed. First, we describe the set of characteristics of Web reference streams we chose to model, and explain why we believe that this set is important. Second, we describe the results of new measurements of the Web that were needed to populate the SURGE models. Third, we describe the issues involved in incorporating these models into a single output reference stream, and how we resolved those issues.

The final result, SURGE, shows a number of properties that are quite different from a common Web workload generator: SPECweb96 [4]. We configured SURGE and SPECweb96 to exercise a server at equal rates measured in terms of bytes transferred per second. Under these conditions, SURGE places much higher demands on the server's CPU, and the typical number of open connections on the server is much higher.

In addition we show that in an another important respect, the workload generated by SURGE is quite different from that of typical Web workload generators. In particular, the traffic generated by SURGE shows self-similar properties; we show that this is not generally true for a typical alternative Web workload generator (SPECweb96) when run at high loads. Because SURGE's analytic approach makes its models explicit, we can examine the causes of self-similarity in SURGE traffic. Based on this insight we conclude that most Web workload generators proposed to date probably do not generate self-similar traffic at high loads.

This paper is organized as follows. The next section describes the particular characteristics of Web reference streams we chose to model, and why this set is important. Section 3 describes the problems to be overcome in creating a single reference stream adhering to those characteristics; Section 4 describes our solutions to those problems, and the resulting tool. Section 5 then compares the effects of SURGE and SPECweb96 on a Web server, and discusses the reasons for the differences observed. We present our conclusions in Section 6.

2 Web Workload Characteristics

Our goal in developing SURGE is to be able to exercise servers and networks in a manner representative of the Web. In particular, for servers we are interested in the behavior of the network stack and filesystem, and their associated buffering subsystems. In the case of both networks and servers, we are also interested in the effects of high variability in the workload.

These motivations drove the choice of particular Web workload characteristics used in SURGE. These characteristics can be divided into two main categories. The first category is concerned with the idea we call *user equivalents.* The second category is the set of *distributional models.*

User Equivalents. The idea behind user equivalents is that the workload generated by SURGE should roughly correspond to that generated by a population of some known number of users. Thus, the intensity of service demand generated by SURGE can be measured in user equivalents (UEs).

A user equivalent is defined as a single process in an endless loop that alternates between making requests for Web files, and lying idle. Both the Web file requests and the idle times must exhibit the distributional and correlational properties that are characteristic of real Web users. Each UE is therefore an ON/OFF process; we will refer to periods during which files are being transferred as ON times, and idle times as OFF times. UEs have a straightforward implementation as independent threads or processes.

There is not generally a simple relation between UEs and metrics like HTTP requests per second or bytes requested per second. This is because the rate at which an individual UE requests service is dependent on the state of the system; delays in the server or network in transferring data will increase the time between requests. However, for any particular experiment it is easy to observe the *effective* request rate and data transfer rate. We use this approach in Section 5 to make comparisons between workloads on the basis of equivalent system loading.

Basing the workload model on user equivalents has important effects on system performance. Since each UE has significant idle periods, a UE is a very bursty process. Each UE exhibits long periods of activity followed by long periods of inactivity.

This is quite different from approaches typically taken in other Web workload generators. Most other workload generators send requests without preserving consistent properties of OFF times [4, 15, 19]. The general approach used in these workload generators is simply to make requests for files from a server as quickly as possible. We will show in Section 5 that ignoring OFF times destroys self-similarity in the resulting traffic at high loads.

Specifying UE behavior during the ON period requires understanding some of the details of Web organization. In particular, Web files may include by reference other files; these files are required to display the result properly. (Typically these included files might supply images or graphics.) Thus the user's request for a single Web file often results in the transfer of multiple files from the Web server. We call a Web file along with all the files that also must be transferred to display it a *Web object.*

The particular details of how components of a Web object are transferred depends on the browser used and the version of the HTTP protocol employed. HTTP 0.9/1.0 uses a separate TCP connection for each file, while HTTP 1.1 allows a single TCP connection to be used for multiple

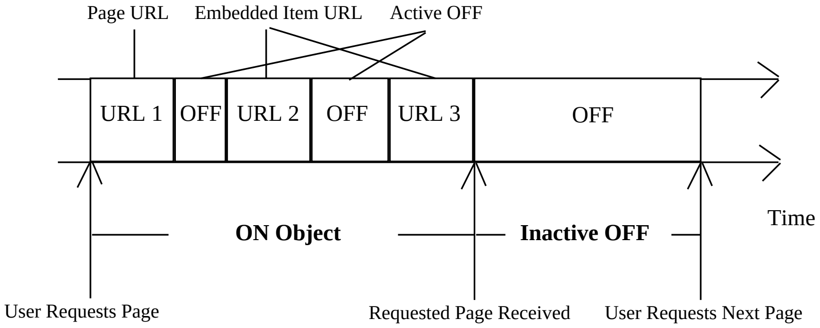

Figure 1: ON/OFF Model Used in SURGE (HTTP 1.0; one TCP connection at a time).

files. In addition, some HTTP 0.9/1.0 browsers open multiple simultaneous TCP connections when transferring components of a Web object. For the experiments reported here, SURGE was configured to use HTTP 0.9 and multiple TCP connections were not used; however modifying SURGE to use other styles is straightforward.

This style of transferring Web objects means that there are two kinds of OFF time, as shown in Figure 1. *Inactive OFF* time corresponds to the time between transfers of Web objects; this is user "think time." *Active OFF* time corresponds to the time between transfer of components of a single Web object; it corresponds to the processing time spent by the browser parsing Web files and preparing to start new TCP connections.

Distributional Models. Having described the UE basis for the design of SURGE, we describe the set of probability distributions needed by each UE.

An important feature of some of the distributions we will consider is that they exhibit *heavy tails.* We say here that a heavy-tailed distribution is one whose upper tail declines like a power law, *i.e.,*

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

where $a(x) \sim b(x)$ means $\lim_{x\to\infty} a(x)/b(x) = c$ for some constant c. Random variables whose distributions are heavy tailed exhibit very high variability; in fact, their variance is infinite, and if $\alpha \leq 1$, their mean is also infinite.

To fulfill SURGE's goals, its output must adhere to the following six statistical properties of Web reference streams.

