Characterizing Temporal Locality and its Impact on Web Server Performance

Ludmila Cherkasova, Gianfranco Ciardo[1]
Computer Systems and Technology Laboratory
HP Laboratories Palo Alto
HPL-2000-82
June 26, 2000*

E-mail: cherkasova@hpl.hp.com
ciardo@cs.wm.edu

work load case studies, web server logs, temporal locality, the scaled stack distance, synthetic web trace generators, performance analysis

The presence of temporal locality in web request traces has long been recognized, and has been incorporated in synthetic web trace generators. 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 simply appear "very frequently" thus they are likely to be "close" in an absolute sense. The latter reflects instead 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 swings in user interests, hence it focuses on "relative" closeness.

In this work, 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 then use the scaled stack distance observed in the original trace to parametrize a synthetic trace generator. Finally, we validate the appropriateness of using this quantity by comparing the file and byte miss ratios corresponding to either the original or the synthetic traces.

[1] Department of Computer Science, College of William and Mary, Williamsburg, VA 23187-8795, USA

Characterizing Temporal Locality and its Impact on Web Server Performance

Ludmila Cherkasova
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304-1126, USA
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 request traces has long been recognized, and has been incorporated in synthetic web trace generators. 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 simply appear "very frequently" thus they are likely to be "close" in an absolute sense. The latter reflects instead 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 swings in user interests, hence it focuses on "relative" closeness.

In this work, 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 then use the scaled stack distance observed in the original trace to parametrize a synthetic trace generator. Finally, we validate the appropriateness of using this quantity by comparing the file and byte miss ratios corresponding to either the original or the synthetic traces.

Contents

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

1 Introduction

Understanding the nature of the web servers' workloads is crucial to properly designing and provisioning current and future web services. Web increasingly becomes a core element of the business strategy. With the rapid growth of web traffic, most popular web sites need to scale up their server capacities. 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], the observation was made that 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], which implies that recently accessed documents are more likely to be referenced in the near future;
- *spatial locality* of references [1, 3] which implies a correlation structure in the reference stream.

All these different types of locality strongly influence the traffic access patterns in web servers, and define individual traffic access profiles specific for particular web sites. Understanding the nature of locality will help to design 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 (cache) usage. A web server works faster when it pulls pages from a cache in RAM. Moreover, its throughput is much higher too. The typical measure of web cache efficiency is the *(file) hit ratio*: the fraction of times (over all accesses) the file was found in the cache. Since the files are of different size, another complementary metric is also important: *byte hit ratio* – the fraction of "bytes" returned from the cache among all the bytes accessed.

The upper bounds for memory requirements can be identified from the static workload analysis. The interesting (to some extent, contradictory issue) is that web server workload exhibits high concentration of references (i.e., a high percentage of requests account for a small percentage of the files), and the fact that there is a very high percentage (40%-70%) of rarely accessed files, which often account for a majority of bytes transferred. Authors in [1] state that file requests are clustered and that the stack distance, characterizing temporal locality, has lognormal distribution. Temporal locality introduces additional important parameter in web server workload characterization.

Good models of reference locality will allow us to generate synthetic web traces which accurately "mimic" essential characteristics of the real web server logs for performance analysis studies.

In the first part of this paper, we analyze *static* locality in web server workloads. Our case study is based on four access logs from very different servers (Section 2 describes them in details). The static locality characterization is based on the *frequency-size profile* of the files from the web server log: we rank files through their frequency (number of times they are requested in a trace) and their size (which is critical in determining cache behavior and server memory requirements).

The second part of the paper concentrates on *temporal locality* characterization. We introduce a new measure of temporal locality, the *scaled stack distance*, which captures impact of short-term correlation.

Finally, we merge these two characterization (static and temporal locality) by using frequency, size, and stack distance distributional information to generate a synthetic trace. To validate how well we captured the original trace characterization, we compared the file and byte miss ratios corresponding to either the original or the synthetic traces.

2 Data Collection Sites

In our study, we used four access logs from very different servers:

- HP WebHosting site, which provides service to internal customers. Our logs cover a four-month period, from April to July, 1999. In April, the service had 71 hosted "subsites" while, by the end of July, it had 89 hosted web sites. Interestingly, in spite of the updated and modified content during these months, the server' logs exhibit many common features in their "traffic profile" or, so-called, site "personality". For our analysis, we chose the month of May, which represents well the specifics of the site.

- OpenView site (www.openview.hp.com): which provides the complete coverage on OpenView solutions from HP: the product descriptions, white papers, demos illustrating the products usage, the software packages, business related events, conferences on the topic, etc. The log covers a duration of 2.5 months, from the end of November, 1999 to the middle of February, 2000.
- HPLabs site (www.hpl.hp.com), which provides information about HP Laboratories, its current projects and research directions, lists current job openings. It also provides access to an archive of published HPLabs research reports and hosts a collection of personal web pages where researches describe their old and current projects as well as share "parts" of their personal lives. The access log was collected during February, 2000.
- HP site (www.hp.com), which provides diverse information about HP: HP business news, major HP events, detailed coverage of the most software and hardware products, and the press related news. The access log covers a few hours[1] during February, 2000, and is a composition of multiple access logs collected on several web servers supporting the HP.com site (sorted by time).

3 Access Log Analysis

3.1 Raw data

The access log records information about all the requests processed by server. Each line from the access log provides a description on a single request for a document (file). A typical entry contains the following fields:
`hostname - - [dd/mm/yyyy:hh:mm:ss tz] request status bytes`
Each log entry specifies the name of the host machine making the request, the timestamp the request was made, the filename of the requested document and size in bytes of the reply. The entry also provides the information about the server's response to this request. Status code 200 means that the request was successfully completed by the server. Status code 304 relates to the documents cached somewhere in the Internet (or by proxy caches) which send a "request-validation" whether the document was *"modified since"* the last requested time, no data bytes need to be transferred in this case. The rest of the codes specify "unsuccessful" requests which the web server was not able to satisfy (typically, a reason why the response was unsuccessful is given).

