Characterizing Temporal Locality and its Impact on Web Server Performance

Ludmila Cherkasova
Hewlett-Packard Laboratories
1501 Page Mill Road,
Palo Alto, CA 94303
cherkasova@hpl.hp.com

Gianfranco Ciardo *
Department of Computer Science
College of William and Mary
Williamsburg, VA 23187-8795, USA
ciardo@cs.wm.edu

Abstract *The presence of temporal locality in web traces has been long recognized. However, the close proximity of requests for the same file in a trace can be attributed to two orthogonal reasons:* long-term popularity *and* short-term correlation. *The former reflects the fact that requests for a popular document appear "frequently" thus they are likely to be "close" in an absolute sense. The latter reflects the fact that requests for a given document might concentrate around particular points in the trace due to a variety of reasons, such as deadlines or surges in user interests, hence it focuses on "relative" closeness. We introduce a new measure of temporal locality, the* scaled stack distance, *which is insensitive to popularity and captures instead the impact of short-term correlation, and use it to parametrize a synthetic trace generator. Then, we validate the appropriateness of this quantity by comparing the file and byte miss ratios corresponding to either the original or the synthetic traces.*

1 Introduction

The web is increasingly becoming a core element of business strategy, and understanding the nature of web server workloads is crucial to properly designing and provisioning current and future services. Issues of workload analysis, performance modeling, and capacity planning become ever more critical. Previous studies identified different types of *locality* in web traffic. *Static locality* or concentration of references [2]: 10% of the files accessed on the server typically account for 90% of the server requests and 90% of the bytes transferred. *Temporal locality* of references [1, 3]: recently accessed documents are more likely to be referenced in the near future. *Spatial locality* of references [1, 3]: there can be a correlation structure in the reference stream.

These strongly influence the traffic access patterns in web servers. Understanding the nature of locality will help the design of more efficient middleware for caching, load balancing, and content distribution systems.

Three main elements define web server performance: the number of requests the server must process, the number of bytes the server must transfer from disk, and the number of bytes the web server must transfer to the network. It is well known that web server performance greatly depends on efficient RAM (web cache) usage. The typical measure of web cache efficiency is the *(file) hit ratio*: the fraction of times (over all accesses) the requested file was found in the cache. Since the files are of different size, another complementary metric is also important: the *byte hit ratio* – the fraction of "bytes" found in the cache among all the bytes accessed.

Upper bounds for memory requirements can be identified from a static workload analysis, but it is important to note that web server workload exhibits high concentration of references (i.e., a high percentage of requests comes to a small percentage of the files), and that a high percentage (40%-70%) of rarely accessed files often accounts for a majority of bytes transferred. Authors in [1] state that file requests are clustered and that the stack distance characterizing temporal locality has a lognormal distribution. This introduces an important additional parameter for the characterization of web server workloads.

A good model of reference locality should allow us to generate synthetic web traces that accurately "mimic" the characteristics of the real web server logs affecting system performance. Thus, in the first part of this paper, we analyze *static* locality in web server workloads. Our study is based on four access logs from very different servers. The static locality characterization is based on the *frequency-size profile* of the files in the logs. The second part of the paper concentrates instead on the characterization of *temporal locality*, for which we introduce a new measure, the *scaled stack distance*, capturing the impact of short-term correlation. Finally, we merge these two characterizations (static and temporal locality) by using frequency, size, and stack distance information to generate a synthetic trace. To validate how well we capture the original trace characterization, we compare the file and byte miss ratios corresponding to the original and the synthetic traces.

*This work was performed while G. Ciardo was on a sabbatical visit at HP Labs.

2 Data collection sites

We use four access logs from very different servers:
1. HP WebHosting site (**WH**), which provides service to internal customers. Our logs cover the four-month period April–July, 1999. For our analysis, we chose the month of May, which is representative for this site.
2. OpenView site (www.openview.hp.com, **OV**), which provides the complete coverage on OpenView solutions from HP: product descriptions, white papers, demos illustrating product usage, software packages, business related events, etc. The log covers the period from the end of November, 1999 to the middle of February, 2000.
3. External HPLabs site (www.hpl.hp.com, **HPL**), which provides information about HP Laboratories, its current projects and research directions, and lists current job openings. It also provides access to an archive of published HPLabs research reports and to personal web pages. The log was collected during February, 2000.
4. HP site (www.hp.com, **HP**), which provides information about HP: HP business news, major HP events, detailed coverage of most software and hardware products, and press related news. The log covers a few hours (as this is business-sensitive data, we cannot be more specific) during February, 2000, and is a time-sorted composition of multiple logs collected on several web servers supporting the HP.com site.

3 Access log analysis

3.1 Raw data

The access log records information about all the requests processed by the server. Each line describes a single request for a document (file), specifying the name of the host machine making the request, the timestamp the request was made, the filename of the requested document and the size of the reply in bytes. The entry also provides information about the server's response to this request. Since only *successful* responses with code 200 are responsible for the files transferred by the server, we concentrate our analysis on these responses. Thus, in the rest of the paper, we use reduced access logs, with *successful*, "200 code" responses only. The following table summarizes this information.

