On the Sensitivity of Web Proxy Cache Performance to Workload Characteristics

Mudashiru Busari Carey Williamson
Department of Computer Science
University of Saskatchewan
carey@cs.usask.ca

Abstract— **This paper describes the design and use of a synthetic Web proxy workload generator (ProWGen) to investigate the sensitivity of proxy cache replacement policies to selected Web workload characteristics. Trace-driven simulations with synthetic workloads from ProWGen show the relative sensitivity of three popular cache replacement algorithms – LRU, LFU-Aging, and GD-Size – to Zipf slope, temporal locality, and correlation (if any) between file size and popularity, and the relative insensitivity of these algorithms to one-timers and heavy tail index. Performance differences between the three policies are also highlighted.**

I. Introduction

The continuing growth of the Web can lead to overloaded Web servers and congestion on the Internet backbone, with the end result being an increase in user-perceived latency for retrieving Web documents.

Caching is one means of improving the performance and scalability of the Web. Caching can be applied at several locations: at the client [13], [19], [26], [38], at the server [7], [8], [13], and within the network [1], [2], [5], [9], [10], [13], [14], [18], [21], [35], [36], [38], [41]. The latter approach involves caching proxies [1], [18], [41], which are the focus of this paper.

Caching proxies have gained widespread popularity on the Internet. By keeping local copies of documents requested by Web clients, and using them to satisfy future requests to the same documents, caching proxies can reduce the amount of traffic flowing between Web clients and Web servers. As a result, Web servers see a reduced load, unnecessarily duplicated traffic through the Internet is reduced, and the delays perceived by users are reduced.

Web proxy workload characteristics and proxy caching performance have been well-studied in the literature. Some common characteristics identified in proxy workloads are: (1) the document popularity often follows Zipf's law [5], [30], [31], [39]; (2) the file size distribution for Web documents is heavy-tailed [1], [5], [30], [31]; (3) many (e.g., 50-70%) of the documents are referenced only once [1], [30], [31], [38]; and (4) temporal locality exists in the Web proxy reference stream [20], [28], [29], [30], [32], [39].

These characteristics pose a challenge for Web proxy cache design. Some authors argue that the limits of cache performance have already been reached, and that new techniques are required to improve performance [25], [34], [40].

This paper addresses the following two research questions:

- In a single-level proxy cache, how sensitive is Web proxy caching performance to certain workload characteristics (e.g., one-timers, Zipf slope, heavy-tailed file size distribution, temporal locality)?
- How does the degree of sensitivity change depending on the cache replacement policy?

Unfortunately, these questions are not easy to answer. For instance, investigating how a particular workload characteristic (e.g., Zipf slope) affects different replacement policies requires empirical workloads that differ only in that characteristic. Such workloads are obviously difficult to obtain.

Therefore, to answer the research questions above, we developed a synthetic Web proxy workload generator that offers the required flexibility. The workload generation tool is then used in a sensitivity study of proxy cache replacement policies.

The trace-driven simulation results show that the replacement policies evaluated are sensitive to Zipf slope, temporal locality, and correlation between file size and popularity, and relatively insensitive to one-timers and heavy tail index. These results provide insight into the design of Web proxy caching policies and Web proxy caching hierarchies [17].

The remaining sections of this paper are organized as follows. Section II provides a brief discussion of related work, and clarifies the contributions of this paper. Section III describes the proxy workload generation tool, and Section IV discusses the validation of the tool. Section V presents the experimental setup for the study of single-level caches, while the simulation results are presented in Section VI. Section VII concludes the paper.

II. Background and Related Work

Workload synthesis allows systems to be studied in controlled ways. This helps in management and capacity planning.

There are several workload generation tools developed to study Web servers and Web proxies. SPECweb99 [42], WebBench [44], and SURGE [11] are among workload generation tools designed to exercise Web servers by repeatedly sending requests (e.g., HTTP requests) from machines configured as clients to the intended server machine under test. Other synthesis tools, such as [33], [31], [45], rely on empirical workloads to generate synthetic ones.

The issue that motivates our proxy workload generation (ProWGen) tool makes it different from previous tools. The goal is to examine the sensitivity of proxy caching policies to certain workload characteristics. Therefore, ProWGen incorporates only those characteristics deemed relevant to caching. In addition, ProWGen models only the aggregate client workloads seen by a typical proxy server, rather than individual clients.

Several studies have focused on the importance of isolated workload characteristics on proxy cache performance. For example, Roadknight *et al.* [39] remarked on the dramatic impact of small changes in Zipf exponent on the popularity of the most popular files and the number of files in the proxy's cache. Mahanti *et al.* [30], [32] investigated the importance of temporal locality on Web proxy cache performance and concluded that temporal locality is an important factor in proxy cache performance. Several other researchers have commented on the importance of other workload characteristics and their impact on proxy server performance (e.g., correlation between file size and popularity [15], one-timers [31], heavy-tail property [23]).

Our work differs from these studies in that a synthetic workload generation tool (ProWGen) is used to generate proxy workloads that differ in one chosen characteristic at a time. These workloads, which would have been impossible to obtain empirically, are then methodically used to investigate the sensitivity of cache replacement policies (recency-based, frequency-based, and size-based) to each workload characteristic, using trace-driven simulations.

III. Description of ProWGen

There are two common approaches to synthesizing Web proxy workloads [11]. The *trace-based* approach uses an empirical trace and either samples it or permutes the orderings of the requests to generate a new workload. This approach is straightforward to implement but has limited flexibility, since the workload is inherently tied to a known system. The *analytical* approach uses mathematical models for the workload characteristics of interest, and uses random number generation to produce workloads that statistically conform to these models. This approach, though challenging, offers a lot of flexibility.

ProWGen uses the analytic approach. The workload generator incorporates five selected workload characteristics, which are deemed relevant to caching performance. These workload characteristics are: *one-time referencing, file popularity, file size distribution, correlation between file size and popularity*, and *temporal locality*. The following subsections describe each of these characteristics and how they are modeled in ProWGen.

A. Modeling One-time Referencing

Studies of Web server and Web proxy workloads have shown that many documents requested from a server or a proxy are requested only once, regardless of the duration of the access log studied [1], [8], [31]. These documents are referred to as "one-timers" in the literature. Clearly, there is no benefit to caching one-timer documents, since they are never accessed again. In fact, caching algorithms need to discriminate against such documents so that they do not clutter the cache and reduce its effectiveness [8].

The study of one-timers is important because of their prevalence in Web workloads. Arlitt and Williamson [8] report that 15-40% of the unique files accessed from a Web server are accessed only once. The situation is even worse in Web proxy access logs, where one-timers can account for 50-70% of the documents [1], [31]. Modeling the one-time referencing characteristic is thus important when generating workloads for evaluating different caching algorithms. Fortunately, the modeling of one-timers is relatively straightforward.