1) File Sizes. In order to exercise the server's filesystem properly, it is important that the collection of files stored on the server agree in distribution with empirical measurements. As noted in [5, 2], these distributions can be heavy-tailed, meaning that the server's filesystem must deal with highly variable file sizes.

2) Request Sizes. In order to exercise the network properly, the set of files transferred should match empirical size measurements. We call this set the *requests.* The distribution of requests can be quite different from the distribution of files because requests can result in any individual file being transferred multiple times, or not at all. Again, empirical measurements suggest that the set of request sizes can show heavy tails [5].

Note that the difference between the sets we call file sizes and request sizes corresponds to differences in distribution between what is stored in the filesystem of the server and what is transferred over the network from the server.

3) Popularity. A third property of the workload, related to the previous two, is the relative number of requests made to individual files on the server. Popularity measures the distribution of requests on a per-file basis. Note that even when the previous two properties are fixed (distributions of file sizes and request sizes) there still can be considerable flexibility in how requests are distributed among individual files. The distribution of popularity has a strong effect on the behavior of caches (*e.g.,* buffer caches in the file system), since popular files will typically tend to remain in caches.

Popularity distribution for files on Web servers has been shown to commonly follow Zipf's Law [21, 1, 6]. Zipf's Law states that if files are ordered from most popular to least popular, then the number of references to a file (P) tends to be inversely proportional to its rank (r). That is:

$$P = kr^{-1}$$

for some positive constant k. This property is surprisingly ubiquitous and empirical measurements of the exponent are often quite close to -1, although the reasons behind this effect in Web workloads are unclear. This distribution of references among files means that some files are extremely popular, while most files receive relatively few references.

4) Embedded References. In order to capture the structure of Web objects, it is important to characterize the number of embedded references in a Web object. This is done by characterizing the distribution of the number of embedded references in Web files. In the configuration of SURGE used in this paper, the number of embedded references is important because OFF times between embedded references (Active OFF times) are typically short, while OFF times between Web objects themselves (Inactive OFF times) are typically much longer. Previous work has not looked closely at the question of how many references are typically contained in a Web object.

5) Temporal Locality. Temporal locality in Web workloads refers to the likelihood that, once a file has been requested, it will be requested again in the near future. Accurate characterization of temporal locality is important because caching effectiveness can be significantly increased when temporal locality is present.

One way to measure temporal locality is using the notion of *stack distance.* Given a sequence of requests, its corresponding stack distance sequence can be generated as follows. Assume that the files are stored in a push-down stack. Considering each request in order, move the requested file to the top of the stack, pushing other files down. The depth in the stack at which each requested file is found is the request's stack distance [1, 13].

Stack distance serves to capture temporal locality because small stack distances correspond to requests for the same file that occur close to each other in the reference stream. Note that, assuming the initial contents of the stack are known, the stack distance sequence contains information equivalent to the request sequence; each can be obtained from the other.

Thus the distribution of stack distances for a Web request sequence serves as a measure of the temporal locality present in the sequence. Typical distributions for request sequences arriving at Web servers have been studied in [1]; results show that these are well modeled using lognormal distributions. Since the lognormal distribution has most of its mass concentrated at small values, this is indicative that significant temporal locality is often present in Web request sequences.

6) OFF Times. As described in the previous section, accurate modeling of inactive OFF times is necessary to capture the bursty nature of an individual Web user's requests. Proper characterization of active OFF times is necessary to replicate the transfer of Web objects. Previous work has measured OFF times [5] and the related question of interarrival times of HTTP requests at a server [8].

In summary, the notion of user equivalents along with the six distributions discussed in this section constitute the definition of the SURGE model for Web workload generation. These characteristics represent a large set of contraints on the properties of the workload generated by SURGE. In the next two sections we describe the difficulties presented in trying to satisfy all these constraints simultaneously, and how those difficulties are addressed in SURGE.

3 Obstacles to Creating Representative Web Workloads

Two basic problems arise in building a workload generator to meet the requirements described in the previous section. First, models for each of the six distributions are required, and second, methods for combining those distributions into a single output stream are required.

3.1 Distributional Model Obstacles

To populate our set of distributional models fully, three new distributions were needed.

File Sizes on the server were identified as a key characteristic of Web workloads in Section 2. File sizes on servers were studied in [5] but that work concentrated only on the tail of the distribution. For SURGE we need a model for the distribution that is accurate for the body as well as the tail.

Active OFF Times have not been specifically addressed in other studies. Client OFF times in general are described in [5]. However, no attempt to describe Active OFF time as a separate distribution was made; thus a model for Active OFF time was required.

Finally, a model for the number of **Embedded References** in a document was required. This number is difficult to extract from client trace data since there is typically no record in the data that indicates which documents are embedded. This characteristic can, however, be inferred using Active OFF times (as is described in Section 4).

The task of developing distributional models for each of these Web characteristics required the analysis of empirically measured Web workload traces. The most common way of specifying a statistical model for a set of data the is through the use of *visual* methods such as quantile-quantile or cumulative distribution function (CDF) plots. These methods, however, do not distinguish between two closely fitting distributions nor do they provide any level of confidence in the fit of the model. To address this drawback one can use goodness-of-fit tests [7, 17]. However, these tests also present a number of problems. Methods which place data into bins (such as Chi-Squared tests) suffer from inaccuracies due to the choice of bin size. Methods based on empirical distribution functions (such as the Anderson-Darling test) are more likely to fail when applied to large datasets. Since most empirical Web measurements constitute large datasets, the methods we used to select distributional models needed to address these shortcomings of goodness-of-fit tests.

3.2 Obstacles in Combining Distributions

Generating a single output workload which exhibits each of the six characteristics which make up the SURGE model is difficult. We address this problem by developing methods for matching requests to files and for generating a representative output sequence.

3.2.1 The Matching Problem

The matching problem starts from three Web workload characteristics: file size distribution, request size distribution, and popularity. Given these three distributions, determining the total number of requests for each file on the server is the matching problem. The matching problem arises because even when these three distributions are fixed, there still can be considerable flexibility in how requests are actually distributed among individual files.

