Analysis and Design of Hierarchical Web Caching Systems

Hao Che, Zhijung Wang, and Ye Tung
Department of Electrical Engineering
Pennsylvania State University
State College, PA 16802

Abstract— **This paper aims at finding fundamental design principles for hierarchical web caching. An analytical modeling technique is developed to characterize an uncooperative two-level hierarchical caching system where the least recently used (LRU) algorithm is locally run at each cache. With this modeling technique, we are able to identify a characteristic time for each cache, which plays a fundamental role in understanding the caching processes. In particular, a cache can be viewed roughly as a lowpass filter with its cutoff frequency equal to the inverse of the characteristic time. Documents with access frequencies lower than this cutoff frequency will have good chances to pass through the cache without cache hits. This viewpoint enables us to take any branch of the cache tree as a tandem of lowpass filters at different cutoff frequencies, which further results in the finding of two fundamental design principles. Finally, to demonstrate how to use the principles to guide the caching algorithm design, we propose a cooperative hierarchical web caching architecture based on these principles. The simulation study shows that the proposed cooperative architecture results in 50% saving of the cache resource compared with the traditional uncooperative hierarchical caching architecture.**

Keywords— **Web caching, Hierarchical caching, Cache replacement algorithm**

I. Introduction

One of the important means to improve the performance of web service is to employ caching mechanisms. By caching web documents at proxy servers or servers close to end users, user requests can be fulfilled by fetching the requested document from a nearby web cache, instead of the original server, reducing the request response time, network bandwidth consumption, as well as server load. However, a cache miss causes long response time and extra processing overhead. Hence, a careful design of cache replacement algorithms which achieve high cache hit ratio is crucial for the success of caching mechanisms. In this paper, we explore the fundamental design principles associated with the cache replacement algorithm design when caches are arranged in a hierarchical structure.

Web caches need to be arranged intrinsically in a hierarchical structure due to the hierarchical nature of the Internet. An example is the four level cache hierarchy which matches with the hierarchical structure of the Internet [8], i.e., bottom, institutional, regional, and backbone. One of the key design issues for web caching is concerned with cache replacement algorithm design. Hence, it is of fundamental importance to explore the design principles of cache replacement algorithms for hierarchical caching.

Most of the research papers on cache replacement algorithm design, to date, have focused on a single cache, e.g., [5], [9], [2], [7], [13]. However, when caches are arranged in a hierarchical structure, running a cache replacement algorithm which is optimized for an isolated cache may not lead to overall good performance. Although results on the optimal hierarchical caching exists, e.g., [10], they are obtained based on the assumption that global caching information is known to every cache in the cache hierarchy, which is generally unavailable in practice. Moreover, most of the research papers heavily rely on the empirical performance comparisons for various cache replacement algorithms, such as the least recently used (LRU) algorithm, the least frequently used (LFU) algorithm, and the SIZE. Very little research effort has been made on the study of fundamental design principles. For example, the LFU algorithm based on a measurement time window for collecting the document access frequencies was proposed to reduce the table size for keeping document access frequencies. However, to the best of our knowledge, no design principles have ever been given as to how to properly select the time window size.

This paper aims at finding fundamental design principles in the context of hierarchical caching algorithm design. Some of the results of this study are also applicable to other caching architectures, such as distributed and hybrid caching architectures [11], [4].

There are three major contributions of this paper. First, unlike the previous analytic work on web caching which is based on statistic analysis, e.g. [1], [2], this paper proposes a stochastic model, which allows us to characterize the caching processes for individual documents in a two-level hierarchical caching system. In this model, the LRU algorithm runs at individual caches in an uncooperative manner. A mean field approximation is employed to solve the problem and the results are found to accurately match with the exact results within 2% error. The modeling technique enables us to study the caching performance for individual document requests under arbitrary access frequency distribution.The second contribution of this paper is the identification of a characteristic time associated with each cache and the finding of two design principles for hierarchical caching. The third contribution is the application of the design principles to the design of a cooperative hierarchical caching architecture. The proposed architecture is found to provide about 50% cache resource saving compared with the traditional uncooperative hierarchical caching architec-

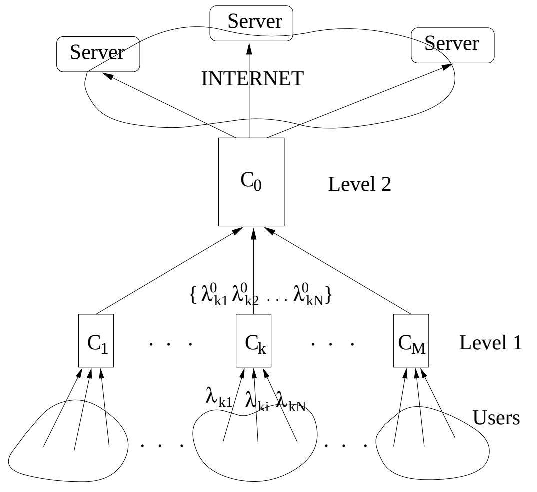

Server
Server
Server
INTERNET
C_0
Level 2
$\{\lambda^0_{k1}\lambda^0_{k2}\ldots\lambda^0_{kN}\}$
C_1
C_k
C_M
Level 1
λ_{k1} λ_{ki} λ_{kN}
Users

Fig. 1. Two level hierarchical web caching structure

ture.

The remainder of this paper is organized as follows. In Section 2, the model is described and analytical results derived. In Section 3, the performance results are presented and the design principles proposed. Based on the principles proposed in Section 3, Section 4 introduces a cooperative hierarchical caching architecture and the performance of the proposed architecture is compared with the traditional uncooperative one by simulation. Finally, conclusions and future work are given in Section 5.

II. A Hierarchical Web Caching Model

The traditional hierarchical web caching is to build web caches into a tree structure with the leaf nodes corresponding to the lowest caches closest to the end users and the root nodes the highest caches. User requests travel from a given leaf node towards the root node, until the requested document is found. If the requested document cannot be found even at the root level, the request is redirected to the web server containing the document. The requested document is then sent back via the reversed path, leaving a copy of the requested document in each intermediate cache it traverses. The hierarchical caching is called uncooperative hierarchical caching if caching decisions are made locally at each cache throughout the cache hierarchy. In the following subsections, we propose a modeling technique to characterize the caching processes for a traditional uncooperative hierarchical caching architecture.