In ProWGen, the user specifies the percentage of one-timers desired in the synthetic workload (as a percentage of the distinct files). Once the value of this parameter is specified, the number of one-timers in the workload is determined, and their reference counts are fixed at one. The reference counts for the remaining distinct (non-one-timer) files are determined from a Zipf-like distribution, as explained in the next section.

B. Modeling File Popularity

One common characteristic in Web workloads is the highly uneven distribution of references to files [5], [30], [39]. In many cases, Zipf's law [37] has been applied to model file popularity [3], [4], [15], [24].

Zipf's law expresses a power-law relationship between the popularity P of an item (i.e., its frequency of occurrence) and its rank r (i.e., relative rank among the referenced items, based on frequency of occurrence). This relationship is of the form $P = c/r^{\beta}$, where c is a constant, and β is often close to 1. For example, Zipf's law arises in the frequency of occurrence of English words [15]; when the number of occurrences is plotted versus the rank, the result is a power-law function with exponent close to 1.

In the Web context, a similar referencing behaviour is observed [5], [31], [39]. Some researchers have found that the value of the exponent β is close to 1 [4], [24], precisely following Zipf's law. Others [3], [15], [31] have found that the value of β is less than 1, and that the distribution can be described only as "Zipf-like", with the value of β varying from trace to trace. This behaviour typically results in a straight line of (negative) slope β on a log-log plot of P versus r. The linear fit is usually good for the main body of the distribution, though it may deviate slightly at both the most popular end (due to "hot" documents) and the least popular end (due to one-timers) [31].

The synthetic workloads produced by ProWGen use the Zipf-like distribution for file popularity. After fixing the number of one-timers as explained in Section III-A, the reference counts of the remaining distinct files are determined from a Zipf-like distribution. The term "Zipf-like" is used here because the value of β supplied to the workload generator determines whether the generated reference stream follows the Zipf distribution perfectly ($\beta = 1$) or approximately ($0 < \beta < 1$).

C. Modeling File Size Distribution

Workload characterization studies [1], [6], [8], [27], [30] have shown that the file size distribution for Web transfers is heavy-tailed. A heavy-tailed distribution implies that relatively few large files account for a significant percentage of the data volume (in bytes) transferred to Web clients. This heavy-tail property contributes to the self-similarity observed in WWW traffic [23].

Clearly, the distribution of file sizes affects the design and performance of caching strategies. Caching only small files can reduce the number of requests to originating servers. This can result in a high document hit ratio, but a low byte hit ratio. On the other hand, caching large files can result in a higher byte hit ratio at the expense of document hit ratio (since many small documents may be forced out of the cache).

For effective evaluation of cache management strategies, the heavy-tailed file size distribution must be incorporated into synthetic workloads. Furthermore, the "heaviness" of the tail needs to be adjustable, to evaluate its impact on caching performance.

Incorporating the heavy tail model into ProWGen begins with partitioning of the distinct files into two sets: those in the body and those in the tail. The body of the file size distribution is modeled with a lognormal distribution, and the tail is modeled with a Pareto distribution. This approach follows that used by other researchers [6], [11], [22], and is thus not described at length here.

Two additional steps are added to the file size modeling process in ProWGen. One (optional) step bounds the maximum file size generated from the Pareto distribution, if so desired. This bound (e.g., 50 MB) can limit the impacts of extremely large files in the workload generated, making it easier to match the workload characteristics of an empirical trace. The other step carefully joins the two statistical distributions (lognormal body and Pareto tail) so that there is no discontinuity at the boundary between the two distributions. Interested readers are referred to [16] for the details of the file size modeling process.

D. Modeling Correlation (File Size and Popularity)

Proxy workload studies show that many of the files transferred on the Web are small [1], [8], [15], [30], [31]. A natural question that arises is whether there is any statistical correlation between the frequency of access to a file and its size. Some studies [15], [30] have shown that there is little or no correlation between the frequency of access to a file and its size, though the issue is still debated [4], [8], [15].

For flexible modeling, ProWGen allows generation of workloads that exhibit positive, negative, or zero correlation between file size and file popularity. Positive correlation means that large files are more likely to be accessed, and negative correlation means that small files are more likely to be accessed. While zero correlation is arguably the closest approximation of reality, the other workload generation options (positive and negative correlations) allow exploration of the sensitivity of caching algorithms to this workload characteristic.

Modeling size-popularity correlation in the workload generator is accomplished in three stages. First, a set of file popularities is generated, as described in Section III-B. Second, the approach in Section III-C is used to generate a set of file sizes. Third, a mapping technique is used to introduce either positive, negative, or zero correlation between file popularities and file sizes. The mapping technique relies on computing the cumulative distribution functions (CDF) for the file popularities and the file sizes, and then using uniform U(0,1) random number generation to generate file popularities and sizes from the CDF values, using the standard inverse mapping technique. For instance, to introduce strong positive correlation, a random number r ($0 \leq r < 1$) is used to access *both* CDFs, interpolating when necessary. For negative correlation, if r is used for the first mapping, then $1 - r$ is used for the other mapping. For zero correlation, independent random numbers r and s are generated for the two mappings. Intermediate degrees of correlation can be achieved by probabilistically choosing between these three mapping strategies on a per-file basis.

E. Modeling Temporal Locality

Temporal locality refers to the tendency for Web documents referenced in the recent past to be referenced in the near future. Clearly, the presence (and strength) of temporal locality in the synthetic workload can have a dramatic effect on caching performance [20], [28], [32].

The approach used in ProWGen to model temporal locality is based on the finite size Least-Recently-Used (LRU) stack model [4], [30], [32], [43]. The LRU stack maintains an ordered list of documents based on recency of access. The LRU stack is updated dynamically as a reference is processed. In some cases, this update involves adding a new item to the top of the stack, pushing others down; in other cases, it involves extracting an existing item from the midst of the stack and moving it to the top again, pushing other items down as necessary.

The important aspect of an LRU stack is that each position in the stack has an associated probability of referencing it. Note that the probabilities are associated with the stack positions, and not the documents. In ProWGen, the stack position probabilities are determined based on desired document popularities. For example, suppose that the popularities of the n distinct files in the workload are represented by $D = \{x_1, x_2, \ldots, x_n\}$, where $x_1 \geq x_2 \geq \ldots \geq x_n$. Then the (stationary) probability a_i for each distinct file i is computed using $a_i = \frac{x_i}{\sum_{j=1}^{n} x_j} \quad \text{for } i = 1, 2, \ldots, n$. Assuming a finite stack of size m such that $m \leq n$, the cumulative probability y_i of referencing stack position i is computed as $y_i = \sum_{j=1}^{i} a_j$.

Two different temporal locality models follow naturally from this formulation. In the *static* model, the probabilities on the LRU stack positions are assigned permanently from the beginning of the reference stream generation. In the *dynamic* model, the probabilities are computed dynamically depending on the files currently occupying each stack position. Note that the static model introduces “homogeneous” temporal locality throughout the reference stream, while the dynamic approach introduces “heterogeneous” (i.e., document-specific) temporal locality, biased toward popular documents. These models both differ from the temporal locality models used by Mahanti [32].