First, any server will contain a set of uniquely named files each with a specific size. Let X be the set of file sizes on the server, where x_i is the size of file i. Second, using Zipf's law we can calculate the set Y which consists of the number of references to each of the files on the server (the popularity distribution). However, Zipf's Law does not determine *which* files correspond to each element of Y. Finally, request size distribution is also described by an empirical distribution, $F(x)$.

Given these as inputs, the matching problem is to find a permutation of the set Y such that the following conditions are satisfied:

1. There is a one-to-one mapping between elements in the sets X and Y. The permutation of Y will be described by the set of indices Z which is a permutation of integers $1, ..., n$. Thus, each y_{z_i} is the number of requests for file i.

2. The mapping between X and Y results in a set of file requests. This set can be described by a CDF $G(x)$ that is defined as follows:

$$G(x_j) = 1 - (\sum_{i=1}^{j} y_{z_i} / \sum_{i=1}^{n} y_{z_i})$$

   Thus, we can determine a desired value for each y_j which we will call $\hat{y}_j$ by equating $G(x)$ and $F(x)$ and solving for $\hat{y}_i$ so:

$$\hat{y}_j = F(x) \sum_{i=1}^{n} y_{z_i} - \sum_{i=1}^{j-1} y_{z_i}$$

3. We create a permutation Z such that $\hat{y}_i$ is as close as possible to y_{z_i} for all i.

3.2.2 Temporal Locality

The result of the matching process is that each unique file will be matched with a total request value. However, this is still not sufficient for the generation of a request stream which exhibits temporal locality. Temporal locality can be analyzed using the distribution of stack distances. This distribution is an indication of temporal locality because stack distance measures the number of intervening references between references to the same document [1]. To obey temporal locality, the sequence of document requests must be arranged such that when cast in a push-down stack the resulting distribution of stack distances will match empirically measured distributions.

In addition to the distribution of stack distances, a method for request sequence generation must distribute references to each file as evenly as possible throughout the entire sequence.

4 Solutions Adopted in Surge

4.1 New Distributional Models

The BU client trace data sets discussed in [6] were used to develop the three models required to complete Surge. These traces record the behavior of users accessing the Web over a period of two months. In general, it is desirable that a large number of datasets from different environments be examined to determine representative distributions. However, our focus in the work to date was in defining the necessary set of workload characteristics, and so we did not include a wide range of datasets in our analysis. We expect to examine a larger number of datasets in future work, and for that reason Surge's algorithms and general structure were developed to allow easy adaptation to other distributions. Thus, Surge has been developed as a highly parameterizable tool.

To develop these models, we used standard statistical methods, similar to those used in [17]. We used the Anderson-Darling (A^2) empirical distribution function (EDF) test for goodness of fit

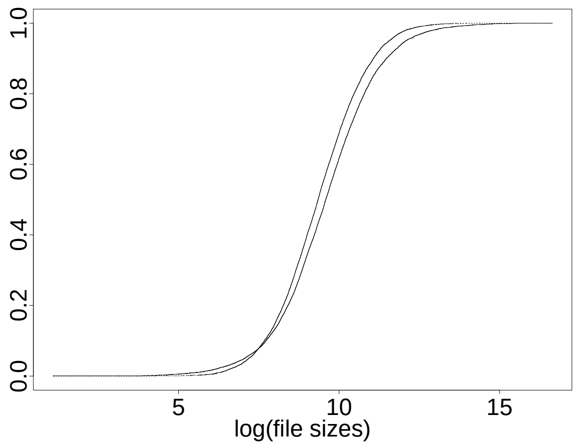

Figure 2: CDF of Log-transformed File Sizes vs. Fitted Normal Distribution

(recommended in [7, 17]) and the λ^2 test (described in [18]) to compare how well analytic models describe an empirical data set.

File Sizes. Completion of the file size model began with the assumption that the heavy tailed characterization of the distribution as described in [5, 2] was accurate. The model was then developed as a hybrid consisting of a new distribution for the body, combined with a Pareto distribution for the upper tail. The λ^2 test on the body of data (11,188 points) versus a number of distributional models (lognormal, Weibull, Pareto, exponential, and log-extreme) showed that the best (smallest) λ^2 value was for the lognormal distribution. The CDF of the log-transformed data versus the normal distribution can be seen in Figure 2.

However, the A^2 showed no significance in terms of goodness of fit. The failure of the A^2 test was due primarily to the fact that a fairly large data set was used for the test; this is a common problem with EDF tests [7, 17]. Therefore a method of testing goodness of fit using random sub-samples (as described in [17, 3]) was used, which indicated a good fit between the EDF of file sizes and the lognormal distribution.

Censoring techniques were employed to determine where to split between the lognormal distribution for the body and the heavy tailed distribution in the tail. A sample is said to be *right censored* if all observations greater than some value are missing. The body of our sample can be assumed to be right censored since we assume that it is contaminated with a heavy tail. We use the A^2 statistic to determine the cutoff point between the body and the tail. By successively increasing the amount of right censoring in the empirical data and then testing goodness of fit, we determine the cutoff between the two distributions to be at approximately 133KB (93% of the files lie below the cutoff). The cutoff value along with the hybrid distributional model allows us to generate the appropriate distribution of file sizes on the server.

Active OFF Times. We consider an OFF time to be "Active" if it is less than a threshold time, which we chose to be 1 second based on inspecting the data. λ^2 tests showed Weibull to be the best fit of those considered. The CDF plot of the set of Active OFF times versus the fitted Weibull distribution is shown in Figure 3. We found no significance at any level for the A^2 test, which we again attribute to the relatively large sample size (40,037 elements). However, for random sub-samples the A^2 test indicated a good fit to the Weibull model.

