Hierarchical Web Caching Systems: Modeling, Design and Experimental Results

Hao Che, Ye Tung, *Member, IEEE*, and Zhijun Wang

***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 low-pass filter with its cutoff frequency equal to the inverse of the characteristic time. Documents with access frequencies lower than this cutoff frequency 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 low-pass 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. Both model-based and real trace simulation studies show that the proposed cooperative architecture results in more than 50% memory saving and substantial central processing unit (CPU) power saving for the management and update of cache entries compared with the traditional uncooperative hierarchical caching architecture.**

***Index Terms*—Cache replacement algorithms, hierarchical caching, least recently used (LRU), web caching.**

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.

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 [9], i.e., bottom, institutional, regional, and backbone. Hence, it is of fundamental importance to explore the design principles of cache replacement algorithms for hierarchical caching. In this paper, we explore the fundamental design principles associated with the cache replacement algorithm design when caches are arranged in a hierarchical structure.

Most of the research papers on cache replacement algorithm design, to date, have focused on a single cache, e.g., [3], [6], [8], [10], [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., [11], 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 based algorithms. 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 [5], [12].

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], [3], 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. An approximation technique 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 (see Section III). The third contribution is the application of the design principles to the design of a cooperative hierarchical caching architecture. Model and real trace based simulations demonstrate that the proposed architecture provides more than 50% memory saving and substantial central processing unit (CPU) power saving for the

Manuscript received May 2001; revised December 2001. This work was presented in part at IEEE Infocom'01 Conference, Anchorage. Alaska, April 2001.

H. Che is with Santera Systems Inc., Plano, TX 75074 USA (e-mail: hao_che@hotmail.com).

Y. Tung is with University of South Alabama, Mobile, AL 36688 USA (e-mail: yxt7psu@hotmail.com).

Z. Wang is with University of Texas at Arlington, Arlington, TX 76019 USA (e-mail: zjxia@yahoo.com).

Publisher Item Identifier 10.1109/JSAC.2002.801752.

management and update of cache entries compared with the traditional uncooperative hierarchical caching architecture.

The remainder of this paper is organized as follows. In Section II, the model is described and analytical results derived. In Section III, the performance results are presented and the design principles proposed. Based on the principles proposed in Section III, Section IV introduces a cooperative hierarchical caching architecture and the performance of the proposed architecture is compared with the traditional uncooperative one by both model based and real trace based simulations. Finally, conclusions and future work are given in Section V.

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 toward 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 sections, 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 ($k = 1,2,\ldots,M$) at the leaf level. The network architecture of our model is presented in Fig. 1. Since LRU is a widely adopted cache replacement algorithm, 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 ($k = 1,2,\ldots,M$) is Poisson with mean arrival rate λ_k.
2) The arrivals of the request for individual document i ($i = 1,2,\ldots,N$) 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.

Assumptions 1 and 2 imply that the request arrival process for document i ($i = 1,2,\ldots,N$) 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 ($i = 0,1,2,\ldots,M$) 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., [3], [7], [10], as well as our own tests in Section IV, we model p_{ki} by Zipf-like distributions

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

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].