Since the *successful* responses with code 200 are responsible for all of the documents (files) transferred by the server, we will concentrate our analysis only on those responses. Thus, for the remaining analysis in the paper, we used reduced access logs, with *successful*, "200 code" responses only.

In Table 1, we summarize the main information about the reduced access logs, such as the access log duration, number of successful requests, and number of accessed files.

	WebHosting	OpenView	HPLabs	HP.com
Access Log Duration	1 month	2.5 months	1 month	few hours
Total Requests (200 code)	952,300	3,423,225	1,877,490	14,825,457

(1)

The four access logs provide information on web servers with different workload. HP WebHosting, OpenView, and HPLabs servers had somewhat comparable number of requests (if normalized per month). HP.com site had three orders of magnitude heavier traffic.

3.2 Traditional "90% Percentile" Characterization

Several studies [1, 2, 3, 5] address the characterization of web workloads. In [2], the authors showed 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 this characterization only uses basic per-file frequency-size information.

Figure 1 shows the "concentration of references" for the four access logs used in our study. For three sites (OpenView, HPLabs, and HP.com), 90% of the server requests come to only 2%-4% of the files. The WebHosting site exhibits less "reference locality": 90% of its most popular requests cover 12% of the overall file set.

Figure 2 shows the bytes transferred due to these requests for all four access logs used in our study. The

[1]As this is business-sensitive data, we cannot be more specific.

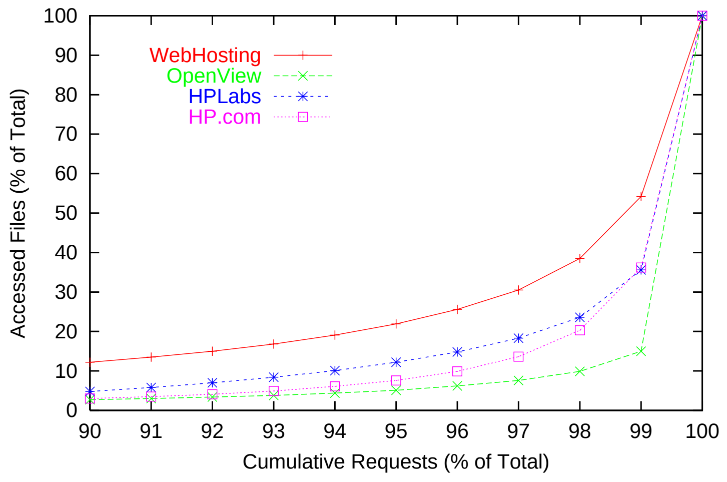

Figure 1: Four Traces Compared: Concentration of References.

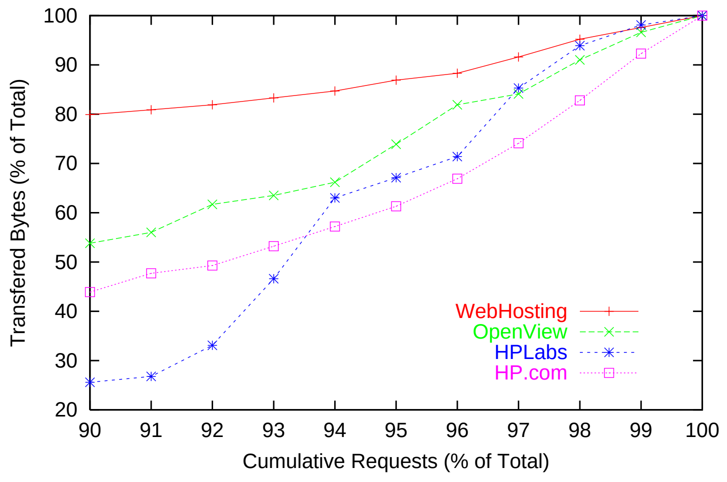

Figure 2: Four Traces Compared: "Bytes-Transferred" Characterization.

bytes transferred due to these requests vary in much broader range: from 26% for the HPLabs site to 80% for the WebHosting 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:

- WS - the combined size of all the accessed files (in *bytes* during the observed period, so-called "working set");
- BT - the "bytes transferred" during the observed period;
- $(\mathbf{fr}, \mathbf{s})$ - the table of all F accessed files with their frequency (number of times a file was accessed during the observed period) and their size.

The site working set characterizes the memory requirements of the site. If the working set fits in RAM, only the first request for a file will require a disk access (a "cold miss"), resulting in high server performance. The number of bytes transferred, 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. Table 2 shows the sites' working sets and the amount of bytes transferred. From the Table 2, high-level, "at-a-glance" site specifics can be observed.

	WebHosting	OpenView	HPLabs	HP.com
Working Set Size	865.8 MB	5,970.8 MB	1,607.1 MB	4,396.2 MB
Bytes Transferred	21.3 GB	1,079.0 GB	43.3 GB	72.3 GB
Number of Accessed Files	17,489	10,253	21,651	114,388

(2)

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

3.4 File Size 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.com 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 HPLabs site provides access to an archive of published HPLabs research reports (postscript and pdf files) which tend to be rather large files.

We will 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).

Figure 3 shows the file size distribution for our collection of four logs. To plot this distribution, we use the $(\mathbf{fr}, \mathbf{s})$ table of all accessed files (from the site profile in Section 3.3) sorted in increasing file size order.