	WH	OV	HPL	HP
Duration	1 month	2.5 months	1 month	few hours
Requests	952,300	3,423,225	1,877,490	14,825,457

These logs show quite different workloads. The WH, OV, and HPL sites have somewhat comparable numbers of requests (if normalized per month), while the HP site has three orders of magnitude heavier traffic.

3.2 "90% percentile" characterization

Several studies [1, 2, 3, 5] address the characterization of web workloads. [2] shows that web traffic exhibits a strong concentration of references: "10% of the files accessed on the server typically account for 90% of the server requests and 90% of the bytes transferred". We call this "static" locality because it can be characterized based only on per-file frequency-size information.

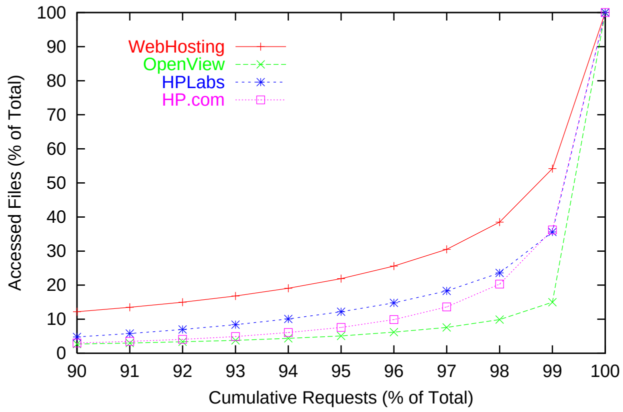

Figure 1: Concentration of references.

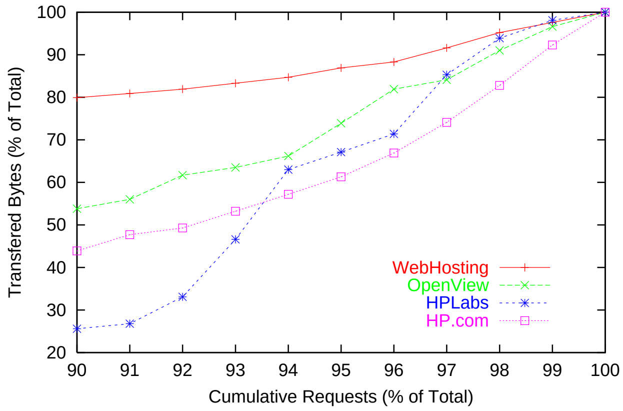

Figure 2: "Bytes-transferred" characterization.

Fig. 1 shows the "concentration of references" for the four access logs used in our study. For three sites (OV, HPL, and HP), 2%-4% of the files account for 90% of the server requests. The WH site exhibits less "reference locality": 12% of the overall file set is accessed by 90% of its most popular requests. Fig. 2 shows the bytes transferred due to these requests for all four access logs used in our study. The bytes transferred due to these requests vary over a much broader range: from 26% for the HPL site to 80% for the WH site, stressing the importance of this complementary metric.

3.3 Site profile

For each access log, we build a site profile by evaluating the following characteristics during the observed period:

- *WS*, the combined size of all the accessed files (in *bytes*, so-called "working set");
- *BT*, the combined number of "bytes transferred";

- $(\mathbf{fr}, \mathbf{s})$, the table of all the F accessed files, describing their frequency (number of accesses) and size.

WS characterizes the memory requirements of the site, while BT gives an approximation of the load offered to a server by the traffic to the site. These parameters provide a high-level characterization of web sites and their system resource requirements. The following table shows these parameters for our sites' working sets. From it, "at-a-glance" site specifics can be observed.

	WH	OV	HPL	HP
WS (MB)	865.8	5,970.8	1,607.1	4,396.2
BT (GB)	21.3	1,079.0	43.3	72.3
NumFiles	17,489	10,253	21,651	114,388

If we compare the characteristics of the OV and HP sites, there is a drastic difference in the number of accessed files and their cumulative sizes. The OV working set is the largest of the four sites considered, while its file set (number of accessed files) is the smallest, more than 10 times smaller than for the HP site. In spite of comparable number of requests (normalized per month) for the WH, OV, and HPL sites, the number of bytes transferred by the OV server is almost 20 times greater than for the WH and HPL servers, but still an order of magnitude less than the bytes transferred by the HP site. The number of bytes transferred (in addition to the number of hits) provides valuable insight into the traffic the server needs to support.

3.4 File and request size distribution

There is a high degree of variation in documents stored and accessed on different web servers. To some extent, this depends on the role and objective of the site. Thus, the HP site has many short updates on HP business news, different HP events, promotional coverage of newly introduced software and hardware products, and press-related news. On the other hand, the HPL site provides access to an archive of published HPLabs research reports (postscript and pdf files) which tend to be rather large.