The reference generation process then proceeds as follows. First, a U(0,1) random number x is generated. If the stack is empty, then a reference is generated for a file selected uniformly at random from the set of distinct files requiring further references, and the remaining reference count for the selected file is decremented by one. Otherwise, the stack is searched until a position i is found such that $x \leq y_i$. If position i exists on the stack, then a reference is generated for the file in that stack position, and that file is either moved to the top of the stack or removed from the stack, depending on whether that file requires more references or not. If there is no position i such that $x \leq y_i$, then a reference is generated for a file selected uniformly at random from the set of remaining distinct files not on the stack. The reference stream generation ends when all references for the distinct files have been generated.

Each reference in the generated workload is a two-tuple consisting of a *file id* followed by a *file size*. The temporal locality model merely controls the relative ordering of these references in the generated workload.

F. Summary of ProWGen

ProWGen is a software tool to generate synthetic Web proxy workloads of arbitrary length. The dozen or so input parameters for ProWGen provide control over five key workload characteristics, namely one-time referencing, file popularity, file size distribution, correlation between file size and popularity, and temporal locality. Each modeled document in the workload is independent of other documents.

Each reference in the generated workload consists of a file id and a file size, with the re-referencing of files determined by the file popularity and temporal locality models. Note that ProWGen in its current form does not model individual client behaviours; rather it models the aggregate workload as generated from many clients, and seen at a Web proxy. Furthermore, ProWGen does not model time-of-day effects (i.e., request arrival times), document types, dynamic content, or document modifications. Post-processing scripts can be used to add such information to a ProWGen workload, if desired, or indeed to convert a ProWGen workload into a standardized Web proxy access log format for processing by other tools.

Parameterization of ProWGen can be used to create a wide range of characteristics in the synthetic workloads produced, or to produce characteristics similar to those in an empirically observed Web proxy workload. Validation of ProWGen against an empirical workload is described in the next section.

IV. Validation of ProWGen

The purpose of validation is to ensure that the synthetic workloads produced by ProWGen have realistic quantitative and qualitative characteristics. The validation process establishes confidence in the workload generator and in the statistical properties of the workloads produced. This task is accomplished in two stages: (1) verifying that a sample workload produced by ProWGen has the desired statistical characteristics; and (2) calibrating ProWGen to produce workloads that mimic an empirical workload from a university-level proxy server [32].

Table I and Table II provide statistical summaries of the empirical trace and a sample synthetic trace used for validation. The statistical characteristics of the generated trace match closely with those of the empirical trace used.

A “graphical validation” of the synthetic workload is shown in Figure 1. Figures 1(a) and (b) show the file popularity results for the empirical trace and the synthetic trace, respectively. Figure 1(c) shows the cumulative distributions for file sizes and bytes transferred, while Figure 1(d) shows the tail behaviour using a log-log complementary distribution (LLCD) plot. In the latter two graphs, the results for the empirical trace are shown using solid lines, while the results for the synthetric trace are shown using dashed lines. There is thus strong evidence that workloads generated using ProWGen can match both the body and the tail of the empirical file size distribution, in addition to the Zipf-like referencing behaviour.

The next step of the validation process compares the synthetic and empirical traces in terms of caching performance [16]. These experiments (not shown here) establish that the dynamic LRU stack temporal locality model yields results closer to the empirical trace results than the static LRU stack model (see Figure 7(a), for example). The dynamic LRU stack model is thus used in the experiments.

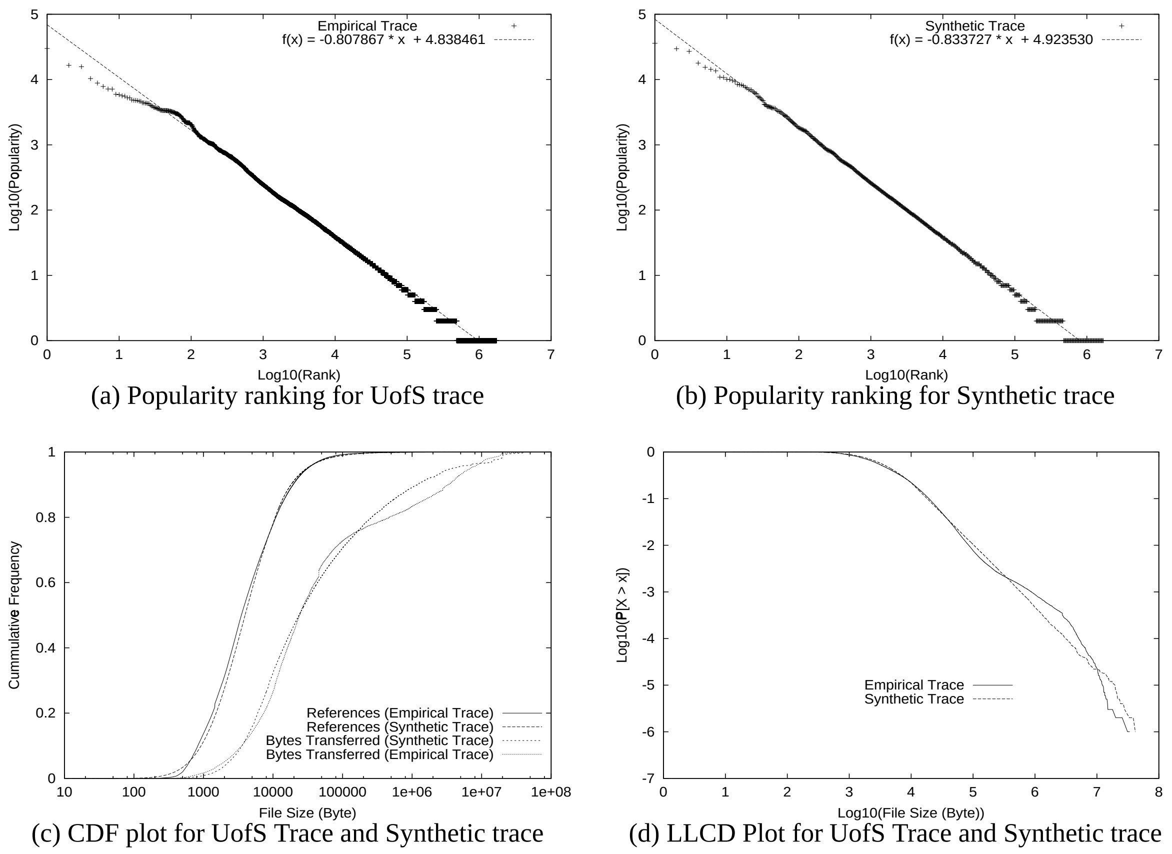
TABLE I
Characteristics of Empirical Trace (UofS Proxy)

Item	Value
Total requests	5,000,000
Unique documents (% of requests)	34%
One-timers (% of unique documents)	72%
Total Gbytes of unique documents	19
Smallest file size (bytes)	0
Largest file size (bytes)	53,857,877
Mean file size (bytes)	11,740
Median file size (bytes)	3,504
Mean file size (body only)	3256
Standard deviation of file size (body only)	2441
Zipf Slope	-0.808
R^2	0.992
Tail index	-1.323
R^2	0.980
Beginning of the tail (k) in bytes	10,000