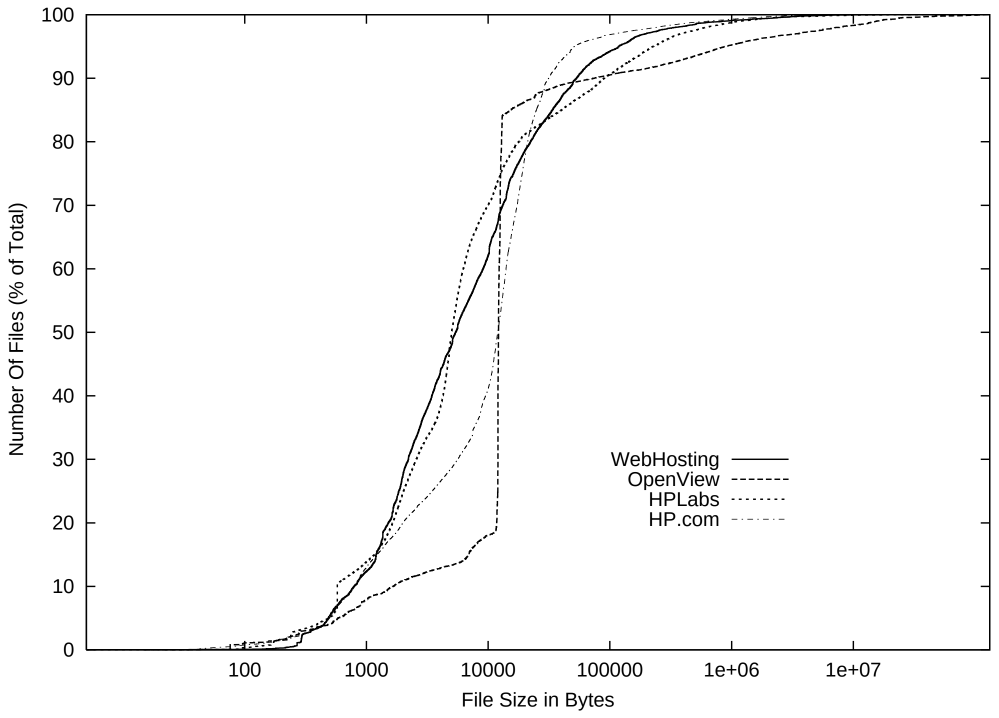

Figure 3: Four Traces Compared: File Size Distribution.

For example, the OpenView file size distribution clearly shows three main groups of files:

1. "small" files (with sizes between 100 bytes and 10 Kbytes), which account for about 20% of the files;
2. "medium" files (with sizes between 10 and 15 Kbytes), the largest group, with almost 65% of the files;
3. "large" files (with sizes 15 Kbytes to 126 Mbytes), accounting for the remaining 15% of the files.

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

Next, Table 3 shows few points of the file size distribution above (remember that to calculate these points we use the $(\mathbf{fr}, \mathbf{s})$ table of all accessed files sorted in increasing file size order).

- the average file size for 30% / 60% / 90% / 99% of the files;
- the average file size across all the files (100%);
- the maximum file size on the site.

Web Site	Average File Size					Maximum File Size
	30%	60%	90%	99%	100%	
WebHosting	1.1KB	2.9KB	8.7KB	22.4KB	50.7KB	19.7 MB
OpenView	6.3KB	9.0KB	11.1KB	228.1KB	596.3KB	126.0 MB
HPLabs	1.2KB	2.8KB	9.0KB	34.4KB	76.0KB	75.2 MB
HP.com	1.7KB	6.0KB	10.5KB	18.3KB	39.4KB	44.0 MB

(3)

These few points from the entire file size distribution accurately reflect the file set specifics. Clearly, the OpenView 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 the files are equally "popular": some of the files (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 can be of interest to a very small group only. To plot the request size distribution, shown in Figure 4, (with respect to most "popular" files), we use the $(\mathbf{fr}, \mathbf{s})$ table of all accessed files (from the site profile in Section 3.3) sorted in decreasing file frequency order.

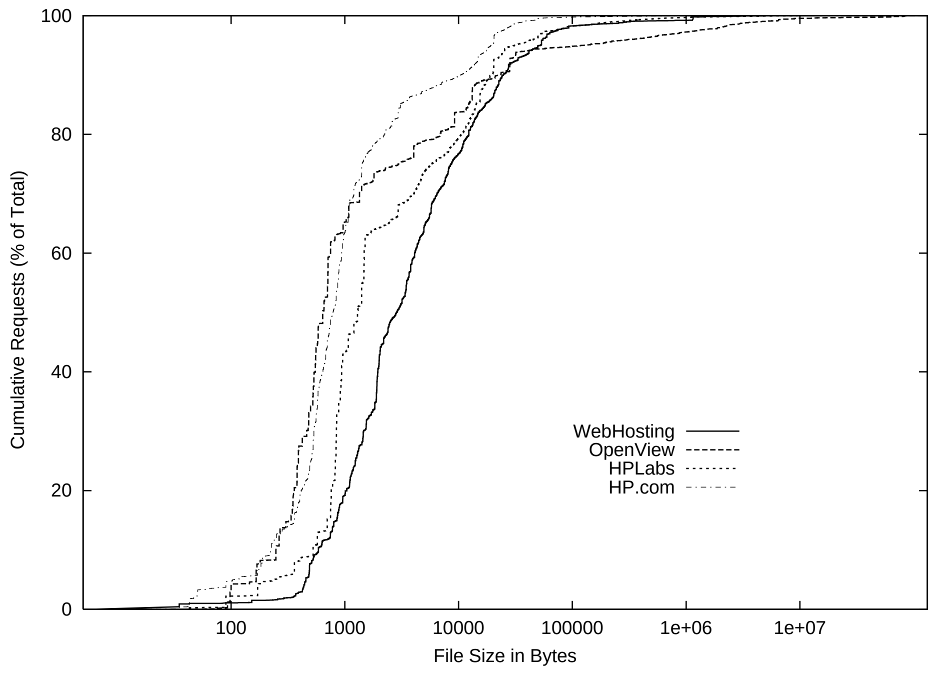

Figure 4: Four Traces Compared: Request Size Distribution.

The request size distribution does look quite different from the file size distribution for those sites. For example, most of the requests to the HP.com site (about 90% of total) account for small files, which are less than 1 Kbyte. The number of large files transferred from HP.com is negligible. The OpenView site has a fairly large percentage of large requests: files larger than 50 Kbytes constitute 10% of the requests. The OpenView site is an interesting representative of the site with many "very popular" large files.