A. Model Description and Notations: Consider a two-level web cache hierarchy with a single cache of size C_0 at the root level and M caches of sizes C_k $(k = 1, 2, ..., M)$ at the leaf level. The network architecture of our model is presented in Fig. 1. We study uncooperative caching with the LRU algorithm locally running for any caches in the cache hierarchy.The following three assumptions are made:
1. The aggregate request arrival process at leaf cache k $(k = 1, 2, ..., M)$ is Poisson with mean arrival rate λ_k.
2. The arrivals of the request for individual document i $(i = 1, 2, ..., N)$ at cache k is independently sampled from the aggregate arrival process based on the probability set $\{p_{ki}\}$, where p_{ki} is the access probability for document i at cache k and $\sum_{i=1}^N p_{ki} = 1$. Here N is the document sample space size.
3. All the documents have the same size.
Assumption 1 and 2 implies that the request arrival process for document i $(i = 1, 2, ..., N)$ is Poisson with mean arrival rate λ_{ki} [15], where

$$\lambda_{ki} = p_{ki}\lambda_k. \tag{1}$$

Assumption 3 is a pretty strong one but it will not affect the correctness of the qualitative results, as we shall explain later. With this assumption, cache sizes C_i $(i = 0, 1, 2, ..., M)$ are measured in the unit of document size.

Since Zipf-like distribution has been widely tested against the distributions collected from the real traces, e.g. [9], [2], [6], we model p_{ki} by Zipf-like distributions,

$$p_{ki} = K_k \frac{1}{R_k(i)^{z_k}}, \quad \text{for } k = 1, ..., M, \text{ and } i = 1, ..., N, \tag{2}$$

where K_k is the normalization factor, $R_k(i)$ is the popularity rank of document i at cache k, and z_k is a parameter taking values in $[0.6, 1]$.

Cache miss ratios are widely used as performance measures of cache replacement algorithms. Let the average arrival rate of document i from leaf cache k $(k = 1, 2, ..., M)$ to the root cache be λ^0_{ki} and the average miss rate of document i at the root cache be λ_{0i}, as shown in Fig. 1. Note that λ^0_{ki} is simply the average cache miss rate of document i at cache k. Then the following cache miss ratios can be defined in terms of λ_{ki}, λ^0_{ki} and λ_{0i} as,

$$\begin{aligned} \eta_{ki} &= \frac{\lambda^0_{ki}}{\lambda_{ki}}, \\ \eta_i &= \frac{\lambda^0_i}{\sum_{k=1}^M \lambda_{ki}}, \\ \eta^0_k &= \frac{\sum_{i=1}^N \lambda^0_{ki}}{\sum_{i=1}^N \lambda_{ki}}, \\ \eta &= \frac{\sum_{i=1}^N \lambda_{0i}}{\sum_{k=1}^M \sum_{i=1}^N \lambda_{ki}}, \end{aligned} \tag{3}$$

where η_{ki} is the cache miss ratio of document i at cache k, η_i is the cache miss ratio of document i for the whole hierarchical caching system, η^0_k is the total cache miss ratio at cache k, and η is the total cache miss ratio for the whole hierarchical caching system. As we shall see that λ^0_{ki} and λ_{0i} themselves are, in fact, more insightful performance measures than the cache miss ratios defined above.

B. Model Analysis: Now, the focal point is to derive λ^0_{ki} and λ_{0i}, or equivalently, the average miss intervals $T_{ki} = {\lambda^0_{ki}}^{-1}$ and $T_{0i} = \lambda_{0i}^{-1}$ for document i $(i = 1, 2, ..., N)$ at leaf cache k and the root cache, respectively.

Calculation of T_{ki}: To find T_{ki} for document i, one notes that the inter-arrival time between two successive cache misses for document i at cache k is composed of a sequence of i.i.d. (i.e., independent identically distributed) random variables $\{t_1, t_2, ..., t_{n-1}\}$ plus an independent random variable t_n. Each epoch t_i $(i = 1, 2, ..., n-1)$ corresponds to a period between two successive cache hit of document i. The last epoch t_n is the time interval between the last cache hit and the next cache miss. The process is shown in Fig. 2. Let's take a look at the first epoch t_1. The cache miss at the beginning of the epoch results in the caching of the requested document to the head of the LRU list. Here we neglect the delay between cache miss and document caching. As time goes, the cached document moves towards the tail of the list until there is a hit of the document at the end of the epoch, when the document is moved back to the head of the list. The movement of the document towards the tail is due to the caching or hits of other documents, which according to the LRU algorithm, will cause the caching or moving of those documents to the head of the list.

Let's first calculate the distribution density function $f^0_{ki}(t)$ for the cache miss interval t at leaf cache k or the request interval of document i from cache k to the root cache. We have

$$t = \sum_{i=1}^{n-1} t_i + t_n. \tag{4}$$

The exact distribution of an epoch t_i can be formally written in terms of the distributions of the constituent Poisson processes for individual document requests. However, the computation complexity is extremely high even when C_k and N are moderately small. In what follows, we propose a "mean field" approximation to make the problem trackable.

Mean field approximations were successfully employed in solving many-object problems in complex systems such as many-body systems in statistical physics [12] and loss networks [14]. The basic idea is to take the interaction of all other objects with the designated object as a mean field and the designated object interacts with this field instead of the other objects. The mean field approximation approaches exact solution asymptotically as the number of objects goes to infinity.