TABLE II
Characteristics of Synthetic Trace (UofS Proxy)

Item	Value
Total requests	4,965,779
Unique documents	1,700,000
Unique documents (% of requests)	34%
One-timers	1,223,719
One-timers (% of unique documents)	71%
Total Gbytes of unique documents	17
Smallest file size (bytes)	13
Largest file size (bytes)	42,975,450
Mean file size (bytes)	11,157
Median file size (bytes)	3,962
Correlation (size and popularity)	-0.005
Zipf Slope	-0.834
R^2	0.998
Tail index	-1.326
R^2	0.998

One interesting observation from the validation experiments is that the *byte hit ratio* performance metric for a Web proxy cache is extremely sensitive to the number of files that fall into the category of being both very large and very popular. Thus two workloads generated from the same set of ProWGen parameters but with independent random number streams can differ significantly in the byte hit ratio results, even for lengthy traces. The document hit ratio results show much less variability. To ameliorate this problem with byte hit ratio results, an optional feature was added to ProWGen to limit the maximum popularity of extremely large files, without seriously compromising the size-popularity correlation model [16]. A good discussion of the mathematical relationships between hit ratio, byte hit ratio, and size-popularity correlation is provided in [12].



(a) Popularity ranking for UofS trace (b) Popularity ranking for Synthetic trace

(c) CDF plot for UofS Trace and Synthetic trace (d) LLCD Plot for UofS Trace and Synthetic trace

Fig. 1. Comparison of Workload Characteristics for Empirical Trace and Synthetic Trace

V. Sensitivity Analysis: Single-Level Caches

This section describes the experimental methodology used to investigate the sensitivity of single-level[1] proxy cache performance to certain workload characteristics (e.g., one-timers, Zipf slope, heavy-tailed file sizes, correlation, temporal locality). Simulation results are presented in Section VI.

A. Simulation Model

The simulation experiments model a single Web proxy server, as shown in Figure 2. In the trace-driven simulation, all requests coming from the clients (i.e., the aggregate workload generated by ProWGen) are directed to the proxy server. When the proxy receives a request from a client, it checks its cache to see if it has a copy of the requested file. If there is a copy in its cache, the proxy records a cache hit and returns the file to the user. Otherwise, the proxy records a cache miss, obtains a copy of the file from the (simulated) Web servers, stores a copy in its cache for future use, and returns a copy to the requesting client. If the cache is full when a file needs to be stored, then a replacement decision is triggered to decide which file (or files) to remove. Note that there are no coherence misses, since file modification events are not modeled in ProWGen; only cold misses (first reference to a file) and capacity misses (a previously referenced file which has been removed from the cache because of capacity constraints and the replacement policy used) can occur. Each workload used in the experiments has approximately 5 million requests, of which the first 10% are used for warm-up, prior to collecting cache statistics [16]. Only static documents are modeled.

[1]Simulation results for multi-level caching hierarchies are presented in a separate paper [17].

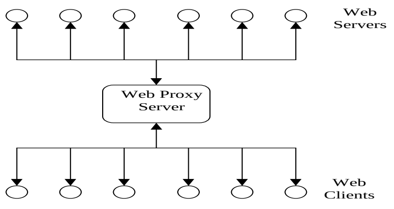


Fig. 2. Single Web Proxy Server Simulation Model

B. Factors and Levels

The sensitivity experiments employ a one-factor-at-a-time methodology, with three main factors: cache size, cache replacement policy, and workload characteristic.

The simulated cache sizes range from 1 MB to 16 GB. The latter value represents an infinite cache size (i.e., enough cache space to store all requested files without any replacements) for the workloads considered, so that the performance of a smaller cache size could be compared to that of an infinite cache.

Three different cache replacement policies are considered, namely Least-Recently-Used (LRU), Least-Frequently-Used-with-Aging (LFU-Aging), and Greedy-Dual-Size (GD-Size), representing a broad range of possible replacement policies [7], [19], [32], [46]. LRU is a recency-based policy that orders files based on recency of use only. When a replacement decision is required, LRU removes from the cache the file that has not been referenced for the longest period of time (i.e., Least Recently Used). LFU-Aging is a frequency-based policy that tries to keep popular documents in the cache. When space is needed for a new file in the cache, LFU-Aging removes files with the lowest reference count (i.e., Least Frequently Used). "Aging" is used periodically to reduce the reference counts of cached documents, so that formerly popular documents do not clutter the cache long after their popularity diminishes. GD-Size is a size-based policy, which tries to keep "small" files in the cache. This policy, proposed by Cao and Irani [19], associates a value [2] H with each file brought into the cache. Whenever replacement is needed, the file with the lowest H value, say H_{min}, is selected for removal, and the H value of all other files in the cache are reduced by H_{min}. When a cache hit occurs, the H value of the file is restored to its original value.

Five different Web workload characteristics are studied, namely one-time referencing, Zipf slope, heavy tail index, correlation between file size and popularity, and temporal locality. Each characteristic is represented by a controllable parameter in ProWGen, so that each workload characteristic can be controlled separately from the other characteristics. For each workload characteristic under study, ProWGen is used to synthesize workloads differing in only that characteristic. Unless stated otherwise, the workload characteristics in Table II are used as the default settings for all experiments.

C. Performance Metrics

The two performance metrics used in the sensitivity study are *document hit ratio* and *byte hit ratio*. The document hit ratio is the number of requests satisfied by the proxy's cache divided by the total number of requests seen by the proxy. The byte hit ratio is the volume of data (in bytes) satisfied by the proxy's cache divided by the total volume of data requested from the proxy. Both metrics are required since Web documents can differ dramatically in size. In general, the higher the hit ratio and byte hit ratio are, the better a replacement policy is.

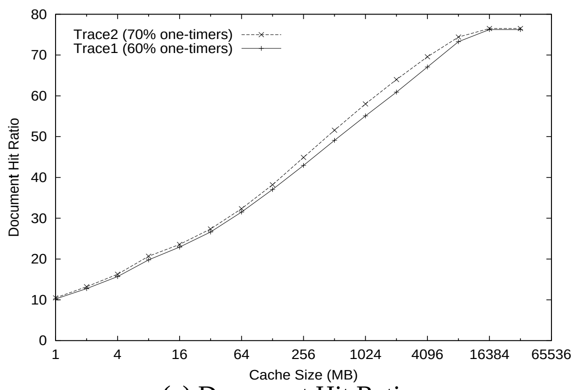
VI. Simulation Results

This section presents the results from the experiments. For space reasons, we focus primarily on the results for the LRU policy (as an example of a canonical replacement policy), and comment upon the differences (if any) observed with other replacement policies. Complete results are available in [16].