We analyze both file size distribution (i.e., the sizes of documents stored on the site) and the request size distribution (i.e., the request sizes for the most frequently accessed files), using the $(\mathbf{fr}, \mathbf{s})$ table of all accessed files (see Sect. 3.3) sorted by increasing file size. Due to space limitations, we omit these figures. For details, see [6]. For example, the OV file size distribution exhibits three main groups of files:

- "small" files between 100 bytes and 10 Kbytes (20%);
- "medium" files between 10 and 15 Kbytes (65%);
- "large" files between 15 Kbytes to 126 Mbyte (15%).

The HP site has the smallest set of large documents: only 3% of the files are larger than 100 Kbytes.

The next table shows few points of the file size distributions. These accurately capture the file set specifics.

Web Site	Average File Size (KB)					Max. File Size (MB)
	30%	60%	90%	99%	100%	
WH	1.1	2.9	8.7	22.4	50.7	19.7
OV	6.3	9.0	11.1	228.1	596.3	126.0
HPL	1.2	2.8	9.0	34.4	76.0	75.2
HP	1.7	6.0	10.5	18.3	39.4	44.0

Clearly, the OV site exhibits quite a different profile from the other three sites we consider. However, to understand how much this matters, we need to analyze the request size distribution as well.

Not all files are equally "popular": some of them (related to current news, interesting papers or press releases, and new s/w or h/w products) are extremely "hot" and requested by many users, while others are of interest to a small group only.

To plot the request size distribution (with respect to most "popular" files), we use the $(\mathbf{fr}, \mathbf{s})$ table of all accessed files sorted in decreasing file frequency order. The request size distribution looks quite different from the file size distribution. For example, about 90% of the requests to the HP site account for files of size less than 1 Kbyte and the number of large files transferred is negligible. The OV site has instead a fairly large percentage of large requests: files larger than 50 Kbytes constitute 10% of the requests. The OV site is a representative of a site with many "popular" large files. The next table shows a few points of the request size distribution:

Web Site	Average Request Size (KB)					Max. Req. Size (MB)
	30%	60%	90%	99%	100%	
WH	0.8	1.6	4.6.	9.5	22.9	19.7
OV	0.3	0.4	2.0	53.4	322.8	126.0
HPL	0.6	0.8	3.1	6.9	23.6	75.2
HP	0.3	0.5	1.1	2.7	5.0	44.0

In spite of the difference in file sets, the profile for the most popular 90% of the requests for all the sites under consideration look somewhat similar: the most popular 90% of the requests come for rather "small" size files: from 1.1 Kbytes (HP site) to 4.6 Kbytes (WH site) on average. The remaining 10% of the requests introduce a lot of variation in the site profiles: the OV site exhibits a significant percentage of the requests due to large files.

Additionally, as it was shown in Sect. 3.2, 90% of all the requests come to a small set of extremely popular files which account from 2% (OV site) to 12% (WH site) of all accessed files on those sites. So, the popular files are indeed very popular.

What is the profile for the remaining files? How many of the files are accessed once? For this analysis, we report the files accessed up to 1, 5, or 10 times.

Web Site	Files requested up to		
	1 time	5 times	10 times
WH	37.1%	62.7%	71.4%
OV	49.2%	70.9%	76.2%
HPL	48.8%	68.4%	74.5%
HP	28.2%	66.1%	76.4%

There is a significant percentage of "onetimers": from 28% (HP) to 49% (OV) of the files are accessed only