Embedded References. The number of embedded references in each file was extracted from the traces of file transfers by identifying sequences of files fetched by a given user for which the

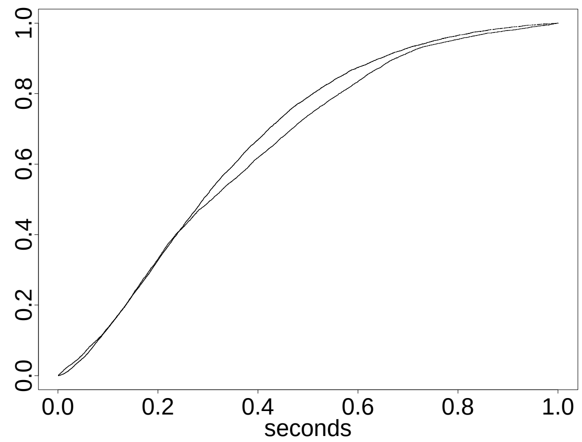

Figure 3: CDF of Active OFF Times vs. Fitted Weibull Distribution

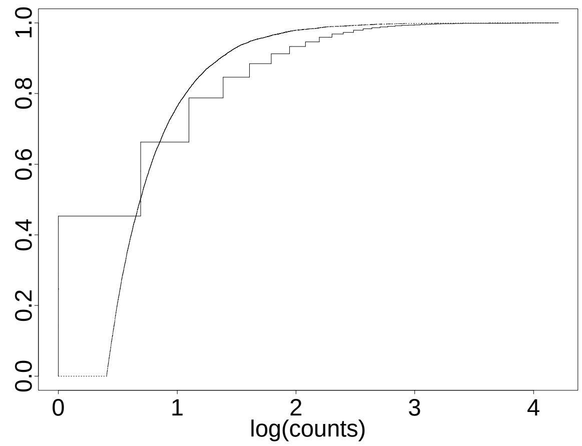

Figure 4: CDF of Number of Embedded References vs. Fitted Pareto Distribution

OFF time between transfers was always less than the one second threshold (resulting in 26,142 data points). Initial inspection of this data set showed that its distribution had a long right tail. Generation of distributional plots suggested the Pareto distribution as the best visual fit for the data. A least squares estimate of the tail slope in a log-log complementary distribution plot resulted in an estimate of α which gave a good visual fit as can be seen in Figure 4.

A^2 tests once again failed to find significant goodness of fit. Even the random sub-sample method did not indicate a good fit which we attribute to the fact that there were only a few values in the tail of our empirical data and thus very few in the tails of the sub-samples.

A summary of the model distributions and parameters used in SURGE are given in Table 1.

4.2 Solutions to the Matching Problem

The solution to the matching problem as described in Section 3.2.1 results in a mapping of requests to files such that the desired file size distribution, request size distribution, and popularity characteristics are all present. The success of a matching is expressed as the difference between $\hat{y}_i$, the

Component	Model	Probability Density Function	Parameters
File Sizes – Body	Lognormal	$p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(lnx-\mu)^2/2\sigma^2}$	$\mu = 9.357$; $\sigma = 1.318$
File Sizes – Tail	Pareto	$p(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$	$k = 133K$; $\alpha = 1.1$
Popularity	Zipf		
Temporal Locality	Lognormal	$p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(lnx-\mu)^2/2\sigma^2}$	$\mu = 1.5$; $\sigma = 0.80$
Request Sizes	Pareto	$p(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$	$k = 1000$; $\alpha = 1.0$
Active OFF Times	Weibull	$p(x) = \frac{bx^{b-1}}{a^b} e^{-(x/a)^b}$	$a = 1.46$; $b = 0.382$
Inactive OFF Times	Pareto	$p(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$	$k = 1$; $\alpha = 1.5$
Embedded References	Pareto	$p(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$	$k = 1$; $\alpha = 2.43$

Table 1: Summary Statistics for Models used in SURGE

desired number of references to file of size x_i, and y_{z_i}, the actual number of references to file of size x_i. There are a number of ways in which these differences can be combined to form an optimization problem. First, we could bound the maximum difference between the two distributions: $\max |\hat{y}_i - y_{z_i}|$. Second, we could bound the total error between the two distributions: $\sum_{i=1}^{n} |\hat{y}_i - y_{z_i}|$. Finally, we could bound the error in some important portion of the distribution: $|\hat{y}_i - y_{z_i}|$ given $z_1 ... z_{i-1}$ are already determined.

A single algorithm is optimal for the first two cases. This algorithm creates the permutation index Z such that the largest value in $\hat{y}$ is matched to the largest value in y, the next largest in $\hat{y}$ in is matched with the next largest in y and so on. The proof of optimality of this method for the first two cases above is straightforward.

However, the optimal method can allocate error between the two distributions in an undesirable way (*e.g.,* it can concentrate the error in the tail of the distribution). Since large files cause the greatest impact on network and server performance, the ability to match most closely in the tail is important. Therefore another method of matching was developed which allows matching in either the tail or head of the distribution to be optimal. This method matches values in the set Y to those generated from $F(x)$ beginning with either the smallest value in X or the largest. This method results in very close fits in the tail of the distribution, and does not introduce large errors in the body.

4.3 Solution to the Sequence Generation Problem

To generate reference sequences with the proper temporal locality, SURGE begins by placing each of the individual file names in a stack (the initial ordering is not important). A sequence of values drawn from the proper lognormal distribution (Table 1) is then generated. This sequence is inverted to obtain a sequence of names. This is done by repeatedly selected files from the stack at a distance equal to the next value in the sequence, reordering the stack after each selection.

Unfortunately this simple method, if followed exactly, results in a nonuniform distribution of file names throughout the request sequence. Therefore the method was modified by defining a small window on either side of the location specified in the lognormal sequence. Each of the files in this window are then assigned a weight whose value is proportional to the remaining number of requests required for that document. The choice of which file to move to the top of the stack is then based probabilistically on the weights of the files within the window.

We found that this method of file sequence generation gives a very good distribution of file names throughout the sequence, and the resulting stack distance values still follow the desired lognormal distribution.

Finally, the use of multiple client hosts presents a problem for temporal locality. When UE threads are running on the same host they can share a common file name sequence, each thread using the next file name in the sequence as needed. However, when UE threads are running on multiple hosts, they cannot share a common list without significant synchronization overhead. To handle this case we generate independent file name sequences with related stack distance properties. This is done by scaling the values output from the lognormal distribution by the number of clients which will be used in the simulation. In the case in which requests from separate hosts interleave at the server in a regular fashion this will result in the proper temporal locality properties. Our results show that this simple scaling approach is sufficient to maintain approximately the same stack distance distribution as in the single client case.