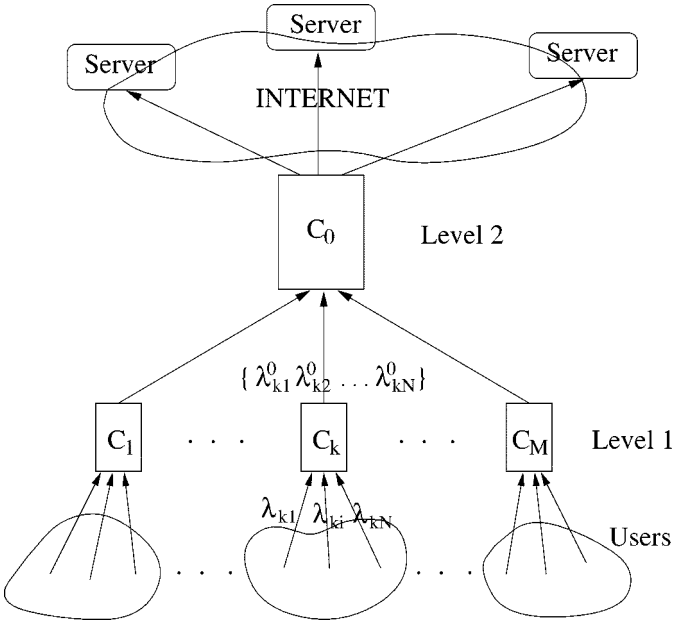

Fig. 1. Two-level hierarchical web caching structure.

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 ($k = 1,2,\ldots,M$) to the root cache be λ_{ki}^0 and the average miss rate of document i at the root cache be λ_{0i}, as shown in Fig. 1. Note that λ_{ki}^0 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_{ki}^0 and λ_{0i} as:

$$\begin{aligned} \eta_{ki} &= \frac{\lambda_{ki}^0}{\lambda_{ki}} \\ \eta_i &= \frac{\lambda_i^0}{\sum_{k=1}^{M} \lambda_{ki}} \\ \eta_k^0 &= \frac{\sum_{i=1}^{N} \lambda_{ki}^0}{\sum_{i=1}^{N} \lambda_{ki}} \\ \eta &= \frac{\sum_{i=1}^{N} \lambda_{0i}}{\sum_{k=1}^{M}\sum_{i=1}^{N} \lambda_{ki}} \end{aligned} \tag{3}$$

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_k^0 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, λ_{ki}^0 and λ_{0i} themselves are, in fact, more insightful performance measures than the cache miss ratios defined above.

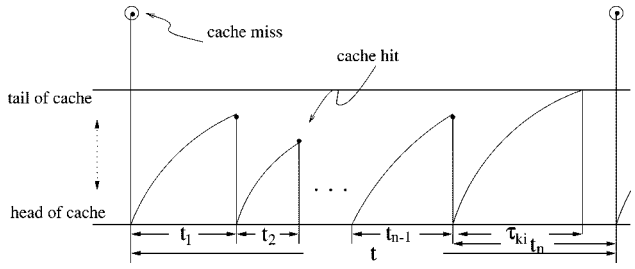

Fig. 2. Arrival processes of a given document at both level caches.

B. Model Analysis

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

1) 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 independent and identically distributed (i.i.d.) random variables $\{t_1, t_2, \ldots, t_{n-1}\}$ plus an independent random variable t_n. Each epoch t_i $(i = 1, 2, \ldots, n-1)$ 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 us 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 toward 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 toward 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 us first calculate the distribution density function $f_{ki}^0(t)$ 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 an approximation technique to make the problem tractable.

Define τ_{ki} as the maximum inter-arrival time between two adjacent requests for document i without a cache miss at cache k $(k = 0, 2, \ldots, M)$ as shown in Fig. 2. In essence, τ_{ki} is a random variable. However, in our approximation technique, two approximations are made.

1) τ_{ki} is a constant for any given k and i.
2) τ_{ki} is a constant with respect to i $(i = 1, 2, \ldots, N)$ for any given k.

The rationale behind these two approximations is based on the following intuition: As the aggregated arrival rate of all the other document requests increases, this rate becomes more and more deterministic and approaches a constant. As long as the cache size is reasonably large, Approximation 1 can be expected to be a good one. Moreover, as long as the request rate of each document i constitutes a small percentage of the aggregated request rate, Approximation 2 can be expected to be a good one too. Since the request rate for the most popular document constitutes less than 10% of the aggregated request rate according to the Zipf-like distribution (for real web traffic, this percentage is even smaller due to the domination of one-time document requests), this approximation should be a good one. Our model based 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. These approximations are made not only for the tractability of mathematical development in this section but also for the identification of τ_{ki} as an important characteristic time for a given cache, as we shall explain in the following sections. For this reason, the accuracy of these two approximations are further verified in Section IV, based on a large number of real trace simulations.

