Modeling Correlations in Web-Traces and Implications for Designing Replacement Policies.

Konstantinos Psounis[†], An Zhu[*], Balaji Prabhakar[‡], Rajeev Motwani[*]
[†]Department of Electrical Engineering [*]Department of Computer Science
[‡]Departments of Electrical Engineering and Computer Science
Stanford University, Stanford, CA 94305
{kpsounis,anzhu,balaji}@stanford.edu, rajeev@cs.stanford.edu

Abstract

A number of web-cache-related algorithms, such as replacement and prefetching policies, rely on specific characteristics present in the sequence of requests for efficient performance. Further, there is an increasing need to synthetically generate long traces of web-requests for studying the performance of algorithms and systems related to the Web. These reasons motivate us to obtain a simple and accurate model of web-request traces.

Our Markovian model precisely captures the degrees to which temporal correlations and document popularity influence web-trace requests. We describe a mathematical procedure to extract the model parameters from real traces and generate synthetic traces using these parameters. This procedure is verified by standard statistical analysis. We also validate the model by comparing the hit ratios for real traces and their synthetic counterparts under various caching algorithms.

As an important by-product, the model provides guidelines for designing efficient replacement algorithms. We obtain optimal algorithms given the parameters of the model. We also introduce a spectrum of practicable, high-performance algorithms that adapt to the degree of temporal correlation present in the request sequence, and discuss related implementation concerns.

Keywords: request sequence model, web-traces, web-caching, replacement schemes, performance analysis.

1 Introduction

The use of web-caches for reducing network traffic and download latency has rendered them an important component of the Internet infrastructure. Algorithms devised for web caches need to take advantage of specific characteristics present in the request sequence for efficient performance. For example, cache replacement policies exploit the temporal locality between successive requests and the long-term popularity of web documents during eviction times. Prefetching policies take advantage of correlations present in inter-document requests (see [21], for example). Since algorithm design relies on the features inherent in HTTP request sequences, it is important to obtain a simple and accurate model of such sequences. This allows one to understand the impact of various characteristics (like temporal correlations, document popularity, etc) on the performance of the cache, and guides the design of better algorithms.

Broadly, there are three key features of web traffic that we wish to model. They are: (i) temporal correlations – this captures the likelihood of requesting a document in the near future, given that it has been currently requested, (ii) long-term document popularity – this captures the probability that a document is likely to be requested relative to other documents, and (iii) spatial (inter-document) correlations – this models the chance that document j will be requested next, given that document i has been requested now. In this work we focus on the relationship between temporal correlations and popularity as it relates to designing cache replacement schemes, and only briefly discuss spatial correlations. For the same reason, we also intially ignore other attributes of a document, such as its size.

Previous work on obtaining models of web traffic have observed that the distribution of the long-term popularity of documents is Zipf-like (see [1], [6] and the references therein). The authors in [16] point out that a Zipf-like popularity distribution alone doesn't suffice to capture temporal locality, the request sequence needs to be correlated in time. (For a more recent treatment see [13].) Our approach simultaneously models both temporal correlations and document popularity. Temporal correlations are captured using a Markov model for the request sequence. We do not assume specific parameters for the Markov chain, instead we infer them from the real traces we wish to model. Similarly, we also infer the distribution of the long-term popularity from the traces (and find that it is indeed Zipf-like). Further, our approach can easily model document popularities that change slowly over time, a phenomenon recently observed in traces [19].

Observations about the characteristics of web trace sequences have influenced the design of document replacement schemes. For example, the least recently used (LRU) algorithm is known to exploit temporal correlations present in the request sequence [28]. Hence, it is suitable for use when the trace has a high degree of temporal correlation. On the other hand, when the trace has a low temporal correlation, the least frequently used (LFU) algorithm exploits long-term document popularity to achieve high hit rates. Indeed, it is easy to see that when request sequences are independent and identically distributed, LFU achieves the highest hit rates [1] [11].

Real web request sequences, however, are influenced both by temporal correlations and document popularity. This observation and clever heuristics have led to the proposal of replacement algorithms which combine features of both LRU and LFU [2, 8, 10, 15, 16, 18]. Our model adds to this body of work in three ways: (a) It allows us to quantify the degree to which real traces are influenced by temporal correlations and document popularity, (b) it allows us to explore extreme cases by considering traces with different mixtures of temporal correlation and document popularity, and (c) this, in turn, points to a way of designing algorithms that trade-off correlations and popularity.

Traditionally, the performance of algorithms is studied using competitive analysis. Another approach is to probabilistically analyze the behavior of algorithms under realistic input. In the case of replacement algorithms, due to the difficulty in taking into account

[1]This result holds under the assumption that all documents are of the same size.

temporal correlations in an analytically tractable manner, these studies are usually done under the assumption of an independence reference model (e.g. [6, 29]). Our model allows to analytically study the performance of eviction schemes under request sequences that exhibit correlations.

Another contribution of our work is for designing good workload generation tools. An important challenge for these tools is to capture temporal correlations present in web request traces. This is commonly done using the LRU stack model (LRUSM) [27]. One issue with the LRUSM is that each document in the generated sequence of requests has an equal long-term probability of being requested [24], which is incompatible with the Zipfian long-term popularity observed in real traces. To solve this problem, researchers have to use various heuristics (see, for example, [4, 7]). Our model provides a rigorous and simple way to incorporate any long-term popularity distribution for the documents. Further, the LRUSM works with stack distances, whereas our model works directly with document requests, and hence it is a more natural tool for generating request sequences.

It has been observed that in addition to short-term correlation and long-term popularity, other attributes of a document, such as its size and the cost of fetching it from the origin-server, play an important role in the performance of web replacement schemes. Since the focus of this work is on the trade-off between temporal correlations and popularity, we initially ignore these other attributes. Later, we comment on how to incorporate these parameters in designing good eviction schemes. As a final comment, we limit our discussion to static documents and do not consider documents that change dynamically over time.

The paper is organized as follows. Section 2 introduces the model whose parameters are extracted from real traces. We then generate synthetic traces and verify that they exhibit the desired statistical properties. Further, we validate the model by comparing the hit rates for real traces and their synthetic counterparts under various caching algorithms. Based on the trace model, in Section 3 we formulate the problem of obtaining an optimal algorithm with respect to hit rate as a Markov decision process, and obtain an online replacement algorithm that is optimal conditioned on the state of the Markov process. This algorithm, however, relies on knowledge of trace characteristics and it is hard to use in practice. Thus, we introduce γ-LRU, a class of practicable replacement policies parameterized by γ with LRU and LFU obtained for extreme values of γ, and study its performance. Finally, we address implementation issues related to γ-LRU, like taking into account its data structure requirement and the variability in document sizes.

2 The Model

In this section we introduce our model for capturing long-term document popularity and short-term temporal correlation in web request sequences.

Index all documents in decreasing order of popularity. Let p_i be the probability of requesting the i^{th} most popular document. We shall call the probability distribution $\{p_i\}$ the distribution of the long term popularity of the documents. In practice, $\{p_i\}$ is known to be Zipf-like [6]; i.e., $P[Y_n = i] = p_i \propto 1/i^\theta$, $0 < \theta < 1$.