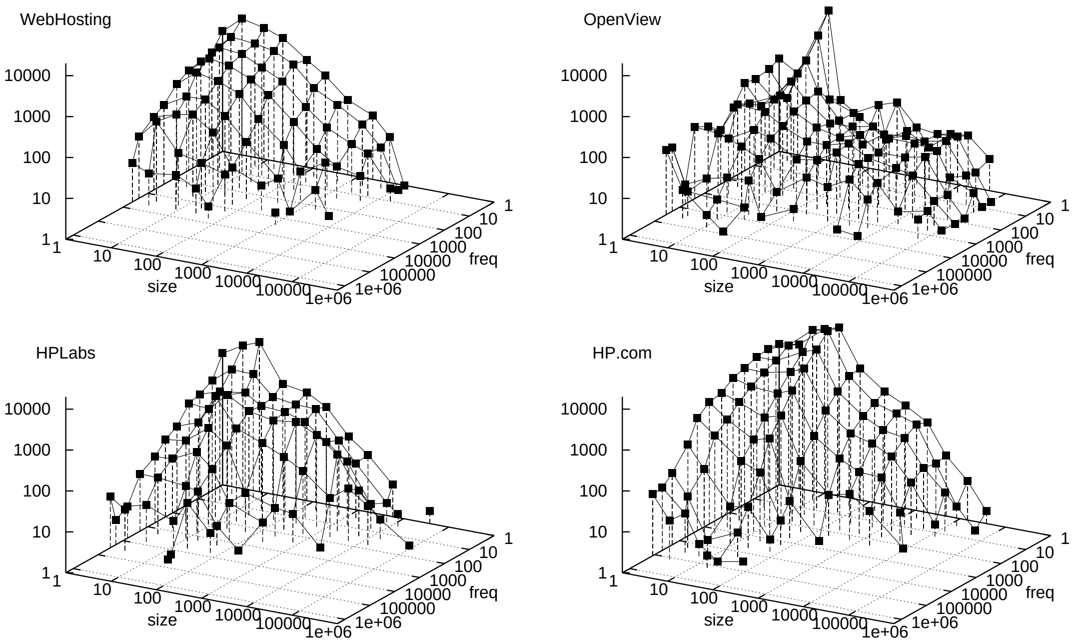

Figure 3: Frequency-size profiles.

once during the log, while "rarely accessed" files (no more than 10 times) account for 71%-76% of all files.

To visualize the frequency-size distribution of the requests, we use the 3-D plots shown in Fig. 3. These are obtained by placing the F files into frequency-size "buckets" of exponentially increasing width. The horizontal X and Y axes correspond to the frequency (number of requests) and size (Kbytes) bucket, while the vertical Z axis counts the number of files falling into that bucket. (a log scale is used on all axes).

The WH trace is nicely behaved: for a given size bucket, the number of files in each frequency bucket tends to decrease as the frequency increases while, for a given frequency bucket, the number of files in each size bucket tends to decrease as the size increases. The HPL and HP profiles also follow this pattern, although not as well. However, the OV profile has steep peaks corresponding to not-so-small or not-so-infrequent buckets, and a large plateau for sizes between 10^2 Kbytes and 10^2 Mbytes and frequencies between 1 and 10^3.

4 Temporal locality and web server cache performance

We are interested in "extracting " a small number of parameters from the real web server access log to generate synthetic traces with "performance characteristics" similar to those of the original one.

The typical measure of web server cache efficiency is the *hit ratio*: the percentage of requested files found in the cache. As files are of different size, another metric is also important: the *byte hit ratio*, the fraction of "bytes" returned from the cache among all the bytes accessed. In our study, we use the complementary quantities, *file miss ratio* and *byte miss ratio*, to measure the "closeness" of the synthetic to the original traces.

If we restrict ourselves to using the $(\mathbf{fr}, \mathbf{s})$ profile to capture the trace, we might assume that the R requests are uniformly distributed over the trace. More precisely, we generate a synthetic trace corresponding to a random permutation of the trace

$$(\underbrace{1,1,\ldots,1}_{\mathbf{fr}_1},\underbrace{2,2,\ldots,2}_{\mathbf{fr}_2},\ldots,\underbrace{F,F,\ldots,F}_{\mathbf{fr}_F}) \tag{1}$$

and feed it to an analysis program that computes the miss ratio $M(x)$, where x is the amount of RAM in the web server. Since the four traces we consider vary in their overall working sets, we choose to specify x as *a percentage* (3%, 5%, 10%, 20%, 30%, 40%, 60%, 80%, or 100%) *of the working set*, to *normalize* the curves. With a RAM equal to 100% of the working set, we have only the F "cold misses" required to bring the files from disk into memory, while, when less RAM is available, "warm misses" become a factor.

However, the *uniform assumption* results in a *pessimistic assessment* of both file and byte miss ratio. Since the original and the synthetic traces contain exactly the same number of occurrences $\mathbf{fr}_f$ for each file f, the difference cannot be due a mismatch in the "popularity" pattern. In addition to the temporal locality naturally arising from the fact that the requests for a popular file appear "close" to each other in the trace, requests for a given file must then also tend to cluster together due to short-term correlation. This important distinction between *long-term popularity* and *short-term temporal correlations* has also been observed recently in [7]: indeed we have adopted their terminology. In the following section, we discuss approaches to capture this short-term correlation.

5 Stack distance

To improve the characterization of web file requests, we resort to the concept of stack distance, introduced in [9] and later adopted by [1] for the study of temporal locality in web traces. Assume that the files are placed on a stack such that, whenever f is requested, it is either pulled from its position in the stack and placed on the top, or it is added to the stack if it is not yet in it. The *stack distance* for the request is then the distance of f from the top in the former case (a nonnegative integer), or *undefined* (or ∞) in the latter case.

Thus, starting with an empty stack, the trace $(f_1, f_2, \ldots, f_R)$ defines a sequence $(d_1, d_2, \ldots, d_R)$ of trace distances, exactly F of which are ∞ (corresponding to the F cold misses). The stack distances $(d_1, d_2, \ldots, d_R)$ affect the miss ratios, hence the performance of the machine serving the requests. For example, if most stack distances are quite small, files are nicely clustered over the trace, and most requests will not require a disk access. Indeed, the ideal trace shown in (1) will only cause cold misses, as long as the RAM is large enough to contain the largest file in the trace: this trace has perfect short-term correlation.