In applying the mean field approximation concept to the problem at hand, the idea is to summarize the effect of the other $N-1$ constituent Poisson processes as a mean field τ_{ki} acting upon the designated document request i. Here τ_{ki} is defined as the maximum inter-arrival time between two adjacent requests for document i without a cache miss as shown in Fig. 2. In essence, τ_{ki} is a random variable. However, in our mean field approximation, τ_{ki} is assumed to be a constant. Our simulation studies showed that the variance of τ_{ki} is very small even for small N, say, $N = 10,000$ and τ_{ki} is very insensitive to i. With the mean field approximation, the problem is greatly simplified because the interaction between the caching process of document i and all other processes is mediated by τ_{ki} only through the conditions: $t_j \le \tau_{ki}$ $(j = 1, 2, ..., n-1)$ and $t_n > \tau_{ki}$. τ_{ki}

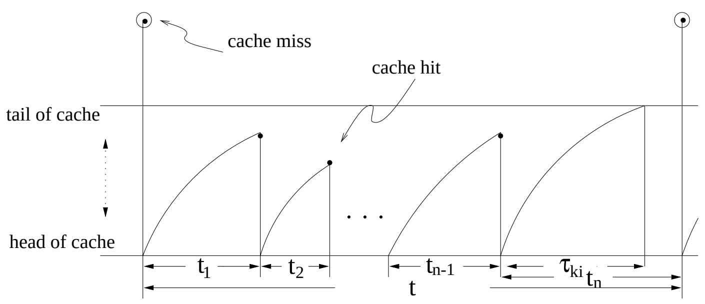


Fig. 2. Arrival processes of a given document at both level caches

can be easily calculated by solving the following equation:

$$\sum_{j=1, j\neq i}^{N} P_{kj}(t < \tau_{ki}) = C_k \tag{5}$$

where $P_{ki}(t < \tau)$ is the cumulative distribution of the request interarrival time for document i at leaf cache k. This equation simply states that within τ_{ki} time units since the last cache hit of document i, there are exactly C_k distinct documents being hit or cached, given that there is no more request for document i during this period of time. Since the cache hit time for document i can occur at anytime, (5) does not hold in general except for the current case where the individual processes are Poisson processes. A general expression is given in (16).

Now, $f^0_{ki}(t)$ can be formally expressed as follows,

$$f^0_{ki}(t) = \sum_{n=1}^{\infty} f_{ki}(t|n) P_{ki}(t < \tau_{ki})^{n-1} (1 - P_{ki}(t < \tau_{ki})), \tag{6}$$

where

$$P_{ki}(t < \tau_{ki}) = 1 - e^{-\lambda_{ki}\tau_{ki}}. \tag{7}$$

Next, take the Laplace transform of $f^0_{ki}(t)$. We have (See Appendix A),

$$\phi_{ki}(s) = \frac{\lambda_{ki} e^{(-s-\lambda_{ki})\tau_{ki}}}{\lambda_{ki} e^{(-s-\lambda_{ki})\tau_{ki}} + s} \tag{8}$$

Then, T_{ki} is readily obtained as,

$$T_{ki} = -\frac{d\phi_{ki}(s)}{ds}|_{s=0} = \lambda_{ki}^{-1} e^{\lambda_{ki}\tau_{ki}} \tag{9}$$

and

$$\lambda^0_{ki} = T_{ki}^{-1} = \lambda_{ki} e^{-\lambda_{ki}\tau_{ki}} \tag{10}$$

Calculation of T_{0i}: With the mean field approximation, one takes away the complex correlation among the request arrival processes for different documents to the root cache. This makes the calculation of T_{0i} possible. The key is to: (1) construct the distribution function $f^0_{ki}(t)$ or the corresponding cumulative distribution function $P^0_{ki}(t < \tau)$; (2) construct the aggregate cumulative distribution function

$P_{0i}(t < \tau)$ for the interarrival time of document i to the root cache; (3) again, apply the mean field approximation to obtain T_{0i}.

To construct $f^0_{ki}(t)$, one needs to find the inverse transform of $\phi_{ki}(s)$ in (8). However, there is no compact solution for $f^0_{ki}(t)$. In Appendix B, we derived an exact solution with infinitely many terms as follows,

$$f^0_{ki}(t) = \sum_{n=1}^{\infty} (-1)^{n+1} \lambda_{ki}^{-1} \frac{(t - n\tau_{ki})^{n-1}}{(n-1)!} u(t - n\tau_{ki}) \tag{11}$$

where $u(t)$ is a step function with $u(t) = 0$ when $t < 0$ and $u(t) = 1$, otherwise. Note that $f^0_{ki}(t) = 0$ when $t < \tau_{ki}$, a consequence of the mean field approximation. With alternate sign changes between any two successive terms and with factorial decaying factors, this series converges very fast. One also note that due to the step function in each term, for any finite t, $f^0_{ki}(t)$ is exactly described by finitely many terms. Despite all these nice features of the above expression, we still find it cumbersome when explicit expressions for T_{0i} is to be sought. Hence, instead of using (11), we use the following approximate expression for further development,

$$f^0_{ki}(t) \approx \begin{cases} \sigma^0_{ki} e^{-\sigma^0_{ki}(t - \tau_{ki})} & \tau_{ki} < t < \infty \\ 0 & t < \tau_{ki} \end{cases} \tag{12}$$

This is a truncated exponential distribution with

$$\sigma_{ki} = \frac{1}{\lambda_{ki}^{-1} - \tau_{ki}}. \tag{13}$$

Here σ_{ki} is chosen in such a way that T_{ki} derived from (12) gives the exact result in (9). Numerical studies showed (not presented in this paper) that the expression in (12) closely matches with the exact solution in (11). From (12), we have,

$$P^0_{ki}(t < \tau) \approx \begin{cases} 1 - e^{-\sigma^0_{ki}(\tau - \tau_{ki})} & \tau_{ki} < \tau < \infty \\ 0 & \tau < \tau_{ki} \end{cases} \tag{14}$$

From Theorem 10.4.5 in [16], it is easy to show that, in general,

$$\begin{aligned} P_{0i}(t < \tau) \quad = \quad & 1 - \textstyle\sum_{i=1}^{N} \lambda^0_{ki}{}^{-1} \sum_{k=1}^{M} \lambda^0_{ki} \\ & (1 - P^0_{ki}(t < \tau)) \textstyle\prod_{k=1, k \neq k'}^{M} \\ & \textstyle\int_{\tau}^{\infty} \lambda_{k'i}(1 - P^0_{k'i}(t < x))dx \end{aligned} \tag{15}$$