Let X_n be the document requested at time $n \geq 1$ (time is slotted, one request per slot), and Y_n, $n \geq 1$, be a sequence of random variables that are independent and identically distributed according to $\{p_i\}$. We propose the following model for web requests for $n > h$:

$$X_n = \begin{cases} X_{n-1} & \text{with probability } \alpha_1 \\ X_{n-2} & \text{w.p. } \alpha_2 \\ \vdots & \\ X_{n-h} & \text{w.p. } \alpha_h \\ Y_n & \text{w.p. } \beta, \end{cases}$$

where

$$\beta + \sum_{i=1}^{h} \alpha_i = 1. \tag{1}$$

In the model we keep a history of the h most recent requests to capture short-term correlations. Y_n injects documents that may or may not have been recently requested and captures long-term popularity. The probability that document i is going to be requested at time n, given that it was not requested in the last h requests is simply βp_i. The probability that document i is going to be requested at time n, given that it was requested only once in the last h requests at time $n-j$ $(1 \le j \le h)$, equals $\alpha_j + \beta p_i$. In general,

$$P(X_n = i | X_{n-1}, \ldots, X_{n-h}) = \beta p_i + \sum_{j=1}^{h} \alpha_j 1_{(X_{n-j}=i)}. \tag{2}$$

For mathematical convenience, another way to interpret the model is the following. At each time slot, toss an $h+1$-sided, biased coin to decide the value of X_n. Let T_n be a random variable indicating the outcome of the toss at time $n > h$. Then, $P(T_n = j) = \alpha_j$ for $1 \le j \le h$ and $P(T_n = h+1) = \beta$. Using this notation, the model is described by the following equation:

$$X_n = \sum_{j=1}^{h} X_{n-j} 1_{(T_n=j)} + Y_n 1_{(T_n=h+1)}. \tag{3}$$

The parameters of the model are the history window h, the α_i's, the β, and the p_i's. Let $\overline{X}_n = (X_n, X_{n-1}, \ldots, X_{n-h+1})$ and $\overline{i} = (i_1, i_2, \ldots, i_h)$ be a vector of h requested documents. Since $P(\overline{X}_{n+1} = \overline{i} \ | \overline{X}_n, \overline{X}_{n-1}, \ldots) = P(\overline{X}_{n+1} = \overline{i} \ | \overline{X}_n)$, $\overline{X}_n$ is a Markov chain or equivalent X_n is an h^{th}-order Markov chain. The $\overline{X}_n$, also X_n, has a unique stationary distribution since its state space is finite and it is irreducible and aperiodic (for $\beta > 0$).

In stationarity, X_n is distributed like Y_n. To see this, use Equation (3) to get

$$P(X_n = i) = \sum_{j=1}^{h} \alpha_j \cdot P(X_{n-j} = i) + \beta \cdot P(Y_n = i).$$

Now, in stationarity, X_n equals in distribution some random variable X. Therefore, $P(X_n = i) = P(X_{n-j} = i) = P(X = i)$ for all $j = 1, \ldots, h$, and we get

$$P(X = i) \cdot (1 - \sum_{j=1}^{h} \alpha_j) = \beta \cdot P(Y_1 = i)$$

for all i. Since $1 - \sum_{j=1}^{h} \alpha_j = \beta$, X has the same distribution as Y_1. (Recall that Y_n is i.i.d.)

2.1 Inferring Model Parameters

In this section we present techniques for inferring the parameters of the model, and use these techniques to obtain the parameters from real traces.

We infer $\{p_i\}$ from its empirical distribution; i.e. set p_i to equal the sample frequency of document i in the trace.

Inferring the α_i's and β is more involved. Let R be the number of requests in the trace. Suppose, for now, that a suitable value of the history parameter, h, has been chosen. (We present a procedure for determining h at the end of this section.) Let C_i, $1 \leq i \leq h$ be a counter that counts how many times the n^{th} request, $h < n \leq R$, is the same with the $(n-i)^{th}$ request. Denote by $c_i, 1 \leq i \leq h$, the ratio $\frac{C_i}{R-h}$. In other words,

$$c_i = \frac{\sum_{n=h+1}^{R} 1_{(X_n = X_{n-i})}}{R - h} \quad \text{for } 1 \leq i \leq h.$$

If the piece of real trace was indeed generated by the proposed Markov chain model, then for $1 \leq i \leq h$:

$$
\begin{aligned}
c_i &\approx P(X_n = X_{n-i}) \\
&= \sum_{j=1}^{h} P(X_{n-i} = X_{n-j}; T_n = j) + P(X_{n-i} = Y_n; T_n = h+1) \\
&= \sum_{j=1}^{h} P(X_{n-i} = X_{n-j}) P(T_n = j) + P(X_{n-i} = Y_n) P(T_n = h+1) \\
&= \alpha_i + \sum_{j=1, j \neq i}^{h} c_{|i-j|} \alpha_j + \beta P(X_{n-i} = Y_n). \qquad (4)
\end{aligned}
$$

Using the independence of Y_n and X_{n-i}, and the fact that X_{n-i} is distributed like Y_n in stationarity, $P(Y_n = X_{n-i}) = \sum_j P(Y_n = j | X_{n-i} = j) P(X_{n-i} = j) = \sum_j p_j^2$ for all $1 \leq i \leq h$.

Equation (4), for $1 \leq i \leq h$, together with Equation (1) form a linear system of $h+1$ equations with $h+1$ unknowns that can be solved to calculate α_i's and β. Use Equation (1) to substitute β with $1 - \sum_{i=1}^{h} \alpha_i$ in Equation (4). Then, the α_i's are computed by solving the linear system $A \cdot \alpha = B$ where $\alpha = (\alpha_i)$ is a $1 \times h$ vector, $B = (c_i - \sum_j p_j^2)$ is a $1 \times h$ vector, and A is an $h \times h$ matrix given by the following expression:

$$
A = \begin{pmatrix}
1 - \sum p_j^2 & c_1 - \sum p_j^2 & c_2 - \sum p_j^2 & \dots & c_{h-1} - \sum p_j^2 \\
c_1 - \sum p_j^2 & 1 - \sum p_j^2 & c_1 - \sum p_j^2 & \dots & c_{h-2} - \sum p_j^2 \\
c_2 - \sum p_j^2 & c_1 - \sum p_j^2 & 1 - \sum p_j^2 & \dots & c_{h-3} - \sum p_j^2 \\
\vdots & & & \ddots & \vdots \\
c_{h-1} - \sum p_j^2 & c_{h-2} - \sum p_j^2 & c_{h-3} - \sum p_j^2 & \dots & 1 - \sum p_j^2
\end{pmatrix}
$$

Once the α_i's are determined, β is computed from Equation (1).

What remains is a procedure for determining the proper history h. In theory, overestimating h, which results in a larger matrix, would still find the correct α_i's and history ($\alpha_i = 0$ for all $i > h$). However, in practice, larger history leads to more rounding and statistical errors, hence to small negative α_i's in the solution of the linear system. A solution to this is to start from an overestimated history, and lower its value until all the α_i's are positive.

This procedure terminates faster, the closer the overestimated history is to the actual history. To determine an overestimated history we execute the following steps: (i) We start with a large value of h, traverse the trace, and compute the c_i's, (ii) randomly permute the trace to cancel any short-term correlations, traverse the permuted trace, and compute the corresponding normalized counters, denoted by $c_i^{'}$, and (iii) output as an overestimated history the number of c_i's that are larger than the average value of the $c_i^{'}$'s. [2]

2.2 Generating Synthetic Traces

Synthetic traces can be generated using the model after inferring the parameters h, α_i, β, and p_i from a real trace. We experiment with daily proxy cache traces from the National Laboratory for Applied Network Research (NLANR) [20] recorded from September '99 to July '03. The length of the traces varies from 300000 to 700000 requests. The traces are from three different sites, the PA, the SD and the RTP site. [3]

Using the procedure described above, the proper history for most of the traces is computed to be around 5000. We also experiment with shorter histories, namely 500, 1000, 2000 and 3000. In the rest of this section we present results from one trace only, since there are no differences in the results obtained from the various traces.

Figure 1 presents the α_i's and β inferred from the trace for $h = 1000$. Notice that the α_i's are rapidly decreasing. Therefore, using a history that is smaller than the one

[2] In theory, $c_i^{'} = \sum p_j^2$ for all i. Since there is some positive correlation between consecutive requests of the original trace, $c_i \geq \sum p_j^2$ for some i. Hence, since the c_i's are decreasing, there is an index i_h such that $c_{i_h} < \sum p_j^2$ while $c_i \geq \sum p_j^2$ for $i < i_h$. This index is the overestimated history that we use.

[3] NLANR traces are a collection of traces from many sites. We use these sites because they have relatively long traces.