We now focus on the definition of a web request profile that captures, in addition to the frequency-size distribution of the trace, also its stack distance behavior. To test its usefulness, we use a synthetic trace generator having this profile as input. We then place the generator in pipeline with the same analysis program used to compute the file and byte miss ratios for the original and synthetic case under the uniform assumption, and compare the results obtained.

5.1 Synthetic trace generator

The key idea of our synthetic generator is the use of a stack data structure to schedule the order in which the requests are generated. Fig. 4 shows the pseudocode for our algorithm. In addition to the global variables describing the request profile (R and F, integers, **fr**, an integer vector, and parameters to describe the stack distance distribution), it uses the local variable $\mathbf{o} = [\mathbf{o}_1, \ldots, \mathbf{o}_F]$, an integer vector of the outstanding requests for each file, and $\mathbf{t}$, an integer stack stored as a vector of fixed dimension F.

The two functions used in our algorithm, *InitializeStack* and *GenerateStackDistance*, are

```
procedure SyntheticTrace(fr) is
  InitializeStack(t);
  top ← F;                          • initial stack height
  o ← fr;              • outstanding requests for each file
  while top > 0               • still requests to generate
    f ← t_top;                 • f is the file to be emitted
    output a request for f;
    o_f ← o_f − 1;
    if o_f = 0 then           • eliminate f from the stack
      top ← top − 1;
    else                         • push down f in the stack
      d ← GenerateStackDistance(top, f);
      for i ← top − 1 to top − d do
        t_{i+1} ← t_i;
      end for;
      t_{top−d} ← f;
    end if;
  end while;
end procedure;
```

Figure 4: Our algorithm to generate a trace of length R.

critical to its correct probabilistic behavior. The call *InitializeStack*$(\mathbf{t})$ initializes the stack $\mathbf{t}$ with the F files $\{1, \ldots, F\}$ in some order. This order affects the placement of file requests in the synthetic trace. We found that initializing the stack probabilistically according to the information in **fr** works well: we place file f on the top of the stack with probability $\mathbf{fr}_f/R$; then, with probability $\mathbf{fr}_g/(R - \mathbf{fr}_f)$ we place file g in the second position, where f is the file chosen for the first position, and so on. The key decision, though, is the definition of the function that attempts to recreate the stack distance patterns presented in the original trace. The call *GenerateStackDistance*(top, f) computes and returns a stack distance d for file f when the stack contains top files, and is used to push file f d positions down from the top. We experimented with three methods, corresponding to the way we captured the stack distance information when analyzing the original trace.

5.2 Scaled stack distance

The lognormal distribution [8] has been shown to be a good model for the stack distance distribution [1] of real web traces. Recall that a continuous random variable X has a lognormal distribution with parameters $a \in \mathbb{R}$ and $b \in \mathbb{R}^+$, we write $X \sim \text{Lognormal}(a, b)$, if and only if $Z = (\log X - a)/b$ has a standard normal distribution, $Z \sim Normal(0, 1)$. Given the observations $(X_1, X_2, \ldots, X_n)$, the maximum-likelihood estimators for the parameters a and b of the lognormal distribution fitting them are [8]:

$$a = \frac{\sum_{i=1}^{n} \ln X_i}{n} \qquad b = \sqrt{\frac{\sum_{i=1}^{n} (\ln X_i)^2}{n} - a^2} \qquad (2)$$

In our first experiment to capture the stack distance information and parametrize the synthetic generator, we use the $R-F$ finite stack distances in $(d_1, d_2, \ldots, d_R)$ to obtain a^u and b^u, the parameters of the lognormal distribution according to (2). The "u" stand for "unscaled", as it will be clear shortly. Then, *GenerateStackDistance*(top, f) generates a random deviate according to the distribution Lognormal(a^u, b^u), regardless of the value of f, and return its rounded value d (if $d \geq top$, it is truncated to $top - 1$, the maximum distance a file on the top can be pushed down the stack).

The miss ratios resulting from this approach, which we call *unscaled stack distance (USD)*, are shown in Fig. 6. It is clear that they are grossly optimistic. To understand the reason for this error, we need to recall that the distribution of the stack distance is affected by two factors: the long-term popularity and the short-term correlation. Considering the plot on the top left of Fig. 5 (for the WH trace), we see that the number of requests for a given file can vary dramatically: from 7,072 for the most popular file, to 1, for each of the 6,493 onetimers in the trace. By collecting the stack distances ignoring the identity of the file they refer to, the resulting lognormal distribution has a large variance: popular files will tend to experience much shorter stack distances than rare files. Thus, a call to *GenerateStackDistance* will often return a large value which will then effectively put the file to the bottom of the stack. After a relatively small number of requests have been generated, most of the files with few requests will have been removed from the stack, so that only the most popular files remain from that point on, and the rest of the trace is mostly filled with them. Clearly, this results in no misses as soon as the small set of frequent files fits in memory. The plot in the same figure on the top right shows the empirical distribution of the unscaled stack distance for all files together and its lognormal fit.