In parallel to the mean field approximation at the leaf cache, the mean field approximation is also employed at the root cache. Define τ_{0i} as the maximum inter-arrival time of the two adjacent requests for document i at the root cache without a cache miss. The average τ_{0i} can be solved from (again, applying Theorem 10.4.5 in [16]),

$$\sum_{i=1, i \neq j}^{N} F_i(t < \tau_{0j}) = C_0 \tag{16}$$

where

$$F_i(t < \tau_{0j}) = 1 - (\sum_{k=1}^{M} \lambda^0_{ki})^{-1} \int_0^{\tau_{0j}} (1 - P_{0i}(t < \tau))d\tau. \tag{17}$$

Finally, with reference to Fig. 2, one can calculate T_{0i} as,

$$\begin{aligned} T_{0i} \quad = \quad & \textstyle\sum_{n=0}^{\infty} [(n-1)\bar{t}_{0i}|_{t < \tau_{0i}} + \bar{t}_{0i}|_{t > \tau_{0i}}] \\ & P_{0i}(t < \tau_{0i})^{n-1}(1 - P_{0i}(t < \tau_{0i})), \end{aligned} \tag{18}$$

or

$$T_{0i} = \bar{t}_{0i}|_{t < \tau_{0i}} \frac{P_{0i}(t < \tau_{0i})}{1 - P_{0i}(t < \tau_{0i})} + \bar{t}_{0i}|_{t > \tau_{0i}}, \tag{19}$$

$\bar{t}_{0i}|_{t < \tau_{0i}}$ ($\bar{t}_{0i}|_{t > \tau_{0i}}$) is the average epoch duration provided that the duration is smaller (larger) than τ_{0i}.

The analytical expression for T_{0i} in (19) can be derived based on (15). However, due to the peculiar dependency of $P^0_{ki}(t < \tau)$ with respect to τ_{ki} in (14), the general expression is lengthy and cumbersome. In this paper, we consider a special case. We assume that λ_k, C_k, and z_k in (2) are the same for all k ($k = 1, 2, ..., M$). Also assume that $\tau_{ki} = \tau_k$, i.e., τ_{ki} is independent of i (a highly accurate approximation). Then $\tau_k = \tau_1$ for $k = 2, 3, ..., M$. This is true simply because τ_k is completely determined by λ_k, C_k, and z_k. Substituting (14) into (15), we then have,

$$P_{0i}(t < \tau) = \begin{cases} 1 - \sum_{k=1,k'=1}^{M} \sigma_{ki} \lambda^{0-1}_{k'i} & \\ \prod_{k=1}^{M} \lambda_{ki} \sigma_{ki}^{-1} e^{-\sigma_{ki}(\tau - \tau_1)} & \text{for } \tau > \tau_1 \\ 1 - \sum_{k=1,k'=1}^{M} \lambda^0_{k'i}(\tau_1 - \tau + \sigma_{ki}^{-1})^{-1} & \\ \prod_{k=1}^{M} \lambda^0_{ki}(\tau_1 - \tau + \sigma_{ki}^{-1}) & \text{for } \tau < \tau_1 \end{cases} \tag{20}$$

With a straightforward but tedious calculation based on (20), we arrived at the following expressions,

$$\begin{aligned} \bar{t}_{0i}|_{t < \tau_{0i}} \quad = \quad & \tau_1 - P_{0i}(t < \tau_{0i})^{-1}[\tau_{oi} \\ & +\left(\textstyle\sum_{k=1}^{M} \lambda^0_{ki}\right)^{-1} \textstyle\sum_{k=1}^{M} \int_0^{\tau_1} \lambda_{ki} \prod_{k'=1,k' \neq k}^{M} \\ & (\tau - \tau_1 - \sigma_{ki}^{-1})d\tau + \textstyle\prod_{k=1}^{M} \lambda^0_{ki} \sigma_{ki}^{-1} \\ & \left(\textstyle\sum_{k=1}^{M} \lambda^0_{ki}\right)^{-1} (e^{-\sum_{k=1}^{M} \sigma_{ki}(\tau_{0i} - \tau_1)} - 1)] \end{aligned} \tag{21}$$

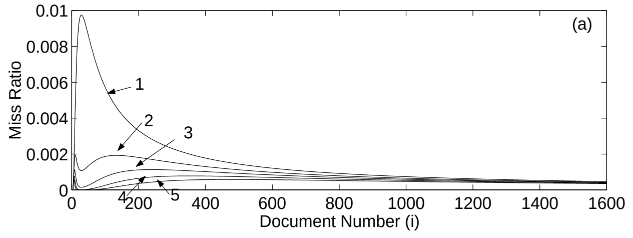
and

$$\begin{aligned} \bar{t}_{0i}|_{t > \tau_{0i}} \quad = \quad & \tau_{0i} - (1 - P_{0i}(t < \tau_{0i}))^{-1} \\ & \textstyle\prod_{k=1}^{M} \lambda^0_{ki} \sigma_{ki}^{-1} \left(\sum_{k=1}^{M} \lambda^0_{ki}\right)^{-1} \\ & e^{-\sum_{k=1}^{M} \sigma_{ki}(\tau_{0i} - \tau_1)} \end{aligned} \tag{22}$$

T_{0i} is obtained by substituting (21), (22), and (20) into (19). This solution is tested against simulation results, which shows that the solution is highly accurate with a maximum error less than 2%.