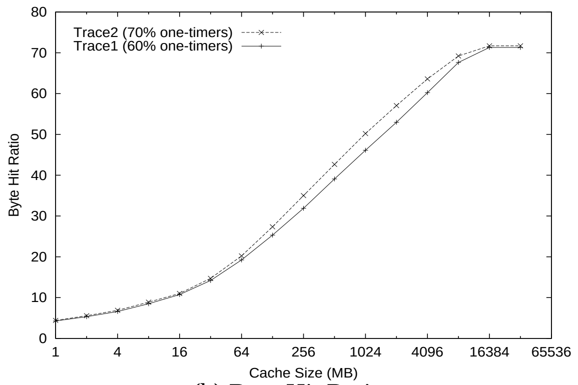
A. Sensitivity to One-timers

To study the effect of one-timers on cache replacement policies, two workloads, `Trace1` and `Trace2`, were produced using ProWGen, with 60% and 70% one-timers, respectively. The performance results for the LRU replacement policy are shown in Figure 3.

[2]In our work, we use $H = 1/s$, where s is the size of the file.



(a) Document Hit Ratio



(b) Byte Hit Ratio

Fig. 3. Sensitivity of LRU Policy to One-timers

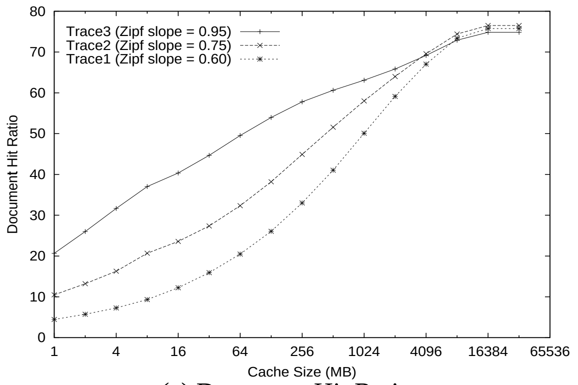
The results in Figure 3 show that the percentage of one-timers has little impact on the caching performance results. The difference in document hit ratios and byte hit ratios is negligible at small cache sizes, and modest (1-4%) at larger cache sizes. The results for LFU-Aging and GD-Size are similar [16].

Surprisingly, the caching performance is *better* for `Trace2` (70% one-timers) than for `Trace1` (60% one-timers). The reason for this behaviour is the higher concentration of references in `Trace2`. Because each trace (by construction) has the same number of references and the same number of unique documents, the traces differ only in how the references are distributed across the documents. The additional one-timers in `Trace2` mean that the non-one-timer documents in `Trace2` have (on average) more references per document, resulting in higher concentration of references and better caching performance. The same trend is observed as the percentage of one-timers is varied from 10% to 90% [16].

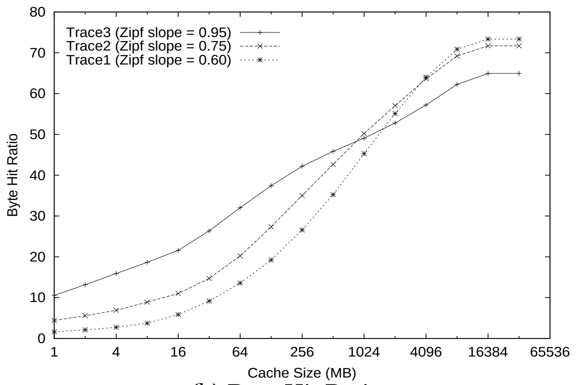
The same trend applies for all the replacement policies considered, though LFU-Aging shows the greatest sensitivity to the number of one-timers [16]. In general, all cache replacement algorithms studied (even Random and FIFO [16]) are relatively insensitive to the percentage of one-timers.

B. Sensitivity to Zipf Slope

Three synthetic workloads are used to study the impact of Zipf-like referencing, with Zipf slopes of 0.60, 0.75, and 0.95. The caching performance results for the LRU replacement pol-



(a) Document Hit Ratio



(b) Byte Hit Ratio

Fig. 4. Sensitivity of LRU Policy to Zipf Slope

icy on these workloads are shown in Figure 4.

The results show that as the exponent of the Zipf formula increases, both the document hit ratio and the byte hit ratio increase. The reason for the improved caching performance is obvious: an increase in concentration of references as the slope becomes steeper [16]. For example, 60% of the total references are accounted for by 3% of the busiest documents in `Trace3`, compared to 12% of the busiest documents for `Trace1`.

Figure 4 also shows that there is an interesting crossover effect in the byte hit ratio[3] graph (Figure 4(b)), such that a steeper Zipf slope leads to a worse byte hit ratio for large cache sizes. This is due to the sensitivity of the byte hit ratio metric to the number of large and popular files [16]. That is, the flatter Zipf slope tends to produce more files in this category, since references are in general spread across more of the documents (and hence across more of the large documents). As a result, the byte hit ratio increases more significantly for the workload with the flatter Zipf slope, at larger cache sizes.

In summary, an increase in the Zipf slope generally translates into better caching performance, except for the crossover effect for byte hit ratio at large cache sizes. These observations are consistent across the three replacement policies studied [16].

[3]The crossover is also weakly evident in the document hit ratio results.

C. Sensitivity to Tail Index

The experiments in this section illustrate the sensitivity of replacement policies to the Pareto heavy tail index. ProWGen is used to generate three workloads, with tail index values of 1.2, 1.3, and 1.4 to reflect "heavier", "medium", and "lighter" heavy-tailed file size distributions, respectively.

Figure 5 shows the results for the LRU and GD-Size replacement policies on these workloads. In terms of document hit ratios (Figures 5(a) and (b)), the tail index has little impact. The reasons for this are twofold: (1) the number of files affected by the tail index parameter is small; and (2) many of these files are so large (e.g., 10-42 MB) that they do not fit (or stay long) in the cache, especially at small cache sizes. As the cache size increases, the impact of the heavy tail index becomes more pronounced, with greater impact on byte hit ratio than on document hit ratio.

Among the replacement policies considered, the GD-Size policy shows the least sensitivity to the tail index parameter [16], in terms of document hit ratio (compare Figures 5(a) and (b)). This behaviour is not surprising, since the GD-Size policy tends to evict large files, while keeping small(er) documents in the cache. However, its bias against large files makes it more sensitive to the tail index parameter, in terms of byte hit ratio (compare Figures 5(c) and (d)).

In general, all replacement policies studied provide lower hit ratios and lower byte hit ratios when the heaviness of the tail increases, but the impact of the tail index is modest compared to that of the Zipf exponent (at least over the range of parameter values studied).