To avoid the effect of popularity on our stack distance observations, we could observe the stack distances experienced by each individual file separately. For example, the two plots in the middle of Fig. 5 show the empirical distribution of the stack distance for the most popular file and for a generic file, and their lognormal fits. Essentially, the shape of the empirical (or the fitted) pmf for all the files together is obtained by merging the pmfs for each individual file f using as weight its number of requests $\mathbf{fr}_f$; however, these pmfs have widely varying averages, since frequent files tend to experience small stack distances, while infrequent files tend to experience larger stack distances.

We consider two different ways to untie file popularity from the short-term correlation captured by the pmfs for the stack distances of individual files. The "most expensive" approach collects the distribution of the stack distances experienced by each file individually. More precisely, for each f, we compute the values $\sum_{i=2}^{\mathbf{fr}_f} \ln d_i^f$ and $\sum_{i=2}^{\mathbf{fr}_f} (\ln d_i^f)^2$ where $(d_1^f, d_2^f, \ldots d_{\mathbf{fr}_f}^f)$ are the stack distances experienced by file f. Then, we apply (2) for each file f, obtaining two vectors $\mathbf{a} = [\mathbf{a}_1, \ldots, \mathbf{a}_F]$ and $\mathbf{b} = [\mathbf{b}_1, \ldots, \mathbf{b}_F]$. In our algorithm, *GenerateStackDistance*(top, f) returns the rounded value of a random deviate sampled from the distribution Lognormal$(\mathbf{a}_f, \mathbf{b}_f)$.

A less expensive, but intellectually appealing, approach is instead to observe that, if files were uniformly distributed over the trace, the expected stack distance experienced by a file f would be

$$\mathbf{d}_f = \textstyle\sum_{g \neq f} \mathbf{fr}_g / (\mathbf{fr}_g + \mathbf{fr}_f - 1)^{-1},$$

a quantity that can be obtained exclusively from the vector $\mathbf{fr}$. The derivation of this expression follows a reasoning analogous to the *Coupon Collecting Problem* discussed in [10, p. 261-3]. With this second approach, we then define the *scaled stack distance* by dividing the actual stack distance d experienced by f by its expected value in the uniform case: $d^{scaled} = d/\mathbf{d}_f$. Intuitively, an observed stack distance of 12 for a frequent file f whose expected stack distance under the uniform assumption is $\mathbf{d}_f = 10$ is actually "longer" than an observed stack distance of 800 for not-so-frequent file g whose expected stack distance under the uniform assumption is $\mathbf{d}_g = 1000$. Thus, we are effectively observing how much the actual stack distances depart from what we would expect to observe on a trace under the uniform assumption, in a dimensionless way. This way of observing the data eliminates the problems due to merging the "unscaled" pmfs of each file. The sequence of $R-F$ finite scaled distances, ignoring the file identity, can then be used to obtain the parameters a^s and b^s of a lognormal distribution, again using (2). The resulting empirical scaled distance pmf (for the WH trace) and its lognormal fit are shown at the bottom-right of Fig. 5. On the left, we show instead the values of $\mathbf{d}_f$ for the files that are not onetimers, in decreasing popularity order. Thus, *GenerateStackDistance*(top, f) generates a random deviate x according to the distribution Lognormal(a^s, b^s) and scales it back by multiplying it by $\mathbf{d}_f$. The results for the miss ratios corresponding to these two approaches, individual stack distance (ISD) and scaled stack distance (SSD), are shown in Fig. 6. Both of them provide a closer match of the original trace than the synthetic trace under the uniform assumption, also shown in Fig. 6. Indeed, the simpler SSD profile $(\mathbf{fr}, \mathbf{s}, a^s, b^s)$ performs at least as well as the ISD profile $(\mathbf{fr}, \mathbf{s}, \mathbf{a}, \mathbf{b})$. We attribute this phenomenon to the variance of the distribution: while, in principle, $\mathbf{a}$ and $\mathbf{b}$ give more detailed information than a^s and b^s, each of their entries $\mathbf{a}_f$ and $\mathbf{b}_f$ is computed using a much smaller set of $\mathbf{fr}_f - 1$ data points, while a^s and b^s are obtained using $R-F$ data points about the scaled stack distance of all files together.