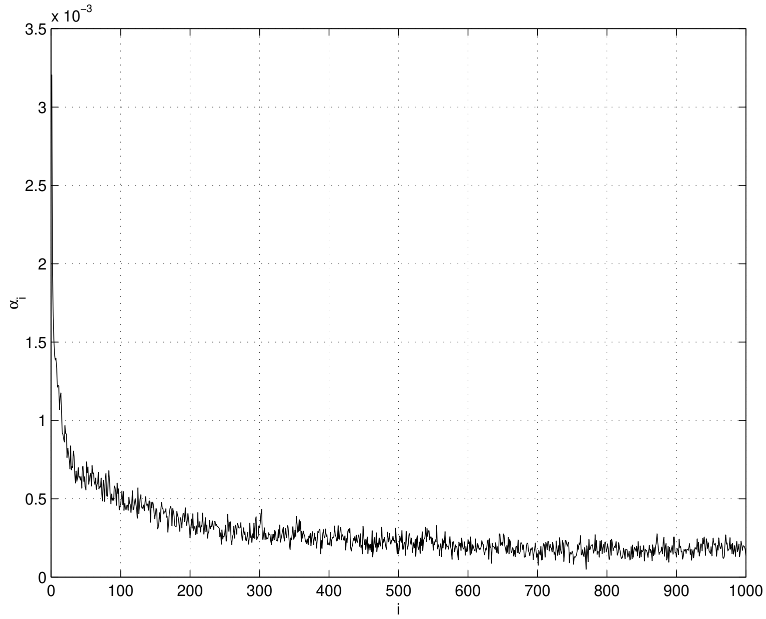

Figure 1: α_i's for the original trace (h=1000, $\beta = 0.708$).

obtained through the procedure above, suffices to capture the most essential short-term correlations. The rapid decrease of the values of the α_i's also implies that long-term correlations are very weak.

To determine whether the synthetic traces exhibit the same long-term popularities and short-term temporal correlations with their real counterparts, we perform the following procedure: (i) We verify that they possess the same history with the real traces, (ii) infer the rest of the model parameters from the synthetic traces, denoted by $\hat{\alpha}_i$, $\hat{\beta}$, and $\hat{p}_i$, and (iii) compare these parameters to the parameters inferred from the real traces.

h	β	$\hat{\beta}$
500	0.789	0.787
1000	0.708	0.707
2000	0.616	0.617

Table 1: β values for various histories.

In Table 1 we compare the values of β as obtained from the real and the corresponding

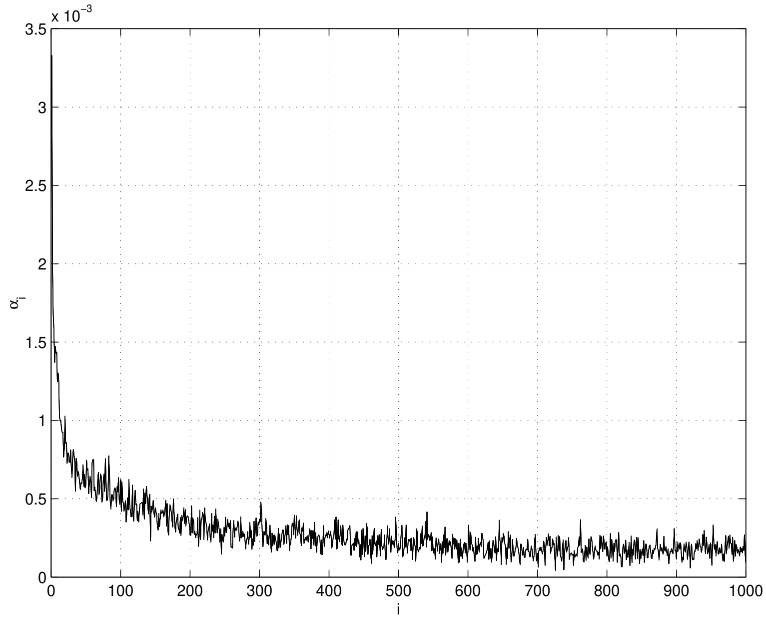

Figure 2: $\hat{\alpha}_i$'s for the synthetic counterpart of the original trace (h=1000, $\hat{\beta} = 0.707$).

synthetic trace. The values of β match very well for all histories shown. Figure 2 plots the $\hat{\alpha}_i$'s for the synthetic trace that is generated using the parameters of the real trace. By comparing Figures 1 and 2, it is evident that the plots match very well. [4] Note that generating longer synthetic traces will only improve the matching between the model parameters. We also generated synthetic traces of length half of that of the original trace, and the matching was quite good again.

In Figure 3 we plot the $\alpha_i' s$ of the real trace in logarithmic scale. The values of α_i roughly follow a power law with $\theta = 0.46$. We will use this observation to generate $\alpha_i' s$ for simulation purposes.

As expected, for popular documents $p_i \approx \hat{p_i}$ as it is shown in Figure 4. However, this is not the case for very unpopular documents since estimating the p_i's of such documents from the sample frequencies is not accurate. The insufficient statistics for unpopular

[4]If α_i's and $\hat{\alpha}_i$'s are plotted in the same plot, the plots are nearly indistinguishable with small differences only in the oscillations.

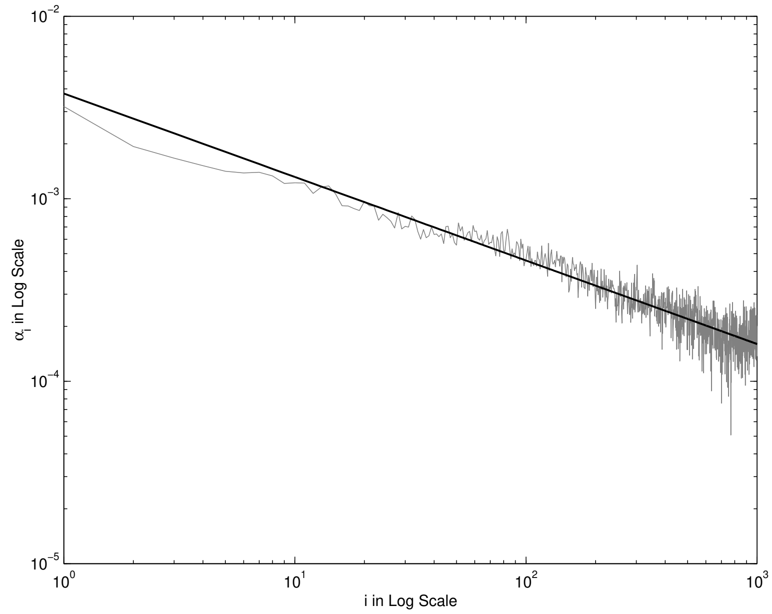

Figure 3: α_i's in logarithmic scale.

documents, and in particular one-timers, lead us to treat them in a special manner. We will first elaborate on the problems associated with one-timers, and then, we will describe one solution to circumvent these problems.

The first issue with unpopular documents is a well-known problem of statistics, often referred to as the problem of estimating the number of classes [14] or distinct values [9] in a population. In the context of web traces, the problem is that when a trace is not long enough, there are many unpopular documents that are not present in the trace. If one uses the sample frequencies to generate a trace with the same length as the original, there will be fewer one-timers in the generated trace than in the original. This is because the pool of available one-timers, from where the i.i.d sequence Y_n takes values, is a lot smaller than in reality. To fix this problem we need to estimate the actual number of unpopular documents available to the users from which the trace is recorded.

The second issue with unpopular documents is the following. Recall that in stationarity

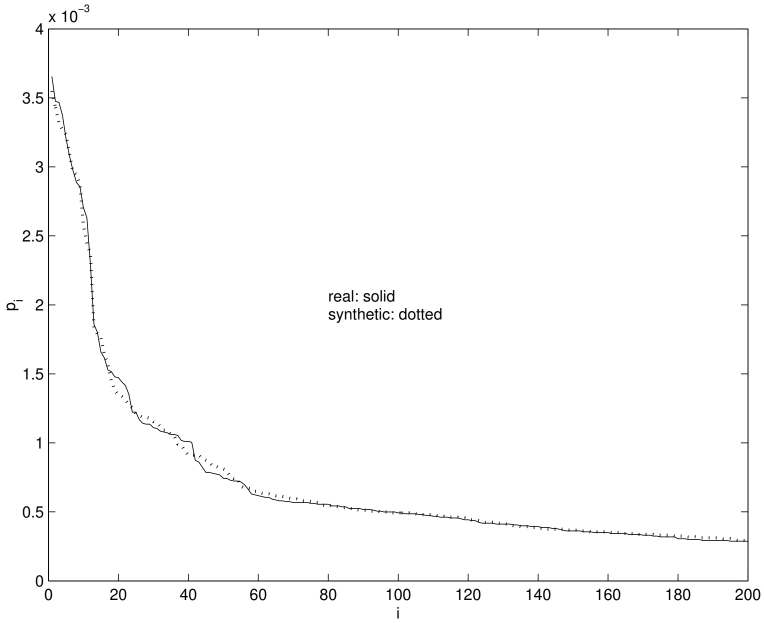

Figure 4: Long-term popularity comparison between the original trace and its synthetic counterpart for popular documents.