Next, Table 4 shows a few points of the request size distribution above (to calculate these points we use the $(\mathbf{fr}, \mathbf{s})$ table sorted in decreasing file frequency order).

- the average response size for 30% / 60% / 90% / 99% of all the requests;
- the average request size across all the requests (100%);
- the maximum request size on the site.

Web Site	Average Request Size					Maximum Request
	30%	60%	90%	99%	100%	Size
WebHosting	0.8 KB	1.6 KB	4.6.KB	9.5 KB	22.9 KB	19.7 MB
OpenView	0.3 KB	0.4 KB	2.0 KB	53.4 KB	322.8 KB	126.0 MB
HPLabs	0.6 KB	0.8 KB	3.1 KB	6.9 KB	23.6 KB	75.2 MB
HP.com	0.3 KB	0.5 KB	1.1 KB	2.7 KB	5.0 KB	44.0 MB

(4)

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 1.1Kbytes (HP.com site) to 4.6 Kbytes (WebHosting site) on average. The remaining 10% of the requests introduce a lot of variation in the site profiles: OpenView exhibits a significant percentage of the requests due to very large files.

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

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

Web Site	Files Requested up to		
	1 time	5 times	10 times
WebHosting	37.1%	62.7%	71.4%
OpenView	49.2%	70.9%	76.2%
HPLabs	48.8%	68.4%	74.5%
HP.com	28.2%	66.1%	76.4%

(5)

There is a significant percentage of "onetimers": from 28% (HP.com site) to 49% (OpenView site) of the files are accessed only once for duration of the log. The collection of "rarely accessed" files (accessed no more than 10 times) accounts for 71%-76% of all the files.

To visualize the frequency-size distribution of the requests, we use the 3-D plots shown in Figure 5. These plots were 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. Note that a log scale is used on all three axes. File in the buckets corresponding to frequency equal one are onetimers.

It is interesting to note that the WebHosting trace is fairly nicely behaved, since, for a given size bucket, the number of files in each frequency bucket tends to decrease as the frequency increases and, vice versa, for a given frequency bucket, the number of files in each size bucket tends to decrease as the size increases. The HPLabs and the HP.com profiles also follow this pattern, although not as well. Finally, the OpenView profile is quite different, with steep peaks in correspondence 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.

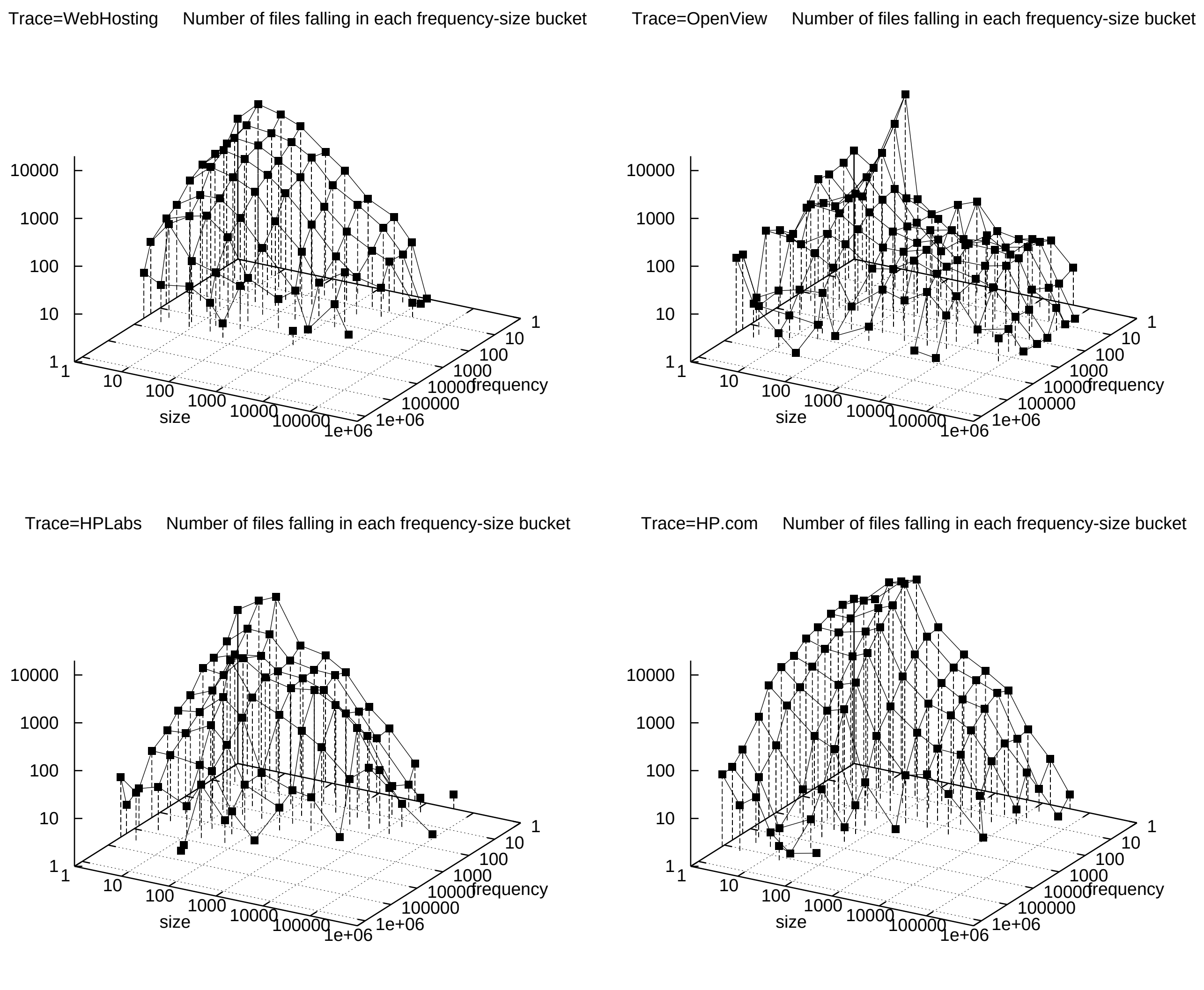

Figure 5: Frequency-size profiles

.