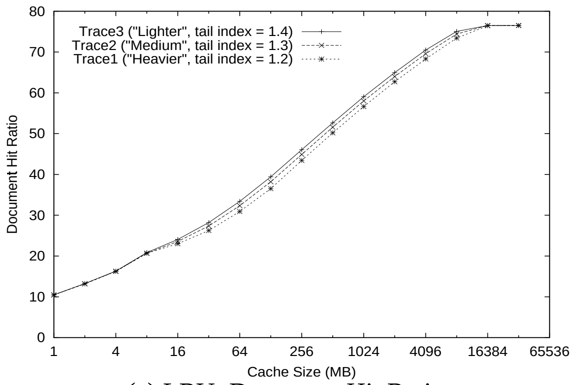
D. Sensitivity to Correlation (File Size and Popularity)

The experiment in this section studies the impact of correlation between file size and popularity on the different replacement policies. In particular, traces with strong positive and strong negative correlation are used, in addition to a trace with zero correlation. While zero correlation is arguably the most realistic, the other two workloads can establish the sensitivity of caching results to correlations, if they were present.

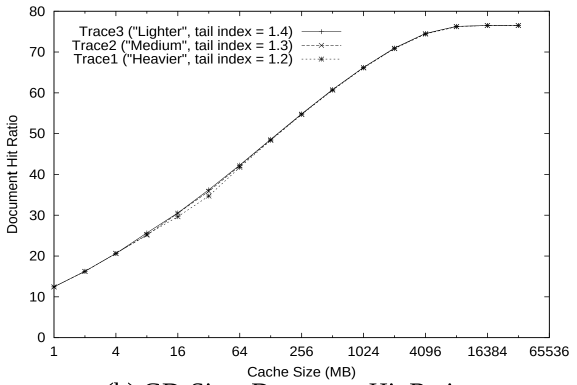
The performance results, again for the LRU policy, are shown in Figure 6. The results show the dramatic impact of the correlation property on the caching performance.

When there is negative correlation, such as in `Trace1`, small files are popular, while large files are not. The result is that the small files, many of which can fit in the cache, receive many re-references, producing a high document hit ratio. The penalty for this success is a poor byte hit ratio, since the large files (responsible for many of the bytes transferred) are rarely accommodated in the cache, except at large cache sizes.

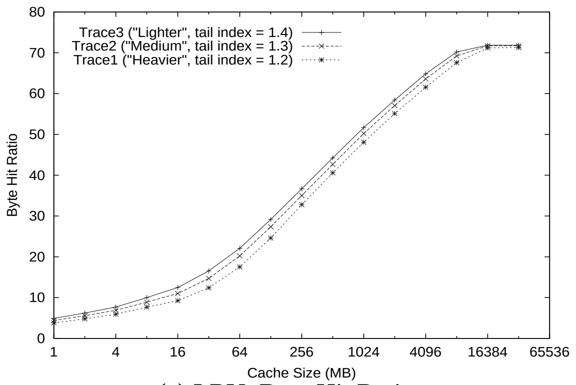
On the other hand, in workload `Trace3` with positive correlation, small files have low popularities and large files have high popularities. It follows that at small cache sizes (say, up to 256 MB), the observed document hit ratio is poor. Similarly, the byte hit ratio is poor (say, up to a cache size of 64 MB), because few of the large files with many references can be accommodated in the cache. The small files that do fit in the cache are not heavily re-referenced, while the large files with many re-references compete with each other for cache space. Beyond



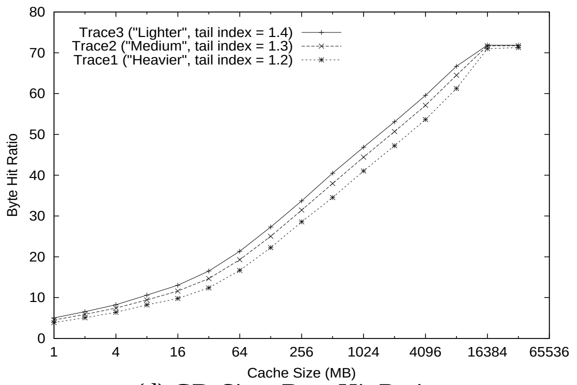
(a) LRU: Document Hit Ratio



(b) GD-Size: Document Hit Ratio



(c) LRU: Byte Hit Ratio



(d) GD-Size: Byte Hit Ratio

Fig. 5. Sensitivity of LRU and GD-Size Policies to Heavy Tail Index

some critical cache size, the document hit ratio and byte hit ratio increase sharply, with the byte hit ratio asymptotically approaching 100% (for `Trace3` only). Again, this reflects the sensitivity of the byte hit ratio metric to files that are both very large and very popular.

The results for zero correlation are (as expected) between the results for positive and negative correlations.

In summary, the presence of correlation between file size and popularity can have a dramatic impact on caching performance (at least for the extreme correlation values considered). The results for LFU-Aging and GD-Size are similar to those presented for LRU [16].

E. Sensitivity to Temporal Locality

The final experiment uses workloads with two different temporal locality models, to assess the impact on caching performance. The first workload (`Trace1`) uses the *static* LRU stack model, while the second (`Trace2`) uses the *dynamic* LRU stack model. By design, the two workloads differ only in the ordering of their references.

Figure 7 shows the performance of the LRU and the LFU-Aging cache replacement policies. A general observation from the figure is that all the policies provided higher document hit ratios and byte hit ratios for `Trace1`, which arguably has “better” (but not necessarily more realistic[4]) temporal locality.

Another observation from Figure 7 is that the LRU policy is more sensitive to the degree of temporal locality in the traces (note the wider vertical gap between the lines for `Trace1` and `Trace2`), while the LFU-Aging policy is less sensitive [16]. These observations are consistent with those reported by Mahanti *et al.* [32].

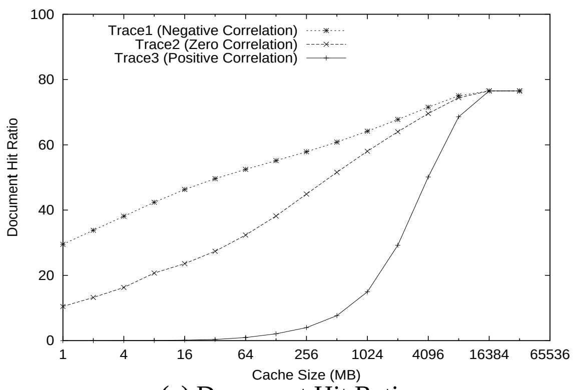
In summary, all three replacement policies are sensitive to temporal locality, with LRU being the most sensitive, followed by GD-Size and then LFU-Aging. The stronger the temporal locality, the better the caching performance. The sensitivity is similar for both document hit ratio and byte hit ratio.

VII. Summary and Conclusions

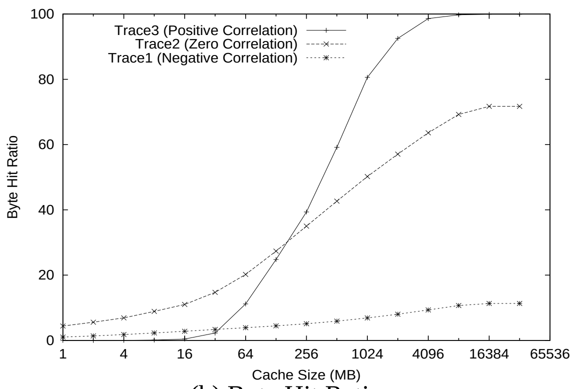
This paper describes the design of a Web proxy workload generator called ProWGen, and the use of synthetic workloads in a sensitivity study of single-level proxy caches. Validation of ProWGen shows that it models and captures the salient characteristics of empirical workloads. ProWGen provides fast workload generation, workload reproducibility, and low storage space requirements. By modeling workload characteristics with controllable parameters, ProWGen provides the flexibility required for synthesizing workloads used to study the effect of workload characteristics on cache replacement policies.