X_n is distributed like Y_n. However, since the available real traces are short, the statistics for unpopular documents are far from their stationary values, and the correlations inherent in the model force unpopular documents to be more popular. The problem is well exemplified with one-timers: Suppose a one-timer, say document j, is requested during the generation of a synthetic trace. Then, based on the model, the probability of requesting document j again in the next h time slots equals $1-\prod_{i=1}^{h}(\beta(1-p_j)+(1-\beta)(1-\alpha_i)) \approx 1-\prod_{i=1}^{h}(1-\alpha_i) \approx \sum_{i=1}^{h}\alpha_i = 1-\beta$. Hence, since this probability is positive, documents that appear only once in the real trace may appear more than once in the synthetic trace.

We solve these problems as follows: Whenever a one-timer is requested during the generation of a synthetic trace, we record a special identifier instead of the identifier corresponding to the document. After the trace is generated, we parse it and replace all the

Trace		LRU		LFU	
Date	Site	HR (real)	HR (synthetic)	HR (real)	HR (synthetic)
May 21^{st}, '01	PA	32.2%	32.1%	29.4%	29.1%
May 23^{rd}, '01	PA	31.7%	31.4%	28.5%	28.1%
January 19^{th}, '02	PA	30.7%	31.1%	28.9%	28.7%
January 20^{th}, '02	PA	32.7%	32.4%	31.1%	30.8%
July 10^{th}, '03	SD	29.3%	29.2%	25.2%	25.0%
July 10^{th}, '03	RTP	34.2%	34.2%	32.5%	32.4%

Table 2: Hit rates comparison for real and synthetic traces under LRU and LFU.

occurrences of the special identifier with one-timers. [5]

2.2.1 Model Validation

A word on model validation is in order. The proposed model captures two important properties of web request streams: (i) the Zipfian nature of long-term popularity, and, (ii) the short-term temporal correlations. There are other properties that are not captured, e.g. inter-document correlations.

To validate the model, we should investigate if it captures all the important properties of request streams for a given application. In this work we use web caching as our application. Hence, we should compare the hit rates achieved by real traces and their corresponding synthetic traces, under various cache replacement schemes. Table 2 compares the hit rates for six traces under LRU and LFU. (We use online LFU; i.e. the popularity/frequency of a document at time t is inferred from the requests that occured at times $t' \leq t$.) The hit rates are close for all traces. (Please refer to Table 5 of Section 3.3 for statistical characteristics of the used traces and for the cache sizes used in the simulations. Note that we do not take different document sizes into account here. Section 3.4 addresses this issue.) In Section 3.3 we compare the hit rates under more sophisticated eviction schemes, after a thorough discussion on cache replacement policies

[5]It is easy to see that the number of occurrences of the special identifier will be very close to the total number of one-timers.

that takes place in Section 3.

2.3 Extensions of the Basic Model

In this section we show how one can extend the proposed model to capture more properties of a request sequence. First, we briefly comment on how to take into account inter-document correlations. Further, we show how to extend the model to capture different degrees of temporal correlations for each document, and work with document popularities that change slowly over time.

A simple way to model inter-document correlations is to associate with each document i a set of neighboring documents $N_i = \{j_1, j_2, \dots\}$ that are strongly correlated with i. Let $|N_i|$ denote the cardinality of that set. Further, assume that all pairs of neighbors exhibit the same degree of inter-document correlation, and that these correlations die after one time slot. Then, it is straightforward to extend the basic model as follows:

$$X_n = \begin{cases} X_{n-1} & \text{with probability } \alpha_1 \\ X_{n-2} & \text{w.p. } \alpha_2 \\ \vdots & \\ X_{n-h} & \text{w.p. } \alpha_h \\ Y_n & \text{w.p. } \beta_1 \\ Z_n & \text{w.p. } \beta_2 \end{cases}$$

where $\alpha_1 + \dots + \alpha_h + \beta_1 + \beta_2 = 1$, $Z_n = j_k$ w.p. $1/|N_{X_{n-1}}|$ for all $k = 1, \dots, |N_{X_{n-1}}|$, and $N_{X_{n-1}} = \{j_1, j_2, \dots, j_{|N_{X_{n-1}}|}\}$ is the set of neighbors of the document requested at time $n-1$. Notice that even this simplistic way to account for inter-document correlations requires us to identify a set of neighbors for each document, which can be quite burdensome in practice.

Recall from Equation (2) that the α_i's are the same for all documents. We generalize this by assuming that the probability of requesting document i, given the last h requests, equals

$$P(X_n = i | X_{n-1}, \ldots, X_{n-h}) = \beta_i p_i + \sum_{j=1}^{h} \alpha_{j,i} 1_{(X_{n-j}=i)},$$

where $\alpha_{1,i} + \cdots + \alpha_{h,i} + \beta_i = 1$ for all i. If document i_1 is more popular than document i_2, then α_{j,i_1} may be larger than α_{j,i_2} for some j's. In particular, the $\alpha_{j,i}$'s corresponding to one-timers are equal to zero for all j. Thus, the issue with the basic model where one-timers appear more frequently in generated traces than in original ones due to correlations, is resolved. Notice that this extension requires to keep track of h parameters for each document, which is also burdensome in practice.

In the interest of simplicity and tractability, we do not study further these extensions to the basic model. The closeness of the hit rates for real and synthetic traces under various replacement schemes (Tables 2, 6, and 7) justify this decision.

Finally, note that our model can work with document popularities that change slowly over time, a phenomenon recently observed in traces [19]. All that is needed is to allow the distribution of Y_n to change slowly over time during the generation of a synthetic trace.

3 Caching Algorithms

As mentioned previously, researchers have proposed various caching algorithms that exploit both short-term temporal correlations and long-term popularity. Despite the good performance of these algorithms, the precise trade-offs between temporal correlations and popularity are not well-understood. Using the request model introduced in Section 2 we can shed light on these trade-offs, formulate the problem of obtaining an optimal algorithm with respect to hit rate as a Markov decision process, and obtain an online replacement algorithm that is optimal conditioned on the state of the Markov process.

We also introduce γ-LRU, a class of practicable replacement policies parameterized by γ, $0 < \gamma \leq 1$, with LRU and LFU obtained for extreme values of γ. When γ is properly chosen, the performance of γ-LRU is close to the performance of the optimal policy.

3.1 Optimal Replacement Policies

We start by introducing some notation and terminology. Let K be the size of the cache. Fix a replacement policy r. Let $\overline{Z}_n^r$ be a random vector with the K elements $Z_{n,i}^r$, $i = 1, \ldots, K$, each of which holds the document at the i^{th} position of the cache at time n.

Let H_n^r be a random variable equal to 1 if there is a hit at time n, and zero otherwise. The long-term hit rate of a replacement policy r equals

$$HR^r \triangleq \lim_{N \to \infty} \frac{\sum_{n=1}^{N} H_n^r}{N}.$$

The H_n^r is a function of $\overline{Z}_n^r$ and X_n. Since $\overline{Z}_n^r$ depends only on the process X_n, and X_n is stationary and ergodic, the ergodic theorem [12] states that

$$\frac{\sum_{n=1}^{N} H_n^r}{N} \overset{a.s.}{\to} E(H_1^r) = P(H_1^r = 1). \tag{5}$$

Recall that $\overline{X}_n = (X_n, \ldots, X_{n-h+1})$. It is easy to see that $(\overline{Z}_n^r, \overline{X}_n)$ is a Markov chain with a finite state space S. Denote by π_s^r the stationary probability of being at state $s \in S$. [6] Then,

$$HR^r \overset{(5)}{=} P(H_n^r = 1) = \sum_{s \in S} \pi_s^r P(H_n^r = 1 | s). \tag{6}$$

A replacement policy r^* is optimal with respect to hit rate if it maximizes the stationary

[6]The Markov chain $(\overline{Z}_n^r, \overline{X}_n)$ is aperiodic for $\beta > 0$ hence it has a stationary distribution.

probability $P(H_n^r = 1)$ over all policies r. Finding r^* is the object of the theory of Markov Decision Processes [23], [26]. Using the vocabulary of this field, the Markov chain $(\overline{Z}_n^r, \overline{X}_n)$ is observed to be in a particular state s_n at time n. After observation of the state, an *action* must be chosen: which document to evict from the cache in case of a miss. [7] Based on the state s_n and the action chosen, a *reward* $R(s_n, r) = P(H_{n+1}^r = 1|s_n)$ is earned and the probability distribution for the next state is determined.