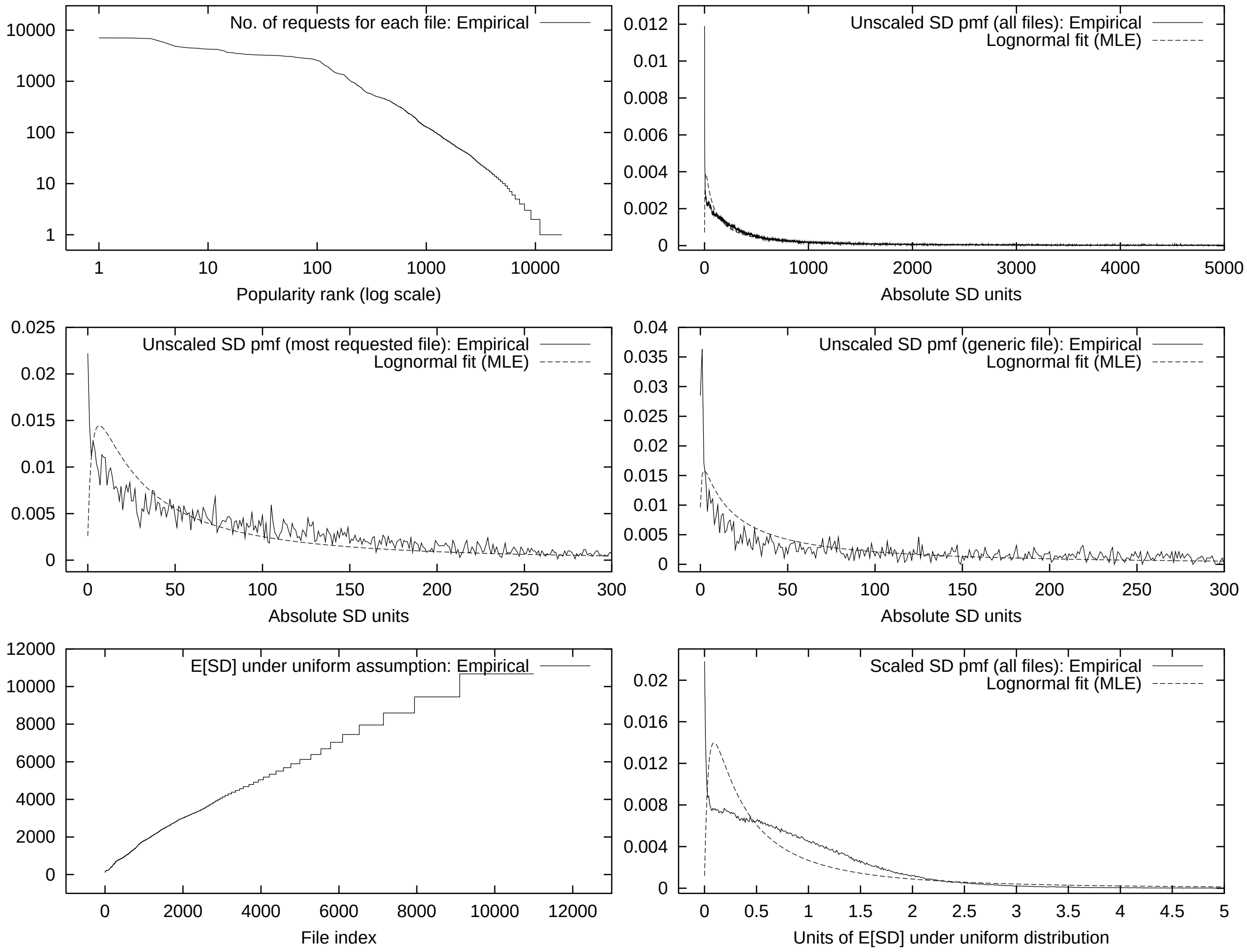

Figure 5: Unscaled and scaled stack distance and supporting plots (WH trace).

6 Conclusion

In this paper, we analyze *static* and *temporal* locality in web server workloads. We introduce a new measure of temporal locality, the *scaled stack distance*, which is insensitive to popularity and captures instead the impact of short-term correlation. We merge the two characterizations (static and temporal locality) by using frequency, size, and stack distance distributional information to generate a synthetic trace. This results in a model of reference locality and allows us to generate synthetic web traces that accurately "mimic" essential characteristics of the real web server logs for future performance analysis studies.

Acknowledgments

This study could not have been possible without server access logs and help provided by Guy Mathews, Dan Schram, Wai Lam, Len Weisberg, and Mike Rodriquez. Their help is highly appreciated.

We would also like to thank Prof. Stephen K. Park of the College of William and Mary for making available his library of random number generation and probability distribution function routines.

References

[1] V. Almeida, A. Bestavros, M. Crovella, and A. Oliviera. Characterizing reference locality in the WWW. In *Proc. 4th Int. Conf. Parallel and Distributed Information Systems (PFIS)*, pp.92–106. IEEE Comp. Soc. Press,1996.

[2] M. Arlitt and C. Williamson. Web server workload characterization: the search for invariants. In Proceedings of the ACM SIGMETRICS '96 Conference, Philadelphia, PA , May 1996. ACM.