The simulation results show the sensitivity of cache replacement policies to Zipf slope, temporal locality, and correlation

[4]For reference purposes, the cache hit ratio for the empirical UofS trace is also shown in Figure 7(a).



(a) Document Hit Ratio



(b) Byte Hit Ratio

Fig. 6. Sensitivity of LRU Policy to Correlation Between File Size and Popularity

(if any) between file size and popularity, and the relative insensitivity to one-timers and heavy tail index. These results illustrate how workload characteristics affect cache performance for different replacement policies, and can provide insight into the design of new cache replacement policies.

Ongoing work [17] is using ProWGen for evaluating multi-level caching hierarchies. Preliminary results indicate modest performance advantages for using different cache replacement policies (and even size-based document partitioning strategies) across different levels of a caching hierarchy.

Acknowledgements

The authors thank Martin Arlitt, Rick Bunt, Anirban Mahanti, Prashant Shenoy, and the anonymous reviewers for their comments and feedback on an earlier version of this paper. The Web proxy workload traces used for the development, parameterization, and validation of the ProWGen tool came from an earlier CANARIE-funded project on Web proxy workload characterization [31]. Greg Oster provided admirable technical support for the collection, storage, processing, management, and analysis of these traces.

Financial support for this research was provided by TR*Labs* (Telecommunications Research Laboratories) in Saskatoon, and by NSERC research grant OGP0121969.

The ProWGen tool is available from
http://www.cs.usask.ca/faculty/carey/software/

References

[1] G. Abdulla, E. Fox, M. Abrams, and S. Williams, “WWW Proxy Traffic Characterization with Application to Caching”, Technical Report TR-97-03, Computer Science Department, Virginia Tech., March 1997.

[2] M. Abrams, C. Standridge, G. Abdulla, S. Williams, and E. Fox, “Caching Proxies: Limitations and Potentials”, *Electronic Proceedings of the Fourth World-Wide Web Conference*, pp. 119-133, Boston, MA, December 1995.

[3] V. Almeida, M. Cesario, R. Fonseca, W. Meira Jr., and C. Murta, “Analysing the Behavior of a Proxy Server in Light of Regional and Cultural Issues”, *Proceedings of the Third International WWW Caching Workshop*, Manchester, England, June 1998.

[4] V. Almeida, A. Bestavros, M. Crovella, and A. Oliveira, “Characterizing Reference Locality in the WWW”, *Proceedings of the 1996 International Conference on Parallel and Distributed Information Systems (PDIS'96)*, pp. 92-103, December 1996.

[5] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, “Evaluating Content Management Techniques for Web Proxy Caches”, *Second Workshop on Internet Server Performance*, Atlanta, GA, May 1999.

[6] M. Arlitt and T. Jin, “A Workload Characterization Study of the 1998 World Cup Web Site”, *IEEE Network*, vol. 14, no. 3, pp. 30-37, May/June 2000.

[7] M. Arlitt and C. Williamson, “Trace-Driven Simulation of Document Caching Strategies for Internet Web Servers”, *Simulation Journal*, vol. 68, no. 1, pp. 23-33, January 1997.

[8] M. Arlitt and C. Williamson, “Internet Web Servers: Workload Characterization and Performance Implications”, *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 631-645, October 1997.

[9] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm, “World-Wide Web Caching: The Application Level View of the Internet”, *IEEE Communications*, vol. 35, no. 6, pp. 170-178, June 1997.

[10] M. Baentsch, L. Barum, G. Molter, S. Rothkugel, and P. Sturm, “Enhancing the Web's Infrastructure: From Caching to Replication”, *IEEE Internet Computing*, vol. 1, no. 2, pp. 18-27, March 1997.

[11] P. Barford and M. Crovella, “Generating Representative Web Workloads for Network and Server Performance Evaluation”, *Proceedings of ACM SIGMETRICS Conference*, Madison, WI, pp. 151-160, June 1998.

[12] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, “Changes in Web Client Access Patterns: Characteristics and Caching Implications”, *World Wide Web*, Vol. 2, No. 1, pp. 15-28, January 1999.

[13] A. Bestavros, R. Carter, M. Crovella, C. Cunha, A. Heddaya, and S. Mirdad, “Application-Level Document Caching in the Internet”, *Proceedings of the Second International Workshop on Services in Distributed and Networked Environments (SDNE'95)*, pp. 166-173, Whistler, BC, June 1995.

[14] J. Bolot and P. Hoschka, “Performance Engineering of WWW: Applications to Dimensioning and Cache Design”, *Electronic Proceedings of the Fifth International World-Wide Web Conference*, Paris, France, May, 1996.

[15] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “Web Caching and Zipf-like Distributions: Evidence and Implications”, *Proceedings of the IEEE Infocom Conference*, New York, NY, pp. 126-134, March 1999.

[16] M. Busari, *Simulation Evaluation of Web Caching Hierarchies*, M.Sc. Thesis, Department of Computer Science, University of Saskatchewan, June 2000.

[17] M. Busari and C. Williamson, “Simulation Evaluation of Heterogeneous Web Proxy Caching Hierarchies”, submitted for publication.

[18] R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich, “Web Proxy Caching: The Devil is in the Details”, *ACM Performance Evaluation Review*, vol. 26, no. 1, pp. 11-15, December 1998.

[19] P. Cao and S. Irani, “Cost-aware WWW Proxy Caching Algorithms”, *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pp. 193-206, December 1997.

[20] L. Cherkasova and G. Ciardo, “Characterizing Temporal Locality and its Impact on Web Server Performance”, *Proceedings of ICCCN'2000*, Las Vegas, NV, October 2000.

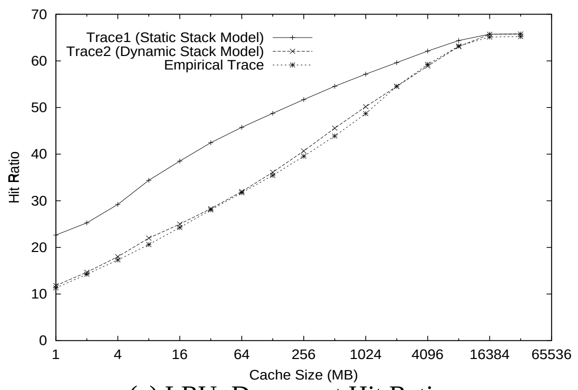
[21] E. Cohen, B. Krishnamurthy, and J. Rexford, “Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters”, *Proceedings of the ACM SIGCOMM Conference*, Vancouver, BC, pp. 241-253, September 1998.