The action for every state is dictated by a stationary policy r. The problem of interest is to determine the policy that maximizes the average reward over an infinite time horizon, $\lim_{N\to\infty} E(\frac{1}{N}\sum_{i=1}^N R(s_i, r))$. It is easy to see by interchanging the limit and the expectation, [8] and using ergodicity, that the average reward is identical to the long-term hit rate. Hence,

$$r^* = \arg\max_r [HR^r] = \arg\max_r \left[\lim_{N\to\infty} E\left(\frac{1}{N}\sum_{i=1}^{N} R(s_i, r)\right)\right].$$

The theory of Markov Decision Processes provides a mathematical framework to identify optimal policies. However, usually only numerical solutions are possible. Furthermore, in many problems including the one we study, the computational requirements to obtain a numerical solution are overwhelming, because the number of states and/or policies is very large. In such situations, suboptimal approximate solutions are derived [5]. Investigating optimality along these lines is out of the scope of this work. Instead, we set for a weaker optimality criterion, namely we wish to identify the policy r that maximizes the reward $R(s, r)$ for all states $s \in S$. This policy is, by definition, optimal with respect to the hit rate conditioned on the current state. In particular, it maximizes $P(H_n^r = 1|s)$ in Equation (6) for all s but does not necessarily maximize $P(H_n^r = 1)$.

An optimal algorithm: Suppose the cache is full. A new request arrives, and the

[7] In case of a miss, $X_n \neq Z_{n,i}^r$ for all i, $1 \leq i \leq K$, and there are $K+1$ documents out of which one should be evicted. In case of a hit, the action is null.

[8] This is justified by the bounded convergence theorem [12].

	$\beta = 0.5$	$\beta = 0.75$	$\beta = 0.95$
LocalOpt	65.34%	47.98%	34.09%
LRU	59.01%	38.55%	22.20%
LFU	34.00%	32.23%	31.23%

Table 3: Hit rate comparison for LocalOpt, LRU, and LFU under various values of β.

cache must evict one out of the $K + 1$ documents. Order the documents in the cache according to their probability of being requested at the next request time, given the last h requests. Call by *LocalOpt*, the algorithm which evicts the document with the smallest probability of being requested at the next iteration. LocalOpt maximizes $P(H_n^r = 1|s)$ for all s by definition.

The probability of requesting document i at the next request time, given the last h requests, is given by Equation (2) for all i. We assume that LocalOpt either knows the p_i's, α_i's and β, or uses the ideas of Section 2.1 to estimate them. Thus, the algorithm can identify the document to evict at each eviction time. In the rest of the section we use LocalOpt as a benchmark for other algorithms.

How much better is LocalOpt than LRU and LFU? In particular, we are interested in comparing the hit rates of LocalOpt, LRU, and LFU under various request sequences with different temporal correlation characteristics. Thus, we use our model to generate such sequences.

We generate request sequences consisting of 5 million requests, drawn from a pool of 10000 documents whose popularity distribution is Zipf-like with parameter 0.5. We set h to 100, β to 0.5, 0.75 and 0.95, and assign to α_i's the rest of the probability according to a Zipf-like distribution with parameter 0.5. The cache size is set to 1000.

A relatively small value of β indicates a request sequence where short term correlations are more significant as compared to a request sequence with a larger β. Thus, smaller values of β correspond to larger hit rates.

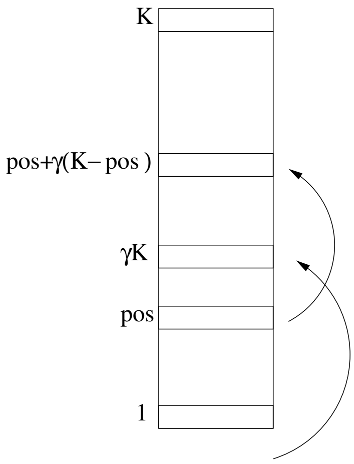

Figure 5: How documents move in a cache that uses γ-LRU.

Table 3 shows the hit rate of LocalOpt, LRU, and LFU. When β is relatively small, LRU's performance is quite good (the hit rate of LRU is at least $1-\beta$) while LFU's performance is bad. This is expected since the primary characteristic of the request sequence is the strong short-term correlations. On the other hand, when β is relatively large, LFU's performance is close to that of LocalOpt, while LRU's performance is bad. Therefore, both LRU and LFU fail to adapt to the degree of short-term correlation versus long-term popularity.

3.2 The γ-LRU Class of Replacement Policies

Given a trace, which algorithm performs better depends on the degree of temporal correlation versus popularity. LocalOpt adapts to the degree of correlation, but requires the knowledge of α_i's and β which is hard to compute in practice. This motivates us to introduce γ-*LRU*, a class of practicable replacement policies parameterized by γ with LRU and LFU obtained for extreme values of γ, that also adapts to the degree of correlation in a trace.

γ-LRU algorithm: Consider the following variation of LRU: Whenever a page is re-

	$\beta = 0.5$	$\beta = 0.75$	$\beta = 0.95$
LocalOpt	65.34%	47.98%	34.09%
$\gamma^* - LRU$	62.81%	45.61%	32.25%
γ-LRU ($\gamma = \frac{h}{K}$)	61.77%	44.87%	31.86%
LRU	59.01%	38.55%	22.20%
LFU	34.00%	32.23%	31.23%

Table 4: Hit rate for γ-LRU and other schemes using the same parameters as in Table 3.

quested, instead of moving the page to the top of the linked-list move it half way up. More generaly, whenever a page is requested move it up by a fraction γ, $0 < \gamma \leq 1$. This is how γ-LRU performs. In particular, name the positions in the cache $1, 2, \ldots, K$ moving from bottom to top, as illustrated in Figure 5. If the requested page is not in the cache, then insert it at position γK. Else, if it is currently at position pos, move it up to position $pos + \lceil \gamma(K - pos) \rceil$. [9]

How well does γ-LRU perform? Table 4 shows the hit rate of γ-LRU for the same parameters as in Table 3. The entries of Table 3 are repeated for comparison. It follows from this table that γ-LRU performs closely to LocalOpt for all values of β. In Table 4 we show the performance of γ-LRU for $\gamma = h/K$ and for the optimal γ, i.e., the γ value that maximizes hit rate, denoted by γ^*. We note that the performance of γ-LRU for $\gamma = h/K$ is close to the performance of $\gamma^* - LRU$. Later in this section we discuss this issue in more detail.

The closeness in the performance of γ-LRU and LocalOpt, for a large range of β values, can be intuitively explained by the documents that they cache. LocalOpt retains two classes of documents: (i) recently accessed documents, because their probability of being requested is high due to the α_i's, and (ii) very popular documents, because their

[9] In a different context, the authors in [3] introduce POS(k), another parameterized algorithm that performs Transpose [25] in the k topmost positions of the cache, and LRU in the rest of the positions. POS(k) has the same behavior with γ-LRU for the two extreme values of k and γ. However, it behaves very differently from γ-LRU for intermediate values of k and γ, and has worse performance. The problem with POS(k) is that it requires an impractically large period of time to gather popular documents at the top of the cache.

probability of being requested is large due to βp_i. γ-LRU also retains these two classes of documents; the former in the lower part of the cache, and the later in the upper part of the cache.

3.2.1 Extreme Values of γ

Let K be the size of the cache. Then, $1/K \leq \gamma \leq 1$. For $\gamma = 1$ γ-LRU coincides with LRU by definition. We will show that for $\gamma = 1/K$, γ-LRU closely resembles LFU in stationarity.

When $1/K$-LRU is used, [10] documents that get requested while in the cache swap positions with the ones that are above them, and documents that get requested while being outside the cache swap position with the document at the bottom of the cache.

Let the total population of documents in the universe be N. Index the documents in decreasing order of their popularity. Denote by S_n the cache occupancy just prior to the n^{th} request. S_n is a Markov chain. Depending on the eviction scheme used, this chain may have up to $\binom{N}{K}K!$ states, corresponding to all possible orderings of all possible K-tuples of the N documents.