5 Performance Characteristics of SURGE Workloads

In this section we describe the current implementation of SURGE and the results of initial experiments performed using SURGE. For comparison purposes we also present results using SPECweb96.

5.1 Implementation and Validation of SURGE

SURGE is implemented in two parts: a set of programs written in C that precompute four datasets, and a multithreaded program written in Java that makes Web requests using the four datasets. The precomputed datasets consist of the sequence of requests to be made, the number of embedded files in each Web object to be requested, and the sequences of Active and Inactive OFF times to be inserted between requests. The implementation used Java for portability to a wide range of client machines.

Validation consisted of verifying that SURGE's output agreed with the six distributional models (file sizes, request sizes, popularity, embedded references, temporal locality, and OFF times). Results showed that the efficiency of the Java implementation affected the accuracy of OFF time generation. In particular, an interpreted implementation of Java introduced too much overhead to allow a close match to ideal OFF times; switching to a compiled implementation remedied the problem. In addition, we found that short runs (less than 15 minutes) did not always allow enough samples to be made of some distributions to reach acceptable match with the ideal cases; however runs of 30 minutes were generally adequate to achieve close fits between the ideal distributions and the measured results.

Of all the distributional models, only temporal locality was affected when SURGE was scaled up (run on larger numbers of host machines). As the number of hosts used by SURGE was increased, the resulting stack distance distributions always appeared to be lognormal, but their characteristics changed somewhat. This was expected since an exact match to temporal locality properties could not be guaranteed unless there was strict synchronization between all clients, which would have resulted in an unacceptably low maximum request rate.

5.2 Experimental Setup

The environment for these experiments consisted of six PCs running on a 100 Mbps network that could be isolated from other networks. The PCs were configured with 200MHz Pentium Pro CPUs and 32MB of RAM. SURGE clients (from 1 to 5 hosts) ran under Windows NT 4.0, while the server system (a single host) ran Apache v1.2.4 under Linux 2.0.

	SPECweb96			Surge		
Nominal pkts/sec	HTTP ops/sec	Requests	TCP Packets	UEs	Requests	TCP Packets
70	3	5901	118560	50	5293	131642
300	14	26028	560238	150	26055	507727
500	25	46520	1000289	250	48238	874570

Table 2: Summary of Comparison Experiments.

Since we are primarily interested in the behavior of the Web server and the network, it was important to prevent bottlenecks on the client from limiting overall system performance. We found that when more than approximately 50 UE threads are run on a client host, its resource limitations start to affect performance. As a result, we never run more than 50 UE threads on any host.

On the Web server, we examined CPU utilization and the number of active TCP connections; for the network, we measured the total number of both files and packets transferred over the run. The number of open connections on the server was sampled every 100ms and the CPU utilization was sampled every second. Both values were taken from the Linux `/proc` filesystem. Measurements of traffic on the network were taken using `tcpdump`.

In each of the experiments discussed in this section, the number of UEs is kept constant throughout the experiment. Thus we only explore the stationary behavior of the workload at this time. This allows a direct comparison with other workload generators, which also assume stationary conditions. However, there is no barrier to constructing a workload with varying numbers of UEs over time, such as would correspond to the case in which users visit the Web site for some period of time and then move on.

This assumption of stationarity in the workload for real servers probably only applies over short timespans during which the number of users at the Web server remains approximately constant. For this reason we restrict ourselves to runs no greater than 30 minutes. Note that even over intervals as short as these, Web servers can show high variability in many system metrics [14].

The workload generated by SPECweb96 depends on two user-defined parameters: a target number of HTTP operations per second, and the number of threads used to make requests. The general approach used by SPECweb96 is for each thread to generate HTTP requests at a constant rate; the request rate for each thread is determined by dividing the target rate by the number of threads. In all our experiments we used 16 threads, which was sufficient since the achieved operations per second were always close to the requested values.

Thus the intensity of the SPECweb96 workload is specified in terms of expected number of HTTP operations per second, while (as discussed in Section 2) the intensity of the Surge workload is expressed in terms of user equivalents. To compare the effects of Surge and SPECweb96, we empirically determined configurations of each resulting in approximately equal amounts of data transferred during a 30 minute run. We identified three such configuration levels of each workload generator; for each level, the difference between the two in terms of data transferred is less than about 20%. These configuration levels are used only for general comparisons between the two workloads. Table 2 summarizes the Surge and SPECweb96 workloads used, showing the number of requests satisfied and the number of TCP packets transferred in each 30 minute run.

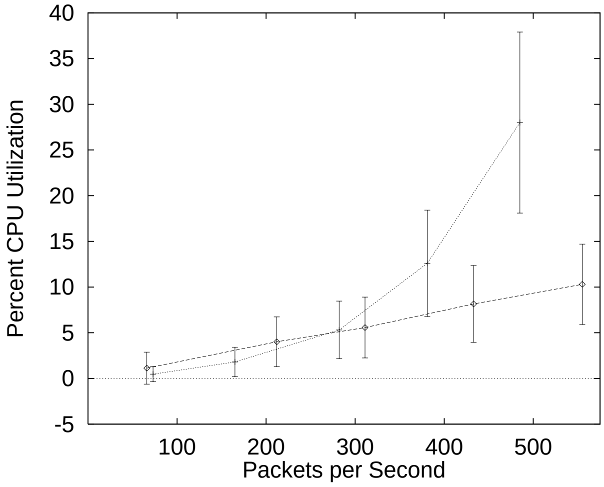

Figure 5: CPU Utilization for Surge (upper at right) and SPECweb96 (lower at right) Workloads

5.3 Results

We first consider the differences between Surge and SPECweb96 in terms of their impact on the server; then we examine their different impacts on the network.

Server Impacts. The most immediately noticeable difference between the effects of the Surge and SPECweb96 workloads is in server CPU utilization. Figure 5 shows plots of mean CPU utilization for the two workloads as a function of average packets transferred per second. Errorbars show one sample standard deviation above and below the mean. Mean and standard deviation were measured over the steady state of each experiment, with the startup transient removed.

Figure 5 shows that when the average transfer rate approaches 500 packets per second, the Surge workload causes a sharp increase in the server's CPU load, while the server's CPU is not very heavily loaded under the SPECweb96 workload.