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 the synthetic trace with similar "performance characteristics" to the original web server log.

The typical measure of web server cache efficiency is a *hit ratio*: the number of times (from all the accesses) the requested file was found in the cache. Since the files are of different size, another complementary metric is also important: *byte hit ratio* - the number of "bytes" returned from the cache as a fraction of all the bytes accessed. In this work, we focus on two key quantities: *file miss ratio* and *byte miss ratio* (complementary metric to *hit* and *byte hit* ratio) to measure the "goodness" or "closeness" of the original and synthetic traces.

If we then restrict ourselves to using the *Frequency-Size* profile $(\mathbf{fr}, \mathbf{s})$ to capture the trace, we could then assume that, in a trace containing R requests, these are uniformly distributed over the entire trace, and simulate the effect of such a uniform distribution on the miss ratios, as a function of the amount of RAM available to the web server. More precisely, we generate a synthetic trace corresponding to a random permutation of the trace

$$(\underbrace{1, 1, \ldots, 1}_{\mathbf{fr}_1}, \underbrace{2, 2, \ldots, 2}_{\mathbf{fr}_2}, \ldots, \underbrace{F, F, \ldots, F}_{\mathbf{fr}_F}) \qquad (6)$$

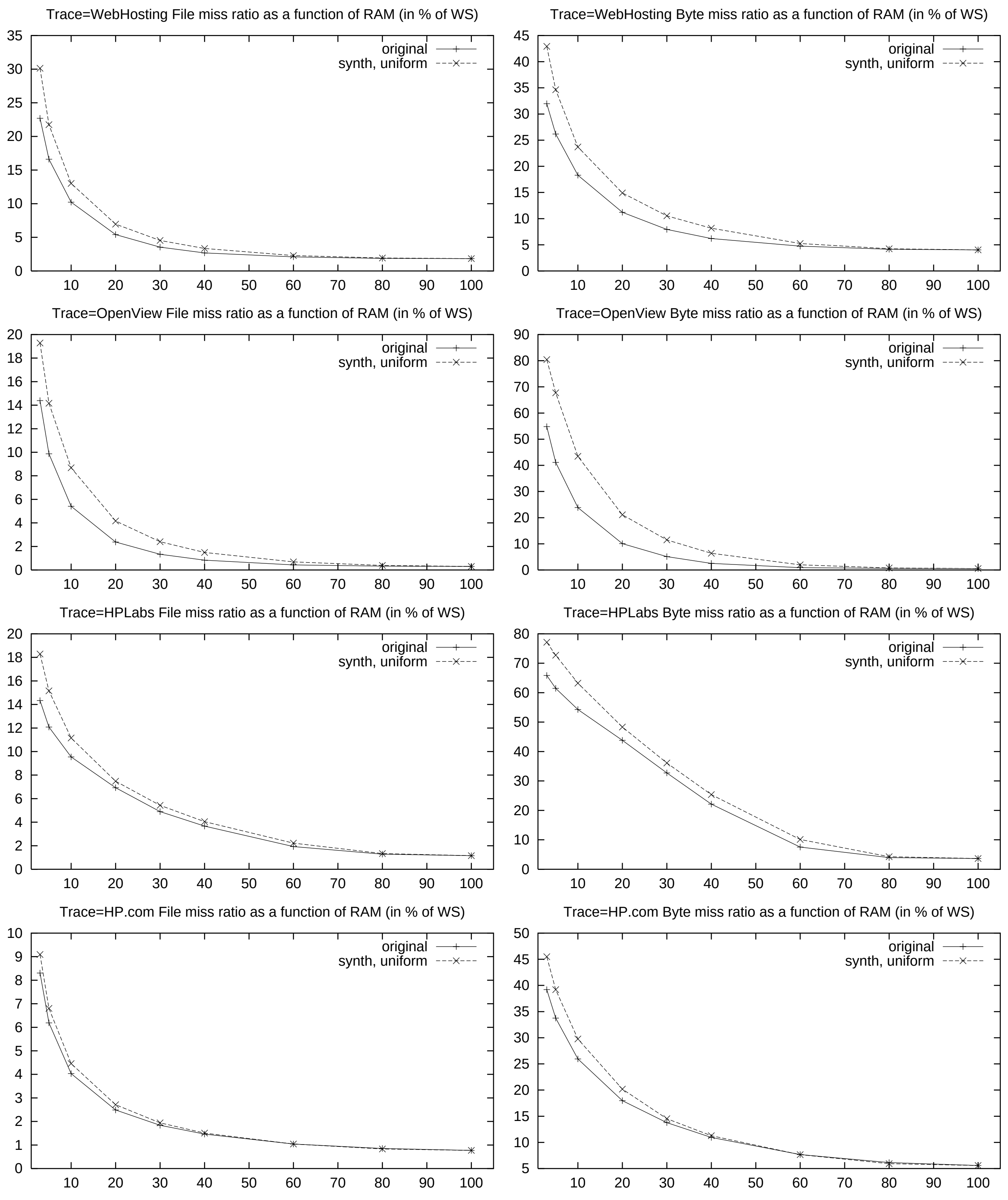

Figure 6: File and byte miss ratios (original vs. synthetic uniform traces).

and feed it to an analysis program that computes $M(x)$, the miss ratio when the web server has a RAM of size x available to store files.

Since the four traces we consider vary in their overall working sets, we choose to specify x as *a percentage of the working set*, in order to *normalize* the curves. In our study, we use 3%, 5%, 10%, 20%, 30%, 40%, 60%, 80%, and 100% of the working set. Note that, with a RAM equal to 100% of the working set, the only misses are the F "cold misses" required to bring the files from disk into memory for the first time, since all subsequent requests for a file are guaranteed to be "hits". When smaller amount of RAM are instead available, "warm misses" also become a factor.