Call by *Offline-LFU* the algorithm which knows the overall popularity of the documents in advance and always keeps the most popular documents in the cache. For long request sequences, the empirical distribution that LFU uses is quite close to the actual distribution, and LFU and Offline-LFU have nearly the same behavior. When Offline-LFU is used, $S_n = (1, 2, \ldots, K)$ for all n. We will analyze $1/K$-LRU and show that this is the highest probability state of the corresponding Markov chain, and that most of the other states have significantly lower probability.

Let $1/K$-LRU be the eviction scheme used. In this case, S_n is time-reversible [11] [17] and

[10] $1/K$-LRU is analogous to Transpose [25], which is an algorithm for list search.

[11] When γ is larger than $1/K$ the corresponding Markov chain is not time-reversible.

it is easy to find the steady-state distribution in closed-form. Denote by $\pi_{(i_1,i_2,\ldots,i_K)}$ the steady state probability of state $i = (i_1, i_2, \ldots, i_K)$, i.e., document i_1 occupies position 1 in the cache, i_2 occupies position 2 in the cache and so on. Further, denote by P_{ij} the transition probability from state i to state j. One may verify that the condition

$$\pi_i P_{ij} = \pi_j P_{ji} \tag{7}$$

is satisfied by the stationary distribution:

$$\pi_{(i_1,i_2,\ldots,i_K)} = C \cdot p_{i_1}^K p_{i_2}^{K-1} \cdots p_{i_K}, \tag{8}$$

where C is a normalizing constant. For example, if $\pi_i = \pi_{(1,2,\ldots,K-1,K)}$ and $\pi_j = \pi_{(1,2,\ldots,K-1,K+1)}$, then $P_{ij} = p_{K+1}$ and $P_{ji} = p_K$, and Equation (7) holds.

Since the p_i's are decreasing as i increases, the highest probability state is $(1, 2, \ldots, K)$. Also, since p_i's are Zipf-like distributed, the probability of the states where unpopular documents reside in the cache is very low.

3.2.2 On the Optimal Value of γ

It is interesting to further investigate the behavior of γ-LRU as γ varies from $1/K$ to 1. Let γ^* denote the optimal value of γ with respect to hit rate given the history h, α_i's, β, p_i's, and the cache size K. We first argue that $\gamma^* \leq \frac{h}{K}$.

To see this, name the positions in the cache $1, 2, \ldots, K$ moving from bottom to top. Recall that $Z_{n,i}$ is a random variable indicating the document occupying position i at time n. Also, recall that in stationarity the hit rate of a replacement policy equals $P(H_n = 1) = \sum_{i=1}^{K} P(X_n = Z_{n,i})$, where H_n equals 1 if there is a hit at time n and zero otherwise, and X_n represents the document requested at time n.

If $\gamma K \geq h$ then all documents stay in the cache for at least h time slots, after their last request before they leave the cache. Since $\alpha_{h+i} = 0$ for all $i \geq 1$, these replacement policies take full advantage of short-term temporal correlation. Thus, $P(H_n = 1) = \sum_{j=1}^{h} a_j + \beta \sum_{i=1}^{K} P(Y_n = Z_{n,i})$ where Y_n, $n \geq 1$, is independent and identically distributed according to $\{p_i\}$.

The sum $\sum_{i=1}^{K} P(Y_n = Z_{n,i})$ is maximized by Offline LFU, since Offline LFU places in the cache the K most popular documents. Also, the more popular documents a replacement policy places in the cache, the larger this sum is. Now, the smaller the value of γ, the more popular are the documents in the cache. [12] Thus, this sum is larger for $\gamma K = h$ than for $\gamma K > h$, and the same is true for the hit rate.

Very small values of γ suffer from very large mixing times of the corresponding Markov chains. In practice, this means that it takes too much time for the cache to be populated with the most popular documents in the upper part of the cache. We choose to use γ values close to h/K since we find their performance to be close to optimal, see for example Table 4, and the corresponding mixing times to be small.

Figure 6 plots the hit rate of γ-LRU under the second synthetic trace used in Table 3 ($\beta = 0.75$), as a function of γ. For $\gamma = 1$ the hit rate equals that of LRU, and for small γ it is close to LFU's hit rate. Also, the maximum hit rate is achieved for a value of γ less than h/K while for $\gamma = h/K$ the performance is very close to optimal.

3.3 Performance Under Real Traces

In this section we study the performance of γ-LRU under real web traces from *NLANR* [20]. We also compare the hit rate obtained for real traces and their synthetic counterparts under various algorithms, to investigate whether the request model captures the essential properties of real traces as they relate to web caching.

[12] We don't have a formal proof for this statement. Intuitively, larger values of γ cause larger perturbations whenever unpopular documents are requested. This intuition is verified by simulations, see for example Figure 6.

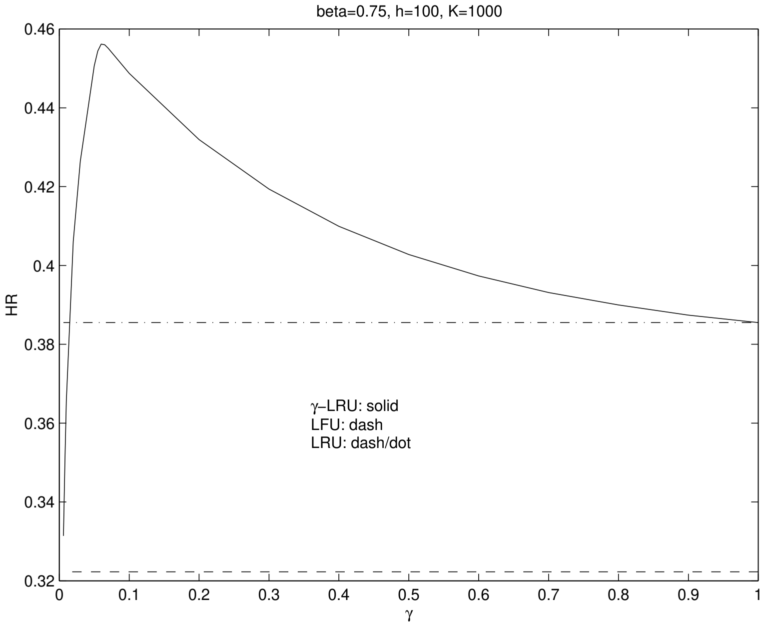

Figure 6: Hit rate of γ-LRU as a function of γ under the second synthetic trace (β=0.75) of Table 3. $\frac{h}{K} = 0.1$.

We present results from six traces from three different sites, recorded in May '01, January '02, and July '03. For the generation of the synthetic traces we use the procedure presented in Section 2. The history is set to 1000, and the parameters α_i's, β, and p_i's are inferred from the traces. The cache size is set to 5% of the total number of distinct requests in each trace, and the value of γ is set to 0.1 which is close to h/K in all the simulations. Finally, the cache is warmed up in all simulations. We don't take document size into account for now. Section 3.4 addresses this issue.

Table 5 reports the total requests and total distinct requests of each of the traces, as well as the cache size used in each case.

Table 6 presents the hit rates of LocalOpt, γ-LRU, LRU and LFU under the two May traces and their synthetic counterparts. Tables 7 and 8 present the same quantities for

Trace	Total Requests	Distinct Requests	Cache Size
May 21^{st} '01, PA	558826	275136	13500
May 23^{rd} '01, PA	508980	261480	13000
January 19^{th} '02, PA	512458	256486	12800
January 20^{th} '02, PA	507827	238785	11900
July 10^{th} '03, SD	434604	265674	13200
July 10^{th} '03, RTP	681383	387828	19400

Table 5: Trace characteristics and cache size.

	May 21^{st} '01, PA		May 23^{rd} '01, PA	
Scheme	HR (real)	HR (synthetic)	HR (real)	HR (synthetic)
LocalOpt	37.8%	41.1%	38.8%	40.2%
γ-LRU	35.3%	36.6%	35.1%	36.1%
LRU	32.2%	32.1%	31.7%	31.4%
LFU	29.4%	29.1%	28.5%	28.1%

Table 6: Hit rates under May '01 traces and their synthetic counterparts.