The difference in CPU load over time is shown in Figure 6. This figure plots instantaneous CPU load over the entire course of each of three experiments for both workloads. This figure shows how the two workloads are quite different in practice. The figure shows that Surge results in a highly varying CPU load with utilization as high as 76% for transfer rates of about 500 pps. In contrast, the CPU load is relatively stable and never goes above 37% for the SPECweb96 workload.

One reason for the difference in CPU demands of the two workloads can be seen by examining the number of active TCP connections in each case. Table 3 shows the mean and standard deviation of the number of connections open on the server measured at 100ms intervals. This table shows that a large number of connections are typically open when running the Surge workload; in contrast, the number of open connections for the SPECweb96 workload is typically very small. In addition, the variability in number of connections is much greater for the Surge workload than for the SPECweb96 workload.

Figure 7 shows the number of open connections at 100ms intervals for the three experiments. Note that figures in the two rows use different scales on the y axis. This figure shows the large fluctuations in the number of open connections for the Surge workload.

Maintaining and servicing a large number of open connections is computationally expensive, as pointed out in [14]. Clearly the difference in the number of active connections is likely to be a contributing factor in the CPU utilization differences seen between the two workloads (Figure 5).

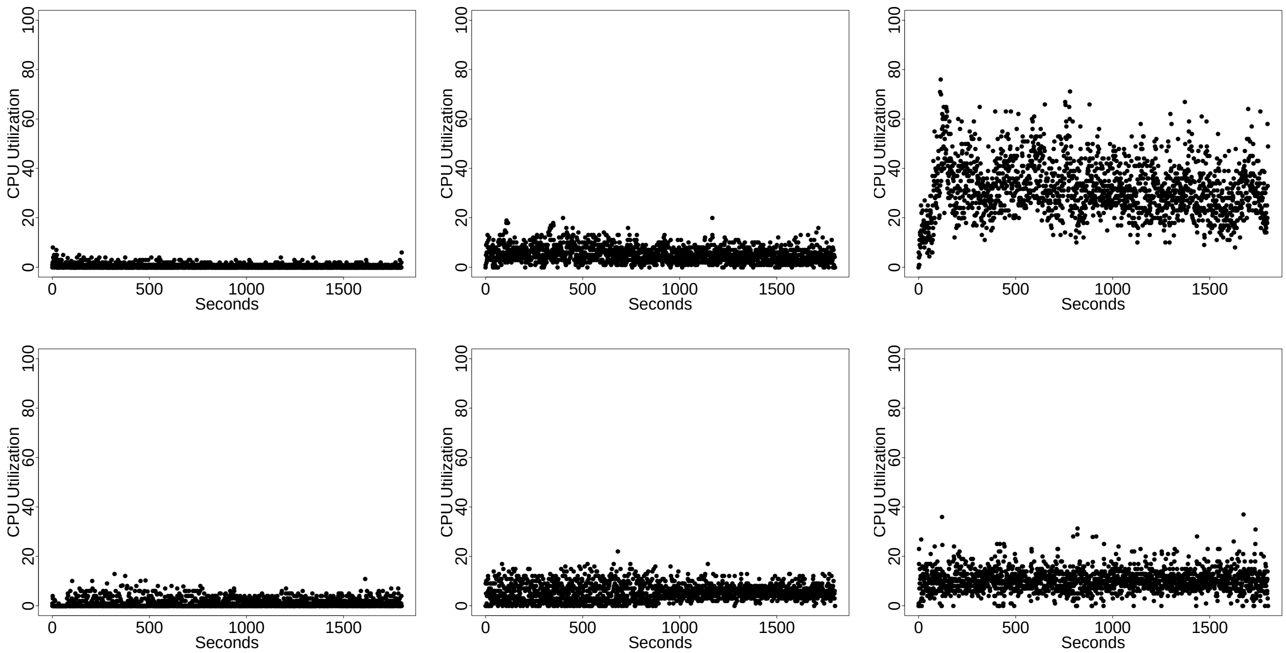

Figure 6: CPU Utilization for 70 pps (left), 300 pps (middle), and 500 pps (right) (upper: SURGE; lower: SPECweb96).

	SPECweb96		SURGE	
Nominal pps	Mean	Standard Deviation	Mean	Standard Deviation
70	0.028	0.18	13.9	3.92
300	0.37	0.69	33.2	12.1
500	0.71	1.41	67.1	35.3

Table 3: Active TCP Connections

The reason that SURGE maintains a much greater number of open server connections on average can be understood by comparing the request generation process in SURGE with that used in SPECweb96. The inter-request time in SPECweb96 is determined by the goal of generating a fixed number of requests per time interval. Thus for SPECweb96, as the number of requests per second increases, each thread's idle time between requests decreases. In contrast for SURGE, as the number of requests per second increases, the idle times of individual threads do not decrease; instead more threads are used.

The result is that in SPECweb96, connections are multiplexed onto a relatively small number of threads, and so the number of simultaneous connections is low. On the other hand, under SURGE the number of threads grows in proportion to the intensity of the workload, which results in a corresponding increase in the number of simultaneous connections that are possible. When connections happen to overlap in large numbers the per-connection rate is slowed, and so in SURGE connections stay open for much longer periods on average than they do under SPECweb96.

In the case of SPECweb96, this difference could be addressed by scaling the number of connections in proportion to the requested operation rate, although it is apparently not customary to