[22] M. Crovella and M. Taqqu, “Estimating the Heavy Tail Index from Scaling Properties”, *Methodology and Computing in Applied Probability*, vol. 1, no. 1, 1999.

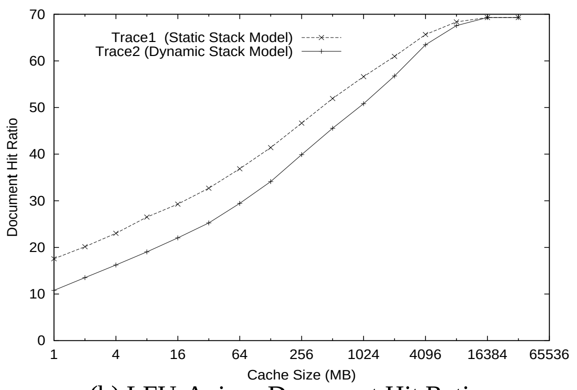
[23] M. Crovella and A. Bestavros, “Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes” *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 835-846, December 1997.

[24] C. Cunha, A. Bestavros, and M. Crovella, “Characteristics of WWW Client-based Traces”, Technical Report TR-95-010, Department of Computer Science, Boston University, July 1995.

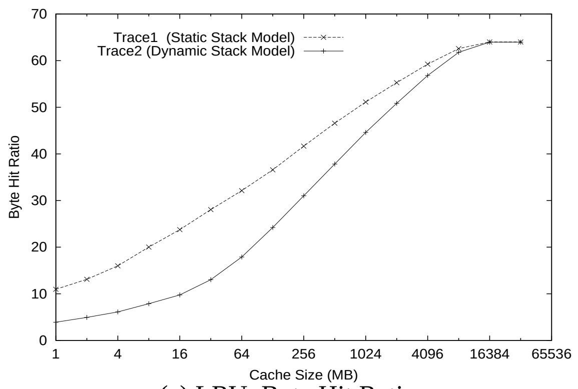
[25] J. Dilley, “The Effect of Consistency on Cache Response Time”, *IEEE Network*, vol. 14, no. 3, pp. 24-28, May/June 2000.



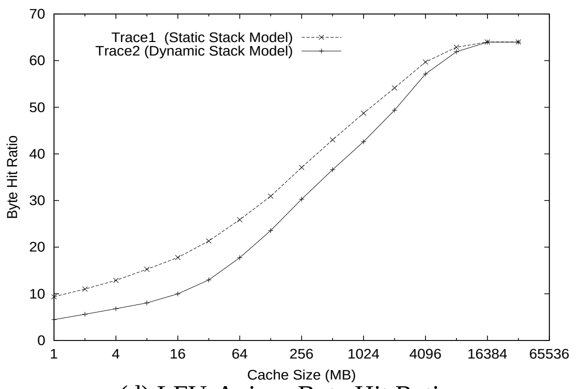
(a) LRU: Document Hit Ratio



(b) LFU-Aging: Document Hit Ratio



(c) LRU: Byte Hit Ratio



(d) LFU-Aging: Byte Hit Ratio

Fig. 7. Sensitivity of LRU and LFU-Aging Policies to Temporal Locality

[26] C. Dodge, B. Marx, and H. Pfeiffenberger, "Web Cataloguing Through Cache Exploitation and Steps Toward Consistency Maintenance", *Proceedings of the Third International World-Wide Web Conference*, April 1995.

[27] B. Duska, D. Marwood, and M. Feeley, "The Measured Access Characteristics of World-Wide Web Client Proxy Caches", *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pp. 23-35, December 1997.

[28] S. Jin and A. Bestavros, "Greedy-Dual* Web Caching Algorithms: Exploiting the Two Sources of Temporal Locality in Web Request Streams", *Proceedings of the Fifth Web Caching Workshop*, Lisbon, Portugal, May 2000.

[29] S. Jin and A. Bestavros, "Sources and Characteristics of Web Temporal Locality", *Proceedings of MASCOTS'2000*, pp. 28-35, San Francisco, CA, August 2000.

[30] A. Mahanti, *Web Proxy Workload Characterization and Modeling*, M.Sc. Thesis, Department of Computer Science, University of Saskatchewan, September 1999.

[31] A. Mahanti, C. Williamson, and D. Eager, "Traffic Analysis of a Web Proxy Caching Hierarchy", *IEEE Network*, vol. 14, no. 3, pp. 16-23, May/June 2000.

[32] A. Mahanti, D. Eager, and C. Williamson, "Temporal Locality and its Impact on Web Proxy Cache Performance", *Performance Evaluation*, Vol. 42, No. 2-3, pp. 187-203, October 2000.

[33] S. Manley, M. Seltzer, and M. Courage, "A Self-Scaling and Self-Configuring Benchmark for Web Servers", *Proceedings of the ACM SIGMETRICS Conference*, Madison, WI, June 1998.

[34] J. Mogul, "Squeezing More Bits Out of HTTP Caches", *IEEE Network*, vol. 14, no. 3, pp. 6-14, May/June 2000.

[35] D. Neal, "The Harvest Object Cache in New Zealand", *Proceedings of the Fifth International World-Wide Web Conference*, May 1996.

[36] D. Povey and J. Harrison, "A Distributed Internet Cache", *Proceedings of the 20th Australian Computer Science Conference*, Sydney, Australia, February 1997.

[37] References on Zipf's Law. Available at http://linkage.rockefeller.edu/wli/zipf/

[38] L. Rizzo and L. Vicisano, "Replacement Policies for a Proxy Cache", Technical Report RN/98/13, Department of Computer Science, University College London, 1998.

[39] C. Roadknight, I. Marshall, and D. Vearer, "File Popularity Characterization", *Proceedings of the Second Workshop on Internet Server Performance (WISP'99)*, Atlanta, Georgia, May 1999.

[40] P. Rodriguez, C. Spanner, and E. Biersack, "Web Caching Architectures: Hierarchical and Distributed Caching", *Proceedings of the Fourth Web Caching Workshop*, San Diego, CA, pp. 37-48, March 1999.

[41] A. Rousskov and V. Soloviev, "On Performance of Caching Proxies", *Proceedings of the ACM SIGMETRICS Conference*, June 1998.

[42] SPECweb99. http://www.spec.org/osg/web99/

[43] J. Spirn, "Distance String Models for Program Behaviour", *IEEE Computer*, vol. 9, no. 11, pp. 14-20, November 1976.

[44] WebBench 3.0. http://www.zdnet.com/zdbop/webbench/webbench.html

[45] Webjamma. http://www.cs.vt.edu/ chitra/webjamma.html

[46] S. Williams, M. Abrams, C. Standridge, G. Abdulla, and E. Fox, "Removal Policies in Network Caches for World-Wide Web Documents", *Proceedings of the ACM SIGCOMM Conference*, Stanford, CA, pp. 293-305, August 1996.