Figure 6 illustrates the misses (cold plus warm) computed from the original traces and from the synthetic traces under the uniform assumption.

In all cases, the *uniform assumption* results in a *pessimistic assessment* of both the file and the 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 must of course appear "close" to each other in the trace, requests for a given file, then, must also tend to cluster together due to short-term correlation (for example, due to deadlines, swings in user interest, and so on). This important distinction between *long-term popularity* and *short-term temporal correlations* has also been observed recently in [6], indeed we have adopted their terminology for these two aspects of temporal locality.

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, which was introduced in [8] and was later adopted in [1] for the study of temporal locality in web traces. To define this notion, 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 simply 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). In particular, $d_1 = \infty$ since the first request cannot be for a file in the stack, as the stack is empty.

The values of the stack distances $(d_1, d_2, \ldots, d_R)$ clearly affect the performance of the machine serving the requests, as they affect the miss ratios. For example, if the stack distances are all quite small (except for the ones equal to ∞), this means that files are nicely clustered over the trace, and most requests will not require a disk access. Indeed, the ideal trace shown in (6) will only cause cold misses, as long as the RAM is large enough to contain the largest file in the trace; we could say that this trace has perfect short-term correlation.

Thus, we now focus on the definition of a web request profile that can capture, in addition to the frequency-size distribution of the trace, also its stack distance behavior.

To test how well the request profile captures the real traces, 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 generation algorithm

The key idea in our synthetic generation algorithm is the use of a stack data structure to schedule the order in which the requests are generated. Figure 7 shows the pseudo-code for our algorithm. The main variables used by it are:

- The global variables describing the request profile: R and F (integers), $\mathbf{fr}$ (integer vector), parameters to describe the stack distance distribution (we experiment with several options for this).
- The local variable $\mathbf{o} = [\mathbf{o}_1, \ldots, \mathbf{o}_F]$ (integer vector), the outstanding requests for each file.
- An integer stack $\mathbf{t}$, stored as a vector of fixed dimension F. The bottom position of the stack is 1 and the top position, *top*, is initially F, but it is reduced as files are removed from the stack when all their outstanding requests have been generated. The stack is empty, and the generation process ends, when *top* reaches 0. The contents of $\mathbf{t}$ reflect the LRU stack corresponding to the stream of requests *in reverse order*, that is, the file f in $\mathbf{t}_{top}$ is always the next one to be generated, then it is pushed down on the stack a certain amount d determined

```
1.  procedure $SyntheticTrace(\mathbf{fr})$ is
2.    $InitializeStack(\mathbf{t})$;
3.    $top \leftarrow F$;                                   • initial stack height
4.    $\mathbf{o} \leftarrow \mathbf{fr}$;                  • number of outstanding requests for each file
5.    while $top > 0$                                       • still requests to generate
6.      $f \leftarrow \mathbf{t}_{top}$;                    • $f$ is the file to be emitted
7.      output a request for $f$;
8.      $\mathbf{o}_f \leftarrow \mathbf{o}_f - 1$;
9.      if $\mathbf{o}_f = 0$ then                          • eliminate $f$ from the stack
10.       $top \leftarrow top - 1$;
11.     else                                                • push down $f$ in the stack
12.       $d \leftarrow GenerateStackDistance(top, f)$;     • returns a value between 0 and $top - 1$ included
13.       for $i \leftarrow top - 1$ to $top - d$ do        • shift $\mathbf{t}_{top-1}$ to $\mathbf{t}_{top-d}$ one position up
14.         $\mathbf{t}_{i+1} \leftarrow \mathbf{t}_i$;
15.       end for;
16.       $\mathbf{t}_{top-d} \leftarrow f$;
17.     end if;
18.   end while;
19. end procedure;
```

Figure 7: Our algorithm to generate a trace of length R.

probabilistically using information about the stack distance distribution, or it is removed from the stack if its number of outstanding requests $\mathbf{o}_f$ has reached 0. This way of "thinking in reverse" allows us to clearly define a policy for the order of file generation that requires only $O(1)$ operations to decide which file to emit[2].

The two functions used in our algorithm, *InitializeStack* and *GenerateStackDistance*, are critical to the correct probabilistic behavior of the algorithm. 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 and it should be carefully chosen, especially for short traces. After experimenting with several approaches, we found that initializing the stack probabilistically according to the information in $\mathbf{fr}$ works very well. More specifically, 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 Collecting stack distance information

The lognormal distribution [7] 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 a sequence of 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 [7]:

$$a = \frac{\sum_{i=1}^{n} \ln X_i}{n} \qquad b = \sqrt{\frac{\sum_{i=1}^{n} (\ln X_i - a)^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n} (\ln X_i)^2}{n} - a^2} \tag{7}$$

[2]The action of pushing down file f by d positions in a stack implemented as an array still requires $O(d)$ steps, of course, but most pushes are for small values of d in practice, due to temporal locality.