[3] P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in web client access patterns: characteristics and caching implications. Technical Report, Boston University, TR-1998-023, 1998.

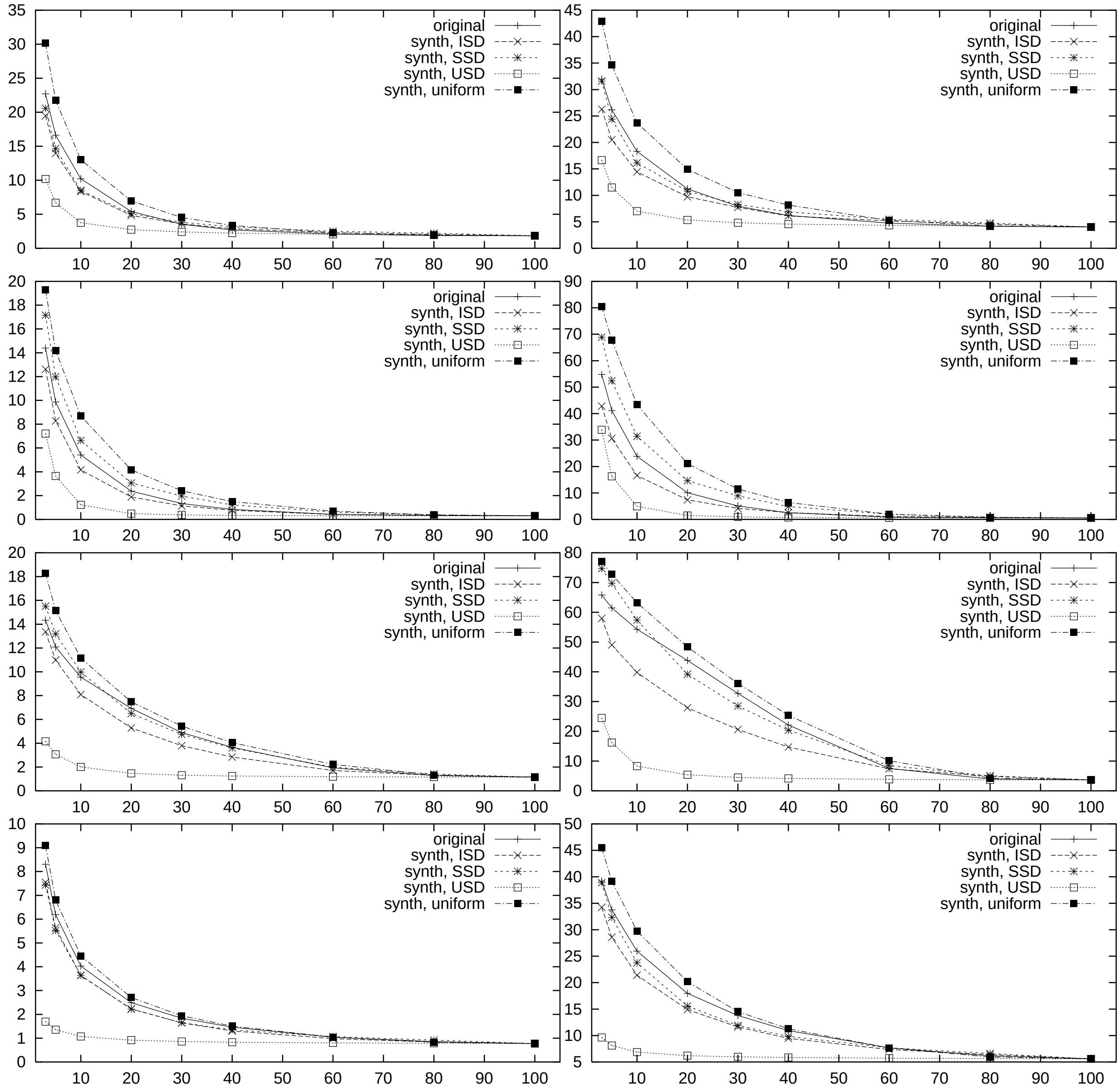

Figure 6: File (left) and byte (right) miss ratios (WH, OV, HPL, and HP, top to bottom).

[4] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. Technical Report, Boston University, TR-1997-006, 1997.

[5] A. Bestavros, R. Carter, M. Crovella, C. Cunha, A. Heddaya, and S. Mirdad. Application-level document caching in the internet. Technical Report, Boston University, TR-1998-023, 1995.

[6] L. Cherkasova, G. Ciardo. Characterizing Temporal Locality and its Impact on Web Server Performance HP Laboratories Report No. HPL-2000-82, July, 2000.

[7] S. Jin and A. Bestavros. Temporal locality in web request streams. In *Proc. 2000 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, Santa Clara, CA, June 2000.

[8] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis.* McGraw-Hill, 1982.

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

[10] S. M. Ross. *Introduction to probability models.* Academic Press, Sand Diego, CA, 1997. 6th Ed.