III. Performance Analysis and Design Principles

A. Performance Analysis: In this subsection, we present the numerical results for the two-level hierarchical caching model proposed in the previous section. Unlike most of the existing papers on web cache replacement



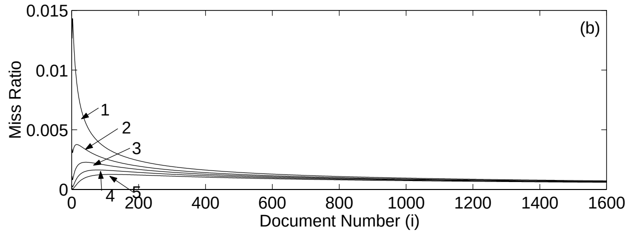
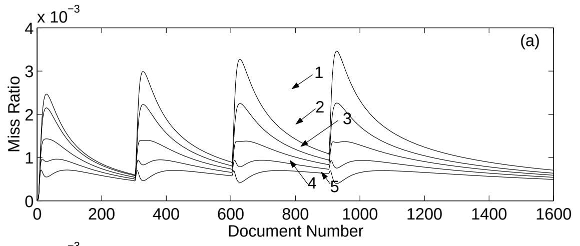


Fig. 3. Cache miss ratios for individual documents– homogeneous case. (a) $z_k = 1$, (b) $z_k = 0.6$. curve 1: cache miss ratio at leaf level cache, curves 2,3,4,5: cache miss ratios at the root cache for $C_0 = 800, 1200, 1600, 2000$



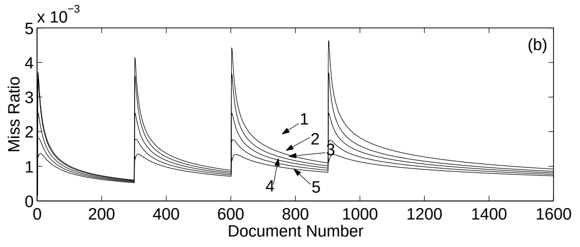


Fig. 4. Cache miss ratios for individual documents–inhomogeneous case. (a) $z_k = 1$, (b) $z_k = 0.6$. curve 1: cache miss ratio at leaf level cache, curves 2,3,4,5: cache miss ratios at the root cache for $C_0 = 800, 1200, 1600, 2000$

which focus on the analysis of aggregate cache hit/miss performance, our study focuses on the performance analysis of cache hit/miss performance for individual documents. This study is important because it provides much better insight on how the cache replacement algorithm should be designed to optimize the overall performance.

For all the case studies in this paper, we set $M = 4$, $N = 20,000$, $\lambda_k = 2$ and $C_k = 200$ for $k = 1, 2, 3, 4$. We study two extreme cases of Zipf-like distributions, i.e., $z_k = 0.6$ and 1 for $k = 1, 2, 3, 4$. We also consider both *homogeneous* and *inhomogeneous* cases. Here homogeneous refers to $p_{ki} = p_{k'i}$ for $\forall\ k, k' = 1, 2, 3, 4$ and inhomogeneous, otherwise. For the inhomogeneous case, the document popularity rank $R_k(i)$ is shifted by 300 documents from one leaf cache to another. In other words, $R_k(i) = [N + i - 300 * (k - 1)]\%N + 1$, for $k = 1, 2, 3, 4$. Now, we present the numerical results for the miss rates $\lambda_i^0 = \sum_{k=1}^4 \lambda_{ik}^0 = \sum_{k=1}^4 T_{ki}^{-1}$ and $\lambda_{0i} = T_{0i}^{-1}$ at the two level caches. Four cases of λ_{0i} are studied corresponding to $C_0 = 800, 1,200, 1,600, 2,000$, respectively. Given that $N = 20,000$, $C_0 = 2,000$ should be considered sufficiently large.

We first focus on the homogeneous case. Fig. 3 presents the miss rates at both level caches versus document number or rank (up to 1600). Subplot (a) and (b) correspond to $z_k = 1$ and 0.6, respectively. In each subplot, curve 1 represents λ_i^0. Curves 2,3,4, and 5 correspond to λ_{0i} for $C_0 = 800, 1,200, 1,600, 2,000$, respectively. As expected, the cache miss rates at both level caches in subplot (b) are higher than that in subplot (a). Both subplots show that increasing root cache size C_0 helps to reduce the miss rates for the first 1000 popular documents but it helps very little to reduce the cache miss rates for the rest of the documents. Also note that after certain size, say, 1,200, further increasing C_0 does not significantly improve cache miss performance, especially for the case in subplot (b) where unpopular documents constitute a large portion of the overall misses.

Interesting enough, both subplots show that the curves are peaked at documents of certain ranks and then the curves drop exponentially. Besides, curves in subplot (b) drop more sharply than that in subplot (a). To explain this phenomena, we note that the analytical expression for the aggregate miss rate at the leaf caches is available. From (10), we have,

$$\lambda_i^0 = \sum_{k=1}^{4} \lambda_{ki}^0 = 4\lambda_{1i} e^{-\lambda_{1i}\tau_1}. \tag{23}$$

Here we have taken $\tau_{ki} = \tau_1$. (23) explains the exponential behaviors for the cache miss rates at the leaf caches. Since τ_1 for $z_1 = 1$ is larger than that for $z_1 = 0.6$, it explains why $z_1 = 0.6$ has a faster exponential dropping rate. λ_i^0 is maximized when $\lambda_{1i} = \tau_1{}^{-1}$ and it is equal to $4e^{-1}\tau_1{}^{-1}$. Since $\tau_1 = 151$ and 102.6 at $z_1 = 1$ and 0.6, respectively, it is easy to verify that the peaks of λ_i^0 in subplot (a) and (b) do occur at these values. Temptated by this observation, we further use $e^{-1}\tau_{0i}^{-1}$ to estimate the peak miss rates at the root cache for curves 2,3,4, and 5. Amazingly, it turns out that the calculated peak rates match with the actual peak rates almost perfectly.

Next, we further examine the above phenomena for the inhomogeneous case. Fig. 4 depicts the results for the inhomogeneous case with the settings and the other parameters the same as the homogeneous one. In the inhomogeneous case, four miss rate peaks are identified due to the shifted document popularities from one leaf cache to another. Again, the miss rate at the root cache is almost leveled at $C_0 = 2,000$ for both cases, indicating that there will be little performance gain by further increasing C_0. Again, we use $e^{-1}\tau_{0i}^{-1}$ to estimate the peak miss rates at the root cache and find they are within 15% differences from the actual peak values in Fig 4.