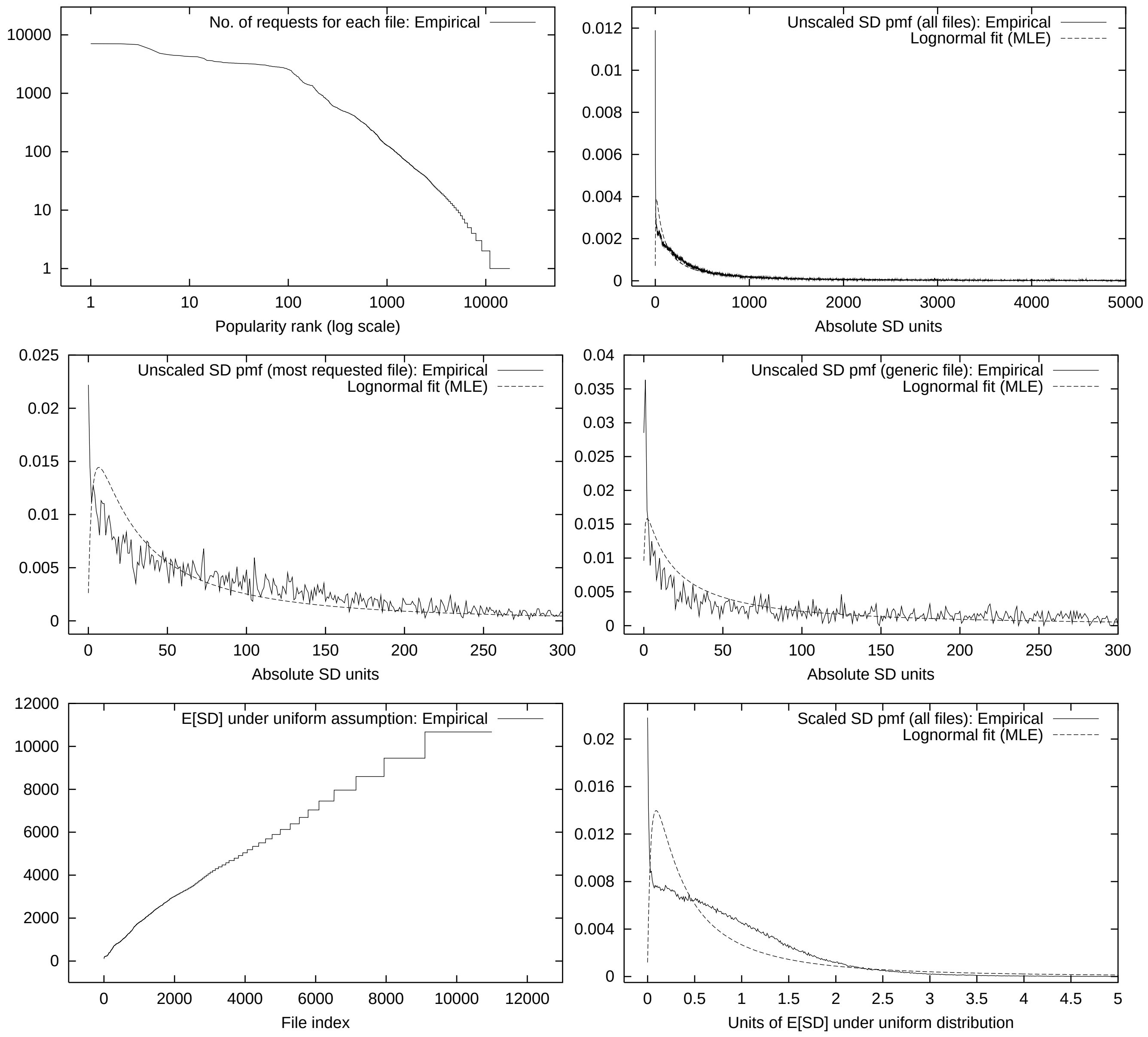

Figure 8: Unscaled and scaled stack distance and supporting plots (WebHosting trace).

In our first experiment to capture the stack distance information and parametrize the synthetic generator, the $R - F$ stack distances in $(d_1, d_2, \ldots, d_R)$ not having value ∞ are the observations we use to obtain a^u and b^u, the parameters of the lognormal distribution according to (7). The "u" stand for "unscaled", as it will be clear shortly. Then, in the synthetic trace generation, $GenerateStackDistance(top, f)$ simply 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 would cause to push f below the bottom of the stack, i.e., if $d \geq top$, this value 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 Figure 9. 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 Figure 8 (for the WebHosting 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 observed ignoring the identity of the file they refer to, the resulting lognormal distribution has a very 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 simply observe the stack distances experienced by each individual file separately. For example, the two plots in the middle of Figure 8 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 then consider two different ways to untie file popularity from the short-term correlation captured by the pmfs for the stack distances of individual files. With the most "expensive" approach we collect the distribution of the stack distances experienced by each file individually. More precisely, for each file 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, and the first one, d_1^f, is of course ∞. Then, we apply (7) for each file f, obtaining two vectors $\mathbf{a} = [\mathbf{a}_1, \ldots, \mathbf{a}_F]$ and $\mathbf{b} = [\mathbf{b}_1, \ldots, \mathbf{b}_F]$. Then, in the synthetic generation algorithm, *GenerateStackDistance*(top, f) simply returns the rounded value of a random deviate sampled from the distribution Lognormal$(\mathbf{a}_f, \mathbf{b}_f)$. Of course, again, values beyond the current stack height simply cause f to be pushed to the bottom.

A less expensive, but intellectually very 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

$$\mathbf{d}_f = \sum_{g \neq f} \frac{\mathbf{fr}_g}{\mathbf{fr}_g + \mathbf{fr}_f - 1},$$

a quantity that can be obtained exclusively form the vector $\mathbf{fr}$. For an explanation of this expression, one can consider the *Coupon Collecting Problem* discussed in Ross's textbook [9, p. 261-3]. The fraction in the summation represents the probability that, if we focus on a particular occurrence of f in the trace and start searching forward, we will find an occurrence of g before finding a second occurrence of f: this is the same as the probability that g is above f on the stack when this second occurrence of f on the trace is encountered. We use "$\mathbf{fr}_f - 1$" in the denominator because, unlike the rates in Ross's book, we use predefined numbers $\mathbf{fr}_f$ and $\mathbf{fr}_g$ of occurrences, hence we need to account for the current occurrence of file f from where we start searching.

With this second approach, we then define the *scaled stack distance* by dividing the actual stack distance d experienced by file f by its expected value in the uniform case: $d^{scaled} = d/\mathbf{d}_f$. Intuitively, an observed stack distance d_f of 12 for a very frequent file f whose expected stack distance under the uniform assumption is $\mathbf{d}_f = 10$ is actually "longer" than an observed stack distance d_g 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 (7). The resulting empirical scaled distance pmf (for the WebHosting trace) and its lognormal fit are shown at the bottom-right of Figure 8. On the left, we show instead the values of $\mathbf{d}_f$ for the file who are not onetimers, in decreasing popularity order. Thus, in the synthetic trace generation, *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 $\mathbf{d}_f$ (again, truncating values larger than $top - 1$).