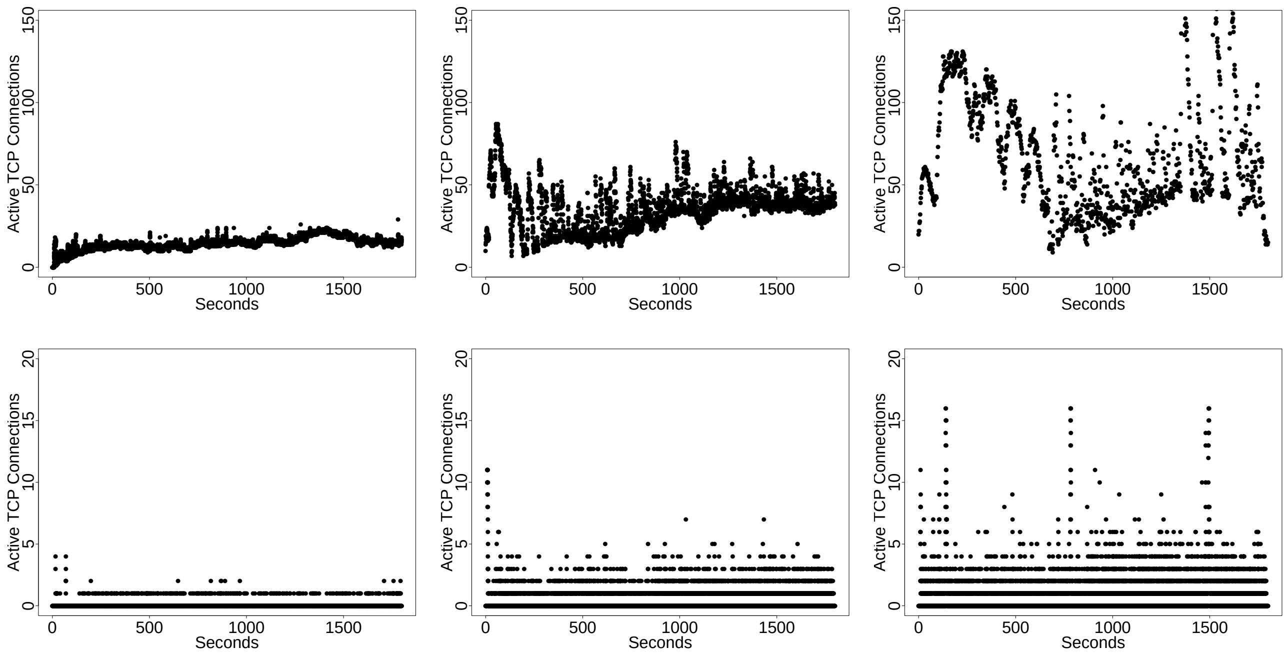

Figure 7: Active TCP Connections for 70 pps (left), 300 pps (middle), and 500 pps (right) (upper: SURGE; lower: SPECweb96).

do so in Web server benchmarking. However, this effect highlights the drawbacks of the approach commonly used in Web workload generators of employing a small or fixed number of threads to make requests.

Network Impacts. Finally we consider differences in network traffic generated by the two workloads. As described in Section 1 we are particularly interested in whether the two workloads generate traffic that is self-similar [12] since this property has been shown to be present in Web traffic [5] and has significant implications for network performance [9, 16].

Self-similarity in the network traffic context refers to the scaling of variability (burstiness). A timeseries $X_t, t = 1, 2, ...$ is said to be *exactly second-order self-similar* if

$$X_t \stackrel{d}{=} m^{-H} \sum_{i=m(t-1)+1}^{mt} X_i \qquad \text{for } 1/2 < H < 1 \text{ and all } m > 0$$

where $\stackrel{d}{=}$ means equality in distribution. This definition suggests a simple test for self-similarity in network traffic, called the variance-time plot. This test plots the variance of $\sum_{i=m(t-1)+1}^{mt} X_i$ against m on log-log axes, where the X_is are measurements of traffic in bytes or packets per unit time. Linear behavior with slope greater than $-1/2$ is suggestive of nontrivial self-similarity.

Variance-time plots for SURGE and SPECweb96 traffic are shown in Figure 8. As the amount of data transferred increases from left to right in the figures, the traffic generated by SURGE shows roughly linear behavior with slope different from $-1/2$. That is, it continues to demonstrate burstiness across all time scales. In contrast, the traffic generated by SPECweb96 shows evidence of self similarity when the traffic intensity is low but as traffic increases, its self-similarity disappears: the variance-time plot approaches a slope of $-1/2$.

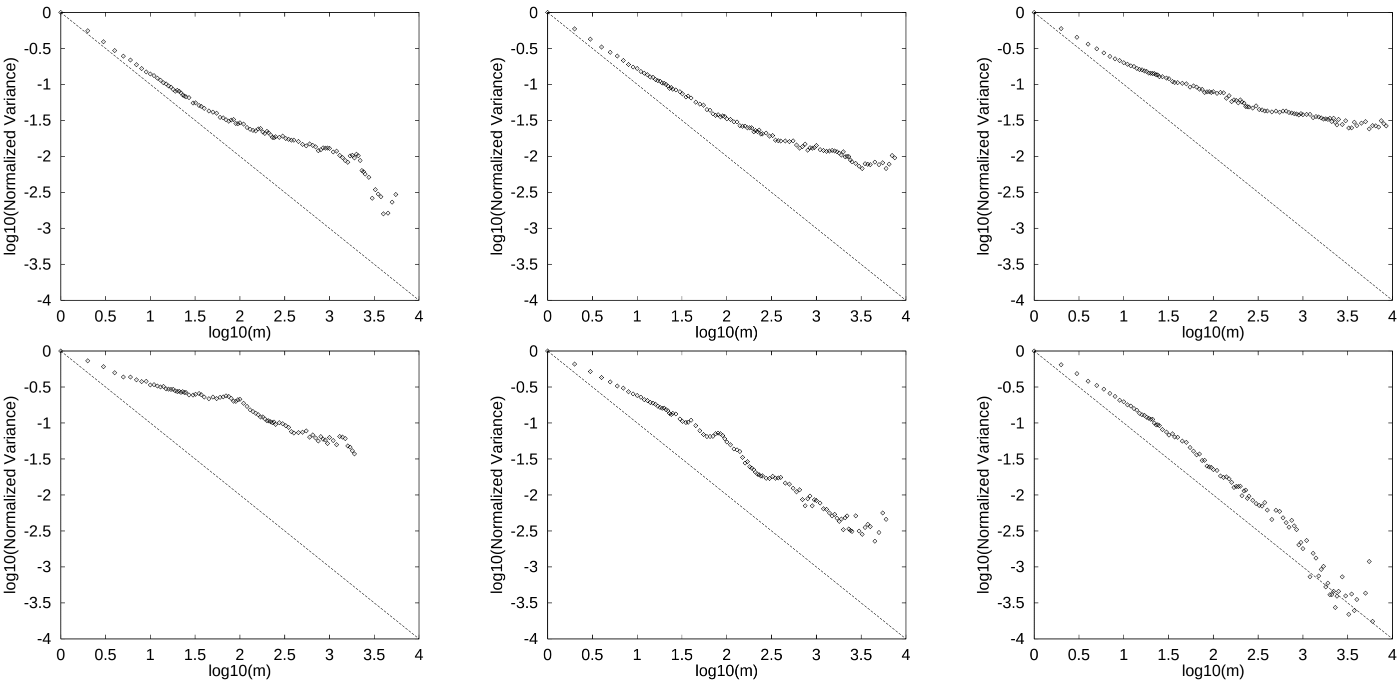

Figure 8: Variance-Time Plots for Surge traffic (upper) and SPECweb96 traffic (lower). Left: 70 pps, Middle: 300 pps, Right: 500 pps.