	January 19^{th} '02, PA		January 20^{th} '02, PA	
Scheme	HR (real)	HR (synthetic)	HR (real)	HR (synthetic)
LocalOpt	37.8%	41.2%	39.6%	42.5%
γ-LRU	34.9%	36.5%	36.6%	37.7%
LRU	30.7%	31.1%	32.7%	32.4%
LFU	28.9%	28.7%	31.1%	30.8%

Table 7: Hit rates under January '02 traces and their synthetic counterparts.

	July 10^{th} '03, SD		July 10^{th} '03, RTP	
Scheme	HR (real)	HR (synthetic)	HR (real)	HR (synthetic)
LocalOpt	33.4%	35.5%	39.3%	41.2%
γ-LRU	31.8%	33.3%	37.1%	39.0%
LRU	29.3%	29.2%	34.2%	34.2%
LFU	25.2%	25.0%	32.5%	32.4%

Table 8: Hit rates under July '03 traces and their synthetic counterparts.

the January and July traces. As expected, LocalOpt is superior to all other algorithms. Also, γ-LRU is competitive against both LRU and LFU. Results are similar for all the other daily NLANR traces that we tried.

The hit rates obtained for the real traces and their synthetic counterparts are close. LocalOpt performs better under synthetic traces since it is designed to exploit the statistical characteristics that the synthetic traces exactly possess. The small differences in the hit rates are likely to be due to (i) the insufficient statistics for the unpopular documents, (ii) the inter-document correlations present in the real traces that are not modeled, and (iii) the fact that the values of the α_i's are the same for all the documents (see Section 2.3 for a related discussion). These modeling decisions do not introducing significant inaccuracies, yet they simplify the model substantially.

3.4 Implementation Concerns

To be able to use γ-LRU in practice, we must address some implementation concerns. First, the size of the documents should be taken into consideration. It is straightforward to extend γ-LRU to take size into account, by making γ depend on the document size. Let $\gamma(S)$-LRU be such an extension where S denotes size and $\gamma(S)$ is a decreasing function of S. [13]

We use a specific function $\gamma(S)$ and calculate the performance of $\gamma(S)$-LRU using real traces. The goal is to show that $\gamma(S)$-LRU is competitive against other high performance replacement schemes that take into account the size and the frequency of use in their eviction decision, in addition to recency of use.

We avoid doing an elaborate design of $\gamma(S)$, or optimizing its design. For $S = 30$KB, $\gamma(S)$ is set to 0.1. (We use 0.1 as the typical γ value. This value works well for all the

[13] Taking into account the size of a document independently of its popularity is justified by the evidence that there is no correlation between them [7].

conducted simulations, since the performance of γ-LRU under real traces is good for a wide range of γ values around h/K.) To penalize large documents, as S grows from 30KB to 250KB, γ drops quadratically to 0.05. Documents larger than 250KB are never cached. As S decreases from 30KB to 1KB, γ increases linearly to 0.2 and retains this value for smaller document sizes.

We compare $\gamma(S)$-LRU to Greedy-Dual-Size-Frequency-Connection (GD-SFC), a simple extension of the Greedy-Dual-Size (GD-Size) algorithm [8]. GD-SFC associates each document in the cache with a value, called eviction value, and evicts the document with the minimum such value. Whenever there is a request for a document i, the new eviction value for that document is set to

$$f_i = \min(f_j : j \text{ in cache}) + \frac{C_i \cdot F_i}{S_i},$$

where C_i is an estimate of the latency for connecting to the corresponding server, F_i is the number of times the document has been requested since it entered the cache, and S_i is the document size. Note that the quantity $\min(f_j : j \text{ in cache})$ is increasing in time and is used to take into account the recentness of a document. Indeed, whenever a document is accessed, its eviction value is increased by the currently minimum eviction value. Thus, the most recently used documents tend to have larger eviction values.

The traces we use are taken from (NLANR) [20]. We only present results from a daily trace of length 500000, since there are no significant differences in the results obtained from the various traces. We only simulate requests with a known reply size.

The performance criteria used are three:

(i) the hit rate (HR), which is the fraction of client-requested URLs returned by the proxy cache,

(ii) the byte hit rate (BHR), which is the fraction of client requested bytes returned

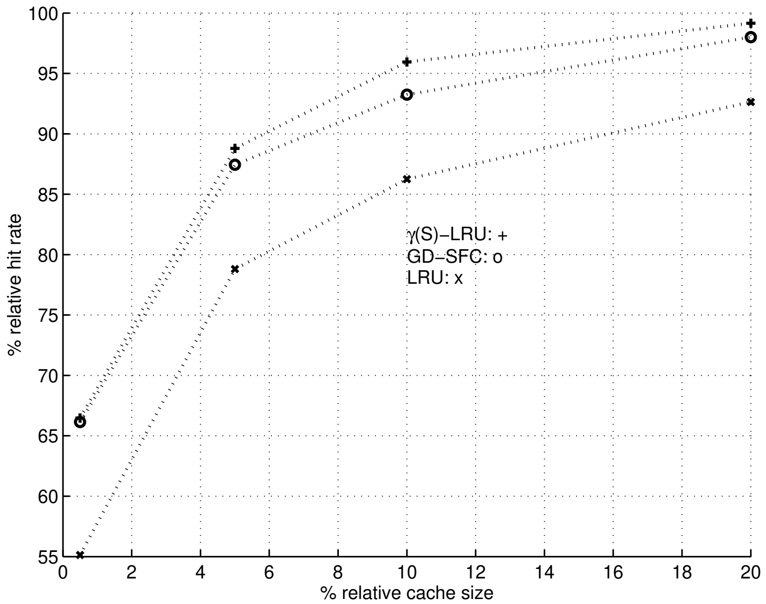

Figure 7: Hit rate comparison between $\gamma(S)$-LRU and GD-SFC. LRU's hit rate also shown.

by the proxy cache, and

(iii) the latency reduction (LR), which is the reduction of the waiting time of the user from the time the request is made till the time the document is received (download latency), over the sum of all download latencies.

For each trace, the HR, the BHR and the LR are calculated for a cache of infinite size. Then, they are calculated for a cache of size $0.5\%, 5\%, 10\%$, and 20% of the maximum size required to avoid any evictions. This size is around 2GB.

Figure 7 presents the ratio of HR of $\gamma(S)$-LRU and GD-SFC over the HR achieved by an infinite size cache. The performance of the two schemes is nearly the same. Figure 8 presents the ratio of BHR of $\gamma(S)$-LRU and GD-SFC over the BHR achieved by an infinite size cache. The performance of $\gamma(S)$-LRU is superior to the performance of GD-SFC for small cache sizes. This is expected since GD-SFC has a strong bias against

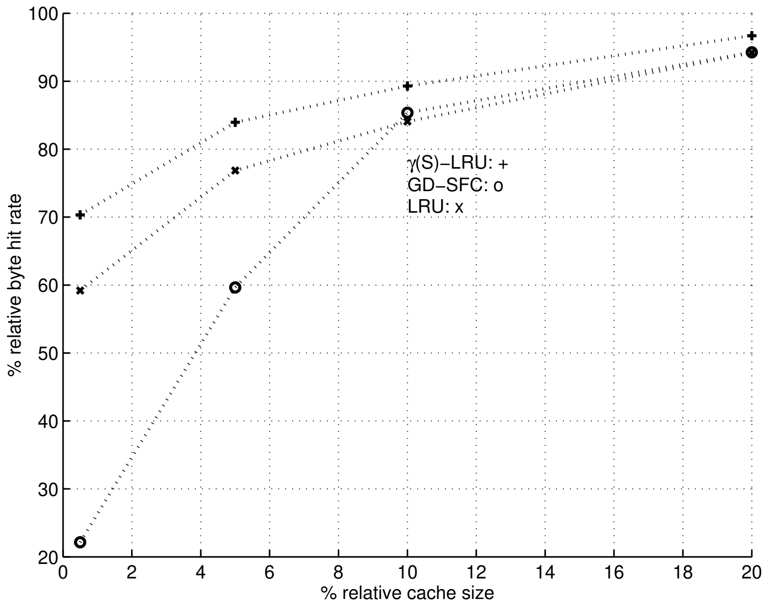

Figure 8: Byte hit rate comparison between $\gamma(S)$-LRU and GD-SFC. LRU's byte hit rate also shown.