The above studies indicate that there is a cutoff frequency at $e^{-1}\tau^{-1} \approx 0.368\tau^{-1}$, where τ is defined as the average maximum document access interval without a cache miss. Here we identify τ as a *characteristic time* for a given cache and it is a function of the request processes, the cache size, as well as the request pattern. The miss rate or miss frequency λ_i of any given document i with respect this cache will be very likely to be smaller than this cutoff frequency. More conservatively, one can set the cutoff frequency at τ^{-1}, which guarantees that no document miss frequency can exceed this cutoff frequency. In fact, only document requests with constant interarrival time slightly larger than τ will have a miss rate close to τ^{-1}. To see why τ^{-1} can be viewed as a cutoff frequency more clearly. We calculate the miss ratio η_{ki} in (3). From (10), we have,

$$\eta_{ki} = e^{-\lambda_{ki}\tau_k}. \tag{24}$$

One observes that the cache miss ratio for document i at cache k quickly approaches 1 as λ_{ki} falls below τ_k^{-1}.

B. Design Principles: The performance analyses in the previous subsection immediately lead to the following conclusions. A cache can be viewed conceptually as a lowpass filter with a cutoff frequency upper bounded by τ^{-1}, where τ is the characteristic time for the cache. Requests of a document with access frequency lower than τ^{-1} will have good chances to pass through the cache, causing cache misses. With respect to this cache, all the documents with access frequencies much lower than τ^{-1}, say, smaller than $e^{-1}\tau^{-1}$, are *effectively* one-time access documents and will surely pass through the cache without a hit.

The above conceptual view is particularly helpful when it is used to identify design principles for a hierarchical cache structure. Consider a branch of an L-level hierarchical cache tree from a given leaf cache at level 1 to the root cache at level L. Denote the characteristic time for the ith level cache as τ_i $(i = 1, 2, ..., L)$. Then if we view these caches as a tandem of lowpass filters with cutoff frequencies τ_i^{-1} $(i = 1, 2, ..., L)$, we can immediately identify two design principles:

1. For higher level cache l to be effective with respect to a lower level cache l', we must have, $\tau_l > \tau_{l'}$ for $l > l', l, l' = 1, 2, ..., L$. Since τ_l is directly related to the cache size (see (5) and (16)), these conditions can be readily used for cache dimensioning.
2. Given the conditions in 1. hold, a document with access frequency lower than $e^{-1}\tau_L^{-1}$ is effectively an one-time access document with respect to the entire cache hierarchy. This document should not be cached in any caches throughout the cache hierarchy.

The assertion in Principle 2 can be readily verified from Fig. 3 and Fig. 4. Note that for all the case studies, $\tau_{0i}^{-1} > 0.001$. One observes that the tail portions of the cache miss curves at both cache levels converges together, meaning that the requests corresponding to the tail portions pass through the cache hierarchy without a hit. They are effectively one-time access documents with access frequencies smaller than 0.001.

Here a comment is in order. Although the above principles are drawn from the analytical results under the assumption that document sizes are the same, the principles generally hold when document sizes are different. This is because the principles are derived from the concept of a characteristic time which exists regardless of whether documents have the same size or not.

IV. A Cooperative Hierarchical Caching Architecture

To demonstrate the power of the design principles obtained in the previous section, we propose in this subsection a cooperative hierarchical caching architecture based on these design principles.

A. Architecture Overview: Our cooperative hierarchical caching architecture can be described as follows: Like the traditional uncooperative hierarchical caching architecture, such as Harvest [3], the LRU algorithm is used locally at individual caches throughout the cache hierarchy and a request is search upward starting from the lowest level cache.

However, our architecture involves two major changes to the traditional architecture. First, in our architecture, a cache hit of document i at level l ($l = 2, 3, ..., L, L+1$. Here $L+1$ refers to the original server) does not result in document caching in all the lower level caches along the request path. Instead, the document will be cached only in level m cache if $m < l$, where $\tau_m^{-1} < \lambda_{ki} < \tau_{m-1}^{-1}$. Here λ_{ki} is the access frequency for document i at the leaf cache k. This is based on the observation that document i has a good chance to pass through all the caches at levels lower than m. Caching document i in those caches is largely a waste of cache resources. This change to the traditional architecture ensures that when the access frequency of document i increases, it will have better chance to be found in a cache closer to the user (i.e., m instead of l). However, it does not take care of the situation when the access frequency of document i decreases. The second change takes care of this situation.

The second change to the traditional architecture is the following. A replaced document from level m cache is further cached at its upper level cache (i.e., level $m+1$ cache) if the document is not in that cache. Otherwise the document is considered to be the most recently used and is moved to the head of the list in level $m+1$th cache. In this way, a document cached at level m cache will have an effective minimum life-time equal to $\sum_{k=m}^{L} \tau_k$. This ensures that subsequent accesses of the document which incur misses at level m will still have good chance to have cache hits at higher level caches, although with possibly longer response time.

Note that the proposed architecture requires that leaf cache k estimates λ_{ki} $(i = 1, 2, ..., N)$ and each level cache estimates its own characteristic time. Also note that the proposed architecture is cooperative in the sense that estimation of λ_{ki} requires the knowledge of the characteristic time of the root level cache, as we shall see shortly. A final note is that caching based on the measured τ_l $(l = 1, 2, ..., L)$ lends us a natural adaptive caching mechanism which takes into account of the cache size, request arrival rate, as well as request pattern. In the following subsection, a measurement scheme is proposed to enable the estimate of τ_l and λ_{ki}.

B. Measurement Schemes: Characteristic Time: In our architecture, the characteristic time τ_l at any cache l in the cache hierarchy needs to be measured periodically. This can be easily done by inserting a timestamp in each cached document and the timestamp is updated to the current time whenever the document receives a hit. The instantaneous τ_l value can then be obtained whenever a document is being replaced, simply by taking the difference between the time of replacement and the timestamp of the document.