With this approximation technique, 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: $t_j \le \tau_{ki}$ $(j = 1, 2, \ldots, n-1)$ and $t_n > \tau_{ki}$. In fact, Approximation 2 is unnecessary for the mathematical formulation. It is used only when explicit analytical results are sought, as we shall see shortly. τ_{ki} 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, $f_{ki}^0(t)$ can be formally expressed as follows:

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

where

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

Next, take the Laplace transform of $f_{ki}^0(t)$. 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

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

and

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

2) Calculation of T_{0i}: With the proposed approximation technique, 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 calculation of T_{0i} involves three key steps: 1) construct the distribution function $f_{ki}^0(t)$ or the corresponding cumulative distribution function $P_{ki}^0(t < \tau)$; 2) construct the aggregate cumulative distribution function $P_{0i}(t < \tau)$ for the interarrival time of document i to the root cache; and 3) again, apply the same approximation to obtain T_{0i}.

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

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

where $u(t)$ is a step function with $u(t) = 0$ when $t < 0$ and $u(t) = 1$ otherwise. Note that $f_{ki}^0(t) = 0$ when $t < \tau_{ki}$, a consequence of the 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, $f_{ki}^0(t)$ is exactly described by finitely many terms. Despite all these nice features of the above expression, we still find it cumbersome when an explicit expression for T_{0i} is to be sought. Hence, instead of using (11), we use the following approximate expression for further development:

$$f_{ki}^0(t) \approx \begin{cases} \sigma_{ki}^0 e^{-\sigma_{ki}^0 (t - \tau_{ki})}, & \tau_{ki} < t < \infty \\ 0, & t < \tau_{ki}. \end{cases} \quad (12)$$

This is a truncated exponential distribution with

$$\sigma_{ki} = \frac{1}{\lambda_{ki}^{-1} - \tau_{ki}}. \quad (13)$$

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

$$P_{ki}^0(t < \tau) \approx \begin{cases} 1 - e^{-\sigma_{ki}^0 (\tau - \tau_{ki})}. & \tau_{ki} < \tau < \infty \\ 0, & \tau < \tau_{ki}. \end{cases} \quad (14)$$

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

$$P_{0i}(t < \tau) = 1 - \sum_{i=1}^{N} \lambda_{ki}^0 \left(\sum_{k=1}^{M} \lambda_{ki}^0 \left(1 - P_{ki}^0(t < \tau)\right) \right)^{-1} \times \prod_{k=1, k \neq k'}^{M} \int_{\tau}^{\infty} \lambda_{k'i} \left(1 - P_{k'i}^0(t < x)\right) dx. \quad (15)$$

In parallel to the approximation technique used at the leaf caches, the approximation technique 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 [16, Th. 10.4.5])

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

where

$$F_i(t < \tau_{0j}) = 1 - \left(\sum_{k=1}^{M} \lambda_{ki}^0 \right)^{-1} \int_0^{\tau_{0j}} (1 - P_{0i}(t < \tau)) \, d\tau. \quad (17)$$

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

$$T_{0i} = \sum_{n=0}^{\infty} [(n-1)\bar{t}_{0i}|_{t<\tau_{0i}} + \bar{t}_{0i}|_{t>\tau_{0i}}] \times P_{0i}(t < \tau_{0i})^{n-1} (1 - P_{0i}(t < \tau_{0i})) \quad (18)$$

or

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

$\bar{t}_{0i}|_{t<\tau_{0i}}$ ($\bar{t}_{0i}|_{t>\tau_{0i}}$) 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). Note that up to this point, Approximation 2 made at the beginning of this section has not been used. However, due to the peculiar dependency of $P_{ki}^0(t < \tau)$ with respect to τ_{ki} in (14), the general expression is lengthy and cumbersome. To get the explicit analytical results, in the following development, we use Approximation 2, namely, $\tau_{ki} = \tau_k$, i.e., τ_{ki} is independent of i. We further consider a special case: assume that λ_k, C_k, and z_k are the same for all k ($k = 1, 2, \ldots, M$). Then $\tau_k = \tau_1$ for $k = 2, 3, \ldots, M$. 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 (20) shown at the bottom of the page. With a straightforward but tedious calculation based on (20), we arrived at the following expressions:

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

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

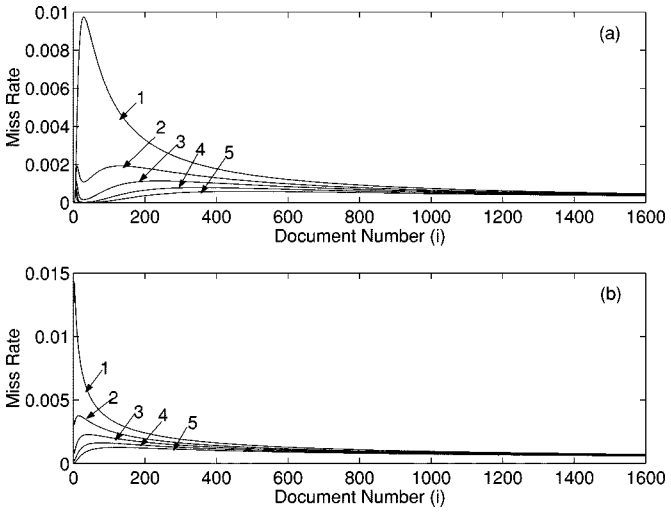

Fig. 3. Cache miss rates for individual documents—homogeneous case. (a) $z_k = 1$. (b) $z_k = 0.6$. Curve 1: cache miss rate at leaf level cache. Curves 2, 3, 4, and 5: cache miss rates at the root cache for $C_0 = 800$, 1200, 1600, 2000.

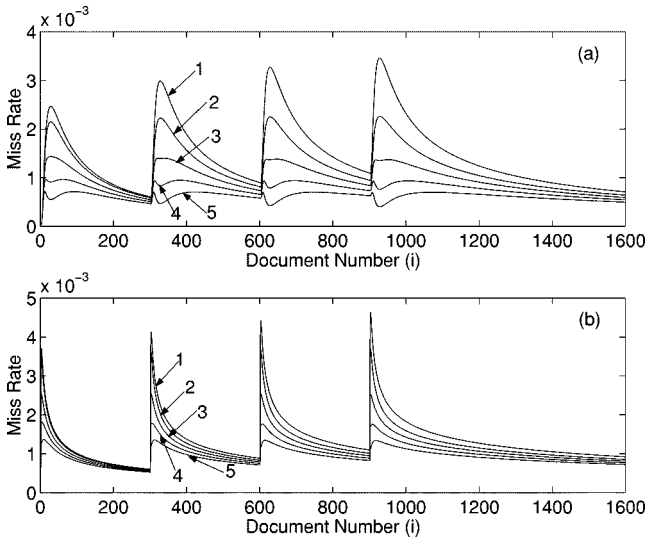

Fig. 4. Cache miss rates for individual documents—inhomogeneous case. (a) $z_k = 1$. (b) $z_k = 0.6$. Curve 1: cache miss rate at leaf level cache. Curves 2, 3, 4, and 5: cache miss rates at the root cache for $C_0 = 800$, 1200, 1600, 2000.

and

$$\bar{t}_{0i}|_{t>\tau_{0i}} = \tau_{0i} - (1 - P_{0i}(t < \tau_{0i}))^{-1} \times \prod_{k=1}^{M} \lambda_{ki}^{0} \sigma_{ki}^{-1} \left(\sum_{k=1}^{M} \lambda_{ki}^{0} \right)^{-1} \times e^{-\sum_{k=1}^{M} \sigma_{ki}(\tau_{0i} - \tau_1)} \tag{22}$$

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. Numerical Analysis and Design Principles

A. Numerical Analysis

In this section, 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 which focus on the analysis of aggregate cache hit/miss performance, our study focuses on the analysis of cache hit/miss performance for *individual* documents. The focal point is on the derivation of design principles.