The results for the miss ratios corresponding to these two approaches, individual stack distance (ISD) and scaled stack distance (SSD), are shown in Figure 9. Both of them result in a closer match of the original trace than that provided by the synthetic trace under the uniform assumption. Indeed, it is interesting to note that the simpler (in terms of size of the data required to store it) 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

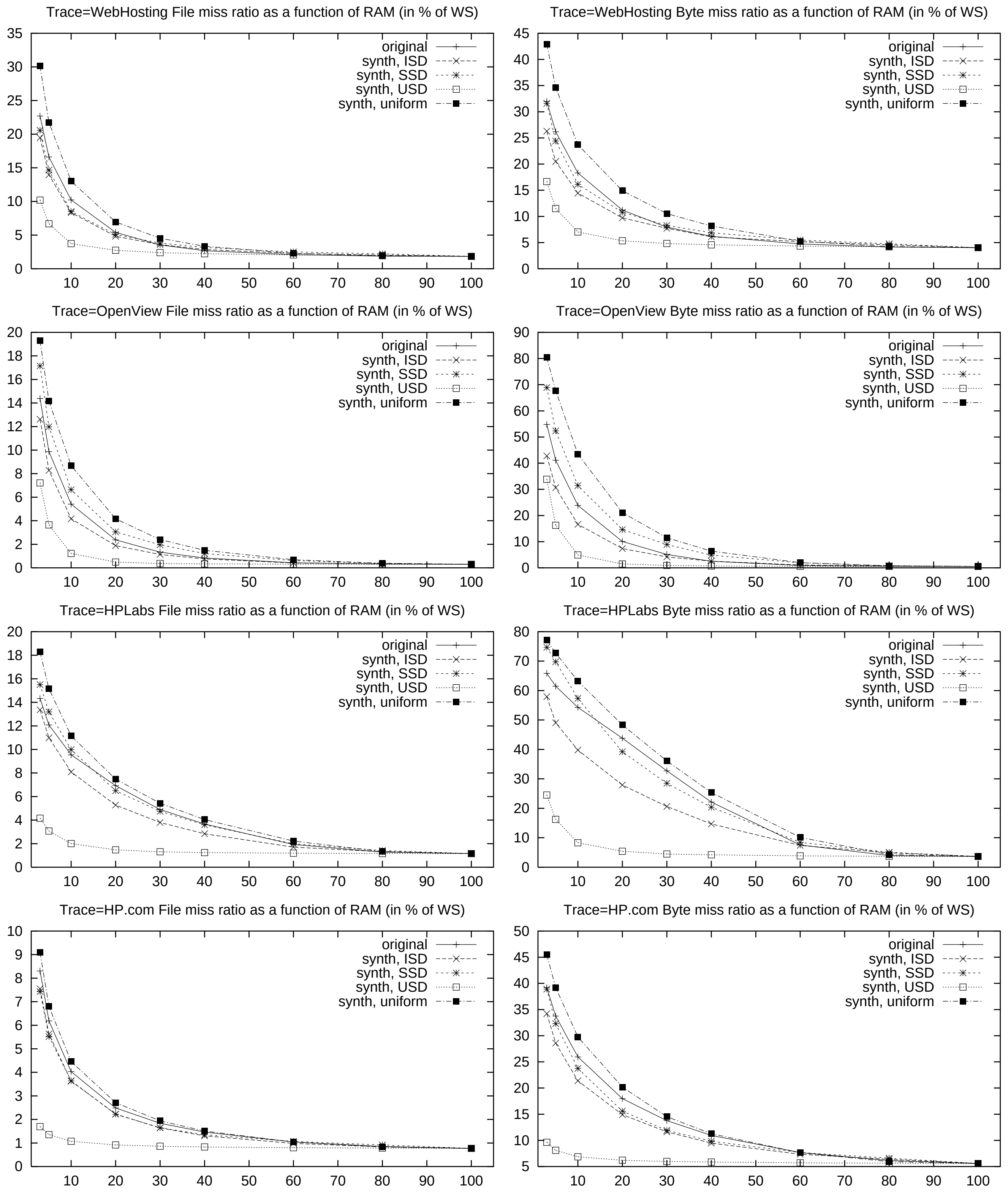

Figure 9: File and byte miss ratios (original vs. synthetic traces).

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.

6 Conclusion

In this paper, we analyze *static* and *temporal* locality in web server workloads.

The *static locality* characterization is based on the frequency-size profile of the files from the web server log: we rank files through their frequency (number of times they are requested in a trace) and their size (which is critical in determining correct cache behavior and server memory requirements).

The *temporal locality* of references implies that recently accessed documents are more likely to be referenced in the near future. 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.*

In this paper, 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 be 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 to us his library of random number generation and probability distribution function routines. This not only saved us much time, but it also ensured a high-quality implementation of this critical portion of the code.

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] Martin F. Arlitt and Carey L. Williamson. Web Server Workload Characterization: The Search for Invariants. In Proceedings of the ACM SIGMETRICS '96 Conference, Philadelphia, PA , May 1996. ACM.

[3] Barford, Paul; Bestavros, Azer; Bradley, Adam; Crovella, Mark. Changes in Web Client Access Patterns: Characteristics and Caching Implications, Technical Report, Boston University, TR-1998-023, 1998.

[4] Barford, Paul; Crovella, Mark. Generating Representative Web Workloads for Network and Server Performance Evaluation, Technical Report, Boston University, TR-1997-006, 1997.

[5] Bestavros, Azer; Carter, Robert; Crovella, Mark; Cunha, Carlos; Heddaya, Abdelsalam; Mirdad, Sulaiman. Application-Level Document Caching in the Internet, Technical Report, Boston University, TR-1998-023, 1995.

[6] 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. To appear.

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

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

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