Note that the computation complexity for updating a timestamp is much lower than the computation complexity for the matching of the request with the cached document and for the shuffling of the document who gets a hit to the head of the list (this involves the exchange of six pointers). When the request process is quite stationary, computation complexity can be reduced by updating τ_l at relatively large time intervals, say, every 5 minutes. Adaptive algorithms can also be designed to fine tune the measurement intervals. For our simulation study based on time invariant Poisson arrival processes, τ_l is found to fluctuate very little around its mean value. In practice, to reduced the effect of possible large fluctuations, τ_l can be updated as follows,

$$\tau_l(n\Delta t) = (1-\alpha)\tau_l((n-1)\Delta t) + \alpha\tau_l' \qquad (25)$$

where Δt is the update interval, τ_l' is the instantaneous measured characteristic time, and α is a parameter taking values between 0 and 1. To reduce the effect of fluctuation, a relatively small α value can be chosen.

Access frequency: In our architecture, the access frequency of each document needs to be estimated at leaf caches for the purpose of making caching decisions at any cache in the cache hierarchy. In this sense, our approach is like a combination of the LRU algorithm and the LFU algorithm, except that here the LFU algorithm is not used for a single cache but for all the caches in the cache hierarchy.

A fundamental difficulty in using LFU algorithm is the need to keep track of the access frequencies for all the documents that have been requested. A practical solution is to keep track of the access frequencies for documents which have been seen in the past ΔT time window. However, an open issue is how to set ΔT value. On the basis of our design principles, we can readily solve this problem. According to Principle 1, a document with access frequency lower than $e^{-1}\tau_L^{-1}$ is effectively an one-time access document and it makes no sense to keep track of the access frequencies of any documents with access interval much larger than τ_L. On the other hand, documents with access interval smaller than τ_L are worth caching and hence should be tracked. Therefore ΔT should be set at $\Delta T \approx \tau_L$. This requires that the root cache periodically broadcasts its measured τ_L value to all the leaf caches. This can be done by piggybacking τ_L to the requested documents sending back from the root cache to the leaf cache.

In our implementation, a leaf cache keeps a table of access frequencies for individual documents. A document is deleted from the table if the elapsed time since the last access exceeds ΔT. To further reduce the computation complexity, in our implementation, only a timestamp of the last access time is kept and updated for each document.

The algorithm works as follows. Upon the arrival of a request at leaf cache k, the cache is searched first. If there is a cache hit, the document is sent back to the requesting user. Otherwise the table is searched for a match. If a match is found, the difference between the request time and the timestamp in the table entry is calculated. The inverse of the difference is then used as the access frequency λ_{ki}, which is then attached to the request and sent to the higher level cache for further search. If a match is not found, an entry is then allocated to this request and λ_{ki} is set to zero before it is attached to the request which is sent to the higher level cache. The number of table entries can be further reduced by deleting the entries with the corresponding documents being cached in the leaf cache. This will not affect the performance because a document will stay in the leaf cache for at least τ_k time units and the next request time of the same document at the table will be longer than τ_k.

C. Performance Evaluation: A comparative performance analysis is performed for the proposed hierarchical caching architecture with the uncooperative caching architecture described at the beginning of Section 2. Since the focal point of this study is to demonstrate the effectiveness of the design principles, the parameters used in this study are not fine tuned to achieve optimal performance, which is subject to future study.

Again, consider a two-level hierarchical caching system with $M = 4$ leaf caches and one root cache. We set $\Delta T = 1.2\tau_0$, where τ_0 is the characteristic time for the root cache. With all the other parameter settings the same as the ones used in the previous section and $z_k = 1$, the cache miss rates at the root level cache are calculated for the proposed architecture at $C_0 = 1,200$ based on simulation and the uncooperative architecture at $C_0 = 1,200$ and $2,400$ based on numerical analysis.

Fig. 5 presents the results. The subplots (a) and (b) give the results for the homogeneous and inhomogeneous cases, respectively. For both cases, the overall cache miss ratio η, as defined in (3), for the proposed architecture at $C_0 = 1,200$ are found to be very close to the cache miss ratio for the uncooperative architecture at $C_0 = 2,400$. This means the proposed architecture can offer comparable performance as the uncooperative architecture with about 50% saving of the root cache resource. More importantly, from Fig. 5, we see that the proposed architecture successfully reduces the miss rates for the popular documents. Comparing the two curves at $C_0 = 1,200$ for both subplots and focusing on the first 1300 popular documents, one observes that on average several times of performance gain is achieved by using our proposed architecture. The proposed architecture leads to a small loss of performance

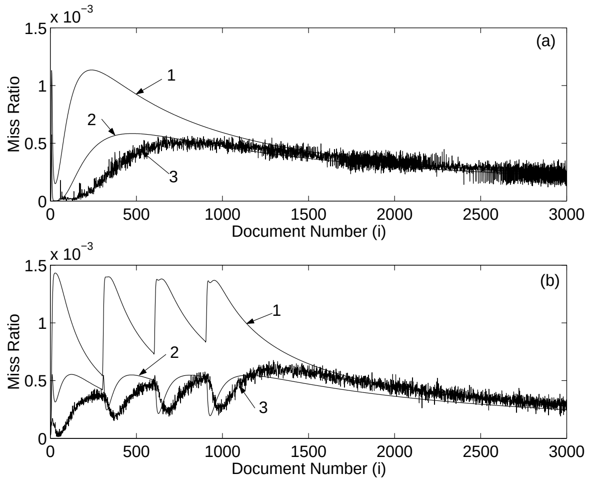


Fig. 5. Cache miss ratios for individual documents at the root cache for cooperative (curve 3) and uncooperative (curves 1 and 2) architectures. (a) homogeneous case, (b) inhomogeneous case. Curves 1 and 3: cache miss ratios at $C_0 = 1200$, and curve 2: cache miss ratio at $C_0 = 2400$

for unpopular documents at the tail portions of the curves due to the exclusion of these documents from being cached. However, this loss of performance is well compensated by the performance gain for the popular documents.