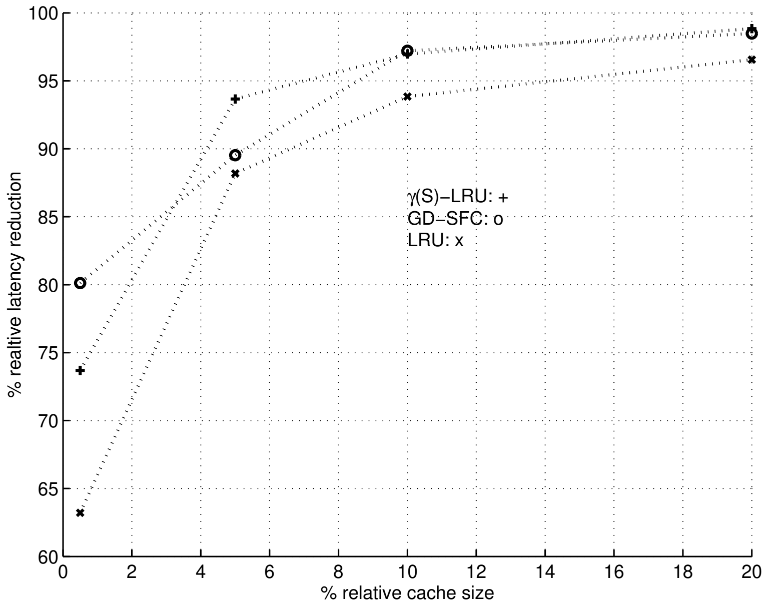

Figure 9: Latency reduction comparison between $\gamma(S)$-LRU and GD-SFC. LRU's latency reduction also shown.

large size documents even when these documents are popular. Figure 9 presents the ratio of LR of $\gamma(S)$-LRU and GD-SFC over the LR achieved by an infinite cache. The performance of the two schemes is more or less the same.

Note that algorithms like GD-SFC usually estimate the frequency of use by recording the number of times documents are requested once in the cache, since it is not feasible to keep counters for all documents. Thus, they follow an approach similar to in-cache-LFU, which has worse performance than LFU. On the other hand, by design, $\gamma(S)$-LRU identifies popular documents as efficiently as LFU.

In addition to presenting the performance of $\gamma(S)$-LRU and GD-SFC, Figures 7, 8, and 9 also present the performance of LRU. The difference in the performance of the three schemes decreases rapidly as the relative cache size increases. Indeed, it has been observed that for very large caches, the performance of different replacement schemes is similar [8, 22]. The model introduced in Section 2 provides the following explanation for this phenomenon: (i) due to the Zipf-like distribution of the popularity of web documents, the fraction of documents that corresponds to most of the hits is small enough to fit in the cache, and (ii) due to the rapid decrease of temporal correlations (Figure 3), large caches store documents long enough to fully exploit these correlations.

Another implementation concern is the data structure required to implement γ-LRU. LRU uses a linked list to maintain the order of the documents in the cache. However, using a simple linked list for γ-LRU is not efficient. Indeed, linked lists are efficient when insertion and deletion operations take place at the top and the tail of the list only. In γ-LRU it is required to perform insertions at arbitrary places, which has complexity $O(\log n)$. To maintain constant-time complexity, we propose to only allow insertions to happen at predefined positions properly distributed along the length of the list. Each list entry will have a pointer pointing to the predefined position that the entry should jump to, if it is accessed. It is easy to see that whenever a jump takes place, only two of these pointers need to be updated.

The focus of this work is on the modeling of web trace correlations. The γ-LRU algorithm serves to illustrate the implications of these correlations on designing replacement policies. Since it is not our goal to fully investigate the performance of γ-LRU as a replacement scheme in practice, we refrain from further elaboration on $\gamma(S)$-LRU with predefined insertion positions, and from a comparison of its performance with other high-performance replacement schemes.

4 Conclusions

In this paper we have proposed a model that precisely captures the degrees to which temporal correlations and document popularity influence web trace requests. We also described a mathematical procedure to extract the parameters of the model from real traces. The model was used to generate synthetic traces of arbitrary length exhibiting any degree of correlation, and provide guidelines for designing efficient replacement algorithms.

We formulated the problem of obtaining an optimal replacement algorithm with respect to hit rate as a Markov decision process, and obtained an online replacement algorithm that is optimal conditioning on the state of the Markov process. We also introduced γ-LRU, a class of practicable replacement policies parameterized by γ with LRU and LFU obtained for extreme values of γ. Finally, we showed that γ-LRU performs close to the optimal policy, under any degree of temporal correlation present in the request sequence, and extended the algorithm to take into account documents of varying sizes.

References

[1] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing reference locality in the www. In *Proceedings of IEEE PDIS*, 1996.

[2] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin. Evaluating content management techniques for web proxy caches. In *Proceedings of the 2nd Workshop on Internet Server Performance*, May 1999.

[3] A.Tenenbaum and R. Nemes. Two spectra of self-organizing sequential search algorithms. *SIAM Journal of Computing*, 11(3), 1982.

[4] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the ACM SIGMETRICS Conference*, June 1998.

[5] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[6] L. Breslau, P. Cao, L. Fan, G. Philips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of IEEE INFOCOM*, New York, 1999.

[7] M. Busari and C. Williamson. Prowgen: a synthetic workload generation tool for simulation evaluation of web proxy caches. *Computer Neworks*, 38(6):779–794, 2002.

[8] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, December 1997.

[9] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards estimation error guarantees for distinct values. In *Proceedings of the 19th ACM Symposium on Principles of Database Systems (PODS)*, 2000.

[10] L. Cherkasova and G. Ciardo. Role of aging, frequency, and size in web cache replacement policies. In *Proceedings of HPCN Europe*, 2001.

[11] E. Coffman and P. Denning. *Operating Systems Theory.* Prentice-Hall, 1973.

[12] R. Durrett. *Probability: Theory and Examples.* Duxbury Press, 2nd edition, 1996.

[13] R. Fonseca, V. Almeida, M. Crovella, and B. Abrahao. On the intrinsic locality properties of web reference streams. In *Proceedings of IEEE INFOCOM*, 2003.

[14] P. Haas and L. Stokes. Estimating the number of classes in a finite population. Technical Report RJ 10025, IBM Almaden Research Center, San Jose, CA, 1996.

[15] S. Jin and A. Bestavros. Popularity-aware greedydual-size algorithm for web access. In *Proceedings of the IEEE ICDCS Conference*, April 2000.

[16] S. Jin and A. Bestavros. Greedydual* web caching algorithm: Exploiting the two sources of temporal locality in web request streams. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 2000.

[17] F. P. Kelly. *Reversibility and Stochastic Networks.* John Wiley & Sons, 1979.

[18] D. Lee, J. Choi, S. Noh, S.L. Min, Y. Cho, and C.S. Kim. On the existence of a spectrum of policies that subsumes the lru and lfu policies. In *Proceedings of the ACM SIGMETRICS Conference*, May 1999.

[19] A. Mahanti, D. Eager, and C. Williamson. Temporal locality and its impact on web proxy cache performance. *Performance Evaluation*, September 2000.

[20] NLANR cache access logs. `ftp://ircache.nlanr.net/Traces/`, accessed January 2002.

[21] V. Padmanabhan and J. Mogul. Using predictive prefetching to improve world wide web latency. *ACM SIGCOMM Computer Communication Review*, 26(3), July 1996.

[22] K. Psounis and B. Prabhakar. Efficient randomized web-cache replacement schemes using samples from past eviciton-times. *IEEE/ACM Transactions on Networking*, 10(4), August 2002.

[23] M. Puterman. *Markov Decision Processes, Discrete Stochastic Dynamic Programming.* Wiley Series in Probability and Mathematical Statistics, 1994.

[24] B. Ramakrishna Rau. Properties and applications of the least-recently-used stack model. Technical Report CSL-TR-77-139, Stanford University, 1977.

[25] R. Rivest. On self-organizing sequential search heuristics. *Communications of the ACM*, 19(2):63–67, February 1976.

[26] S. Ross. *Introduction to Stochastic Dynamic Programming.* Academic Press, 1983.

[27] G. Shedler and C. Tung. Locality in page reference strings. *SIAM Journal of Computing*, 1(3), September 1976.

[28] A. Silberschatz and P. Galvin. *Operating System Concepts.* Addison Wesley Longman, Fifth Edition, 1997.

[29] D. Starobinski and D. Tse. Probabilistic methods for web caching. *Performance Evaluation*, 46(2–3):125–137, 2001.