For all the numerical 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 $C_0 = 800$, 1200, 1600, 2000, respectively. Given that $N = 20\,000$, $C_0 = 2000$ 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). Fig. 3(a) and (b) corresponds to $z_k = 1$ and 0.6, respectively. In each part , curve 1 represents λ_i^0. Curves 2, 3, 4, and 5 correspond to λ_{0i} for $C_0 = 800$, 1200, 1600, 2000, respectively. As expected, the cache miss rates at both level caches in Fig. 3(b) are higher than that in Fig. 3(a). Both parts of Fig. 3 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, 1200, further increasing C_0 does not significantly improve cache miss performance, especially for the case in Fig. 3(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 Fig. 3(b) drop more sharply than that in Fig. 3(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 $\tau_{ki} = \tau_1$. Equation (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 Fig. 3(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 $C_0 = 2000$ 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. Note that τ is a well-defined parameter provided that the two approximations made in the previous section hold. The simulation based on a large number of real traces in the following section verifies that τ is really a well-defined parameter. The miss rate or miss frequency λ_i of any given document i with respect to 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 numerical analyses in the previous subsection immediately lead to the following conclusions. A cache can be viewed conceptually as a low-pass 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 $(i = 1, 2, \ldots, L)$. Then, if we view these caches as a tandem of low-pass filters with cutoff frequencies τ_i^{-1} $(i = 1, 2, \ldots, L)$, 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 $l > l', l, l' = 1, 2, \ldots, L$. 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 Principle 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 Figs. 3 and 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 converge 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 section a cooperative hierarchical caching architecture based on these design principles.

A. Architecture Overview

The proposed cooperative hierarchical caching architecture can be described as follows. Like the traditional uncooperative hierarchical caching architecture, such as Harvest [4], the LRU algorithm is used locally at individual caches throughout the cache hierarchy and a request is searched upward starting from the lowest level cache.

However, the proposed architecture involves two major changes to the traditional architecture. First, in this architecture, a cache hit of document i at level l ($l = 2, 3, \ldots, L, L+1$; 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 and CPU 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 in 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 lifetime 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 estimate λ_{ki} $(i = 1, 2, \ldots, N)$ and each level cache estimate its own characteristic time. Also, note that the proposed architecture is cooperative in the sense that the 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 decision based on the measured τ_l $(l = 1, 2, \ldots, L)$ 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 section, a measurement scheme is proposed to enable the estimate of τ_l and λ_{ki}.

B. Measurement Schemes

1) Characteristic Time: In the proposed 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 document upon caching 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 min. For our simulation study based on time invariant Poisson arrival processes as well as real traces, the fluctuations around the mean value for τ_l is found to be very small.

2) 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, which runs in only the leaf caches, serves the filtering purpose for all the caches in the cache hierarchy, not just for a single cache.

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 broadcast 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 an access frequency table for individual document URLs. A document URL is deleted from the table if the elapsed time since the last document access of the access frequency table exceeds ΔT. To further reduce the computation complexity, in our implementation, only a timestamp of the last document access time is kept and updated for each document. The request access frequency is estimated by taking the inverse of the time interval between the time of the current document access and the time of the last document access.

3) Cache Search Algorithm: The cache search algorithm works as follows.

1) Upon the arrival of a request at leaf cache k $(k = 1, 2, \ldots, M)$, the cache is searched. If there is a cache hit, the document is sent back to the requesting host and the algorithm is exited. Otherwise, go to step 2.
2) Search the access frequency table. If a match is found, calculate the access frequency λ_{ki}, otherwise set $\lambda_{ki} = 0$, create an entry for this request and put the current timestamp in the entry.
3) Send λ_{ki} together with the request to the next level cache for further search.