In essence, with the removal of the unpopular documents from being cached, we, in effect, increase the characteristic times at both level caches, resulting in tremendous overall performance gain. For instance, at $C_0 = 1,200$, $\tau_0 = 332.4$ for uncooperative case and $\tau_0 = 1875$ for cooperative case, increasing by about six times.

V. Conclusions and Future Work

In this paper, a modeling technique is proposed to analyze the caching performance of a two-level uncooperative hierarchical web caching architecture. On the basis of this analysis, an important characteristic time is identified for a cache, which is a function of request arrival processes, the cache size, as well as the request pattern. Two hierarchical caching design principles are identified based on the concept of filtering defined by the characteristic time. The design principles is then used to guide the design of a cooperative hierarchical caching architecture. The performance of this architecture is found to be superior to the traditional uncooperative hierarchical caching architecture.

We believe that the characteristic time and the associated filtering concept can be easily generalized to apply to a cache which runs some other cache replacement algorithm, such as LRU-size or LFU. Also, on the basis of the characteristic time and the filtering concept, design principles can be developed to guide the design of high performance hybrid or distributed caching architectures. We shall look into these issues in the future.

Appendix A

We have,

$$\begin{aligned}\phi_{ki}(s) &= \textstyle\sum_{n=1}^{\infty} P_{ki}(n) \int e^{-st} f_{ki}(t|n)dt \\ &= \textstyle\sum_{n=1}^{\infty} [(1-q_{ki})M_1(s)]^{n-1} q_{ki} M_2(s) \\ &= \frac{q_{ki}M_2(s)}{1-(1-q_{ki})M_1(s)},\end{aligned} \tag{26}$$

where

$$\begin{aligned}M_1(s) &= \int_0^{\tau_{ki}} e^{-st} \frac{\lambda_{ki} e^{-\lambda_{ki} t}}{1-e^{-\lambda_{ki}\tau_{ki}}} dt \\ &= \frac{\lambda_{ki}(e^{(-s-\lambda_{ki})\tau_{ki}}-1)}{(-s-\lambda_{ki})(1-e^{-\lambda_{ki}\tau_{ki}})}\end{aligned} \tag{27}$$

and

$$\begin{aligned}M_2(s) &= \int_{\tau_{ki}}^{\infty} e^{-st} \frac{\lambda_{ki} e^{-\lambda_{ki} t}}{e^{-\lambda_{ki}\tau_{ki}}} dt \\ &= \frac{\lambda_{ki} e^{(-s-\lambda_{ki})\tau_{ki}}}{(\lambda_{ki}+s)e^{-\lambda_{ki}\tau_{ki}}}\end{aligned} \tag{28}$$

Substituting (27) and (28) into (26), we get (8).

Appendix B

From (8) and (10), we have,

$$\begin{aligned}\phi_{ki}(s) &= \frac{1}{1+(\lambda_{ki}^0 s)^{-1} e^{-s\tau_{ki}}} \\ &= \textstyle\sum_{n=1}^{\infty} (-1)^{n+1} (\lambda_{ki}^0)^{-n} s^{-n} e^{-ns\tau_{ki}}\end{aligned} \tag{29}$$

Making use of the inverse Laplace transform,

$$\frac{e^{ks}}{s^{\mu}} \Leftrightarrow \frac{(t-k)^{\mu-1}}{\eta(\mu)} u(t-k), \qquad \mu > 0, \tag{30}$$

the inverse Laplace transform of $\phi_{ki}(s)$ gives $f_{ki}(t)$ in (11).

References

[1] K. Ross, "Hash Routing for Collections of Shared Web Caches," *IEEE Network Mag.*, pp. 37, Nov. 1997.

[2] J. Zhang, R. Izmailov, D. Reininger, and M. Ott, "Web Caching Framework: Analytical Models and Beyond," *IEEE Workshop on Internet Applications 1999*, pp. 132-141, 1999.

[3] R. P. Wooster and M. Abrams, "Proxy Caching that Estimates Page Load Delays," http://vtopus.cs.vt.edu/∼ chitra/docs/96trNEW/

[4] S. G. Dykes, C. L. Jeffery, and S. Das, "Taxonomy and Design Analysis for Distributed Web Caching," *Proceedings of HICSS-32*, pp. 10, 1999.

[5] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, pp. 193-206, Dec 1997.

[6] C. R. Cunha, A. Bestavros, and M. E. Crovella, "Characteristics of WWW Client-based Traces," Technical Report BUCS-TR-1995-010, Boston University, CS Dept, Boston, MA 02215, April 1995.

[7] A. Belloum and L. O. Hertzberger, "Dealing with One-Timer-Documents in Web Caching," *Proceedings of the 24th Euromicro Conference*, Vol. 2, pp. 544-550, 1998.

[8] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell "A Hierarchical Internet Object Cache," http://netweb.usc.edu/danzig/cache/cache.html.

[9] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," *Proceedings of IEEE INFOCOM'99*, Vol. 1, pp. 126-134, 1999.

[10] M. R. Korupolu and M. Dahlin, "Coordinated Placement and Replacement for Large-Scale Distributed Caches," *IEEE Workshop on Internet Applications*, pp. 62-71, 1999.

[11] J. Wang, "A Survey of Web Caching Schemes for the Internet," *ACM Computer Communication Review*, vol. 29, No. 5, Oct 1999.

[12] H. Che, Y. Xia, H. M. Wang, and X. M. Qiu, "Model for Magnetic Phase Transitions of $RBa_2Cu_3O_{6+x}$ Compounds," *Physics Letters A*, Vol. 146, No. 6, pp.343-346, June 1990.
[13] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox, "Caching Proxies: Limitations and Potentials," *Proc. 4th Inter. World-Wide Web Conference, Boston, MA*, Dec. 1995.
[14] F. P. Kelly, "The Clifford Paterson Lecture, 1995: Modeling Communication Networks, Present and Future," *Proceeding of Royal Society, London A*, Vol. 444, pp. 1-20, 1995.
[15] D. Bertsekas and R. Gallager, "Data Networks," Prentice-Hall, 1987.
[16] G. R. Grimmett and D. R. Stirzaker, "Probability and Random Processes," Oxford Science Publications, 1992.