This effect can be understood by analyzing each of the two cases as a collection of individual sources. Consider a thread as an ON/OFF source; then its marginal distribution is that of a Bernoulli random variable. Its variance is maximized when the two states are equiprobable; when one state starts to dominate the other, the thread's variability in demand declines.

As the workload intensity increases, OFF times of individual threads under SPECweb96 grow shorter, so individual threads exhibit lower variability in demand. Threads increasingly approximate an always-ON source. Thus the traffic generated by SPECweb96 comes from a constant number of sources which each decrease their variability as the workload intensity increases. Hence variability in the resulting aggregate traffic is reduced as well.

In contrast, the traffic generated by Surge comes from constant-variability sources; it is the number of such sources that increases as workload intensity increases. Theoretical work [20] has shown that when transfer durations are heavy-tailed (as they are in Surge) this condition is sufficient for the generation of self-similar traffic. As a result we expect that Surge will generally produce self-similar network traffic under conditions of both high and low workload intensity.

Note that in this respect, the source behavior of SPECweb96 is similar to other Web workload generators whose general goals are to make requests from servers as quickly as possible [15, 19]. Since these workload generators do not preserve significant burstiness in individual thread demands, it is unlikely that under heavy loads they will produce self-similar traffic in practice.

6 Conclusions

In this paper we have described a tool for generating representative Web requests that is based on analytical models of Web use. We have described the characteristics that we feel are important to capture in Web workloads, and why they are important. We have shown that satisfying this large set of requirements presents challenges, but that these challenges can be addressed. The resulting tool, SURGE, incorporates the idea of user equivalents as a measure of workload intensity in addition to six distributional characteristics in order to create representative Web workloads.

Our work draws on the wide range of previous research that has characterized Web usage patterns. In addition we present some new measurements of Web usage that are needed to complete the SURGE model.

We have shown that a workload that satisfies all of these characteristics exercises servers in ways quite different from the most commonly used Web workload generator, SPECweb96. In particular, the workload generated by SURGE maintains a much larger number of open connections than does SPECweb96, which results in a much higher CPU load. In addition, we show that SURGE exercises networks differently than SPECweb96. At high loads, SURGE generates network traffic that is self-similar, which does not appear to be true for SPECweb96. Thus SURGE's workloads are more challenging than SPECweb96's to networks as well. These results suggest that accurate Web workload generation is important, since comparison with realistic workloads indicate that traditional workload generators like SPECweb96 may be optimistic in their assessment of system performance.

Acknowledgements The authors would like to thank Vern Paxson, Ralph D'Agostino, Steve Homer and Randy Pruim for their help in various parts of this work.

References

[1] Virgilio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing reference locality in the WWW. In *Proceedings of 1996 International Conference on Parallel and Distributed Information Systems (PDIS '96)*, pages 92–103, December 1996.

[2] M.F. Arlitt and C.L. Williamson. Web server workload characterization: The search for invariants. In *Proceeding of the ACM SIGMETRICS '96 Conference*, Philadelphia, PA, April 1996.

[3] Henry Braun. A simple method for testing goodness of fit in the presence of nuisance parameters. *Journal of the Royal Statistical Society*, 1980.

[4] The Standard Performance Evaluation Corporation. Specweb96. http://www.specbench.org/org/web96/.

[5] M.E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, May 1996.

[6] C.A. Cunha, A. Bestavros, and M.E. Crovella. Characteristics of www client-based traces. Technical Report TR-95-010, Boston University Department of Computer Science, April 1995.

[7] R. B. D'Agostino and M. A. Stephens, editors. *Goodness-of-Fit Techniques.* Marcel Dekker, Inc., 1986.

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

[9] A. Erramilli, O. Narayan, and W. Willinger. Experimental queueing analysis with long-range dependent packet traffic. *IEEE/ACM Transactions on Networking*, 4(2):209–223, April 1996.

[10] A. Feldmann. Modelling characteristics of tcp connections. Technical report, AT&T Laboratories, 1996.

[11] N. Johnson, S. Kotz, and N. Balakrishnan. *Discrete Univariate Distributions.* John Wiley and Sons, Inc., 1995.

[12] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, pages 2:1–15, 1994.

[13] R. Mattson, J. Gecsei, D. Slutz, and I. Traiger. Evaluation techniques and storage hierarchies. *IBM Systems Journal*, 9:78–117, 1970.

[14] J.C. Mogul. Network behavior of a busy web server and its clients. Technical Report WRL 95/5, DEC Western Research Laboratory, Palo Alto, CA, 1995.

[15] University of Minnesota. Gstone version 1. http://web66.coled.umn.edu/gstone/info.html.

[16] Kihong Park, Gi Tae Kim, and Mark E. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proceedings of the Fourth International Conference on Network Protocols (ICNP'96)*, pages 171–180, October 1996.

[17] Vern Paxson. Empirically-derived analytic models of wide-area tcp connections. *IEEE/ACM Transactions on Networking*, 1994.

[18] S. Pederson and M. Johnson. Estimating model discrepancy. *Technometrics*, 1990.

[19] Gene Trent and Mark Sake. Webstone: The first generation in http server benchmarking, February 1995. Silicon Graphics White Paper.

[20] Walter Willinger, Murad S. Taqqu, Robert Sherman, and Daniel V. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, February 1997.

[21] G. K. Zipf. *Human Behavior and the Principle of Least-Effort.* Addison-Wesley, Cambridge, MA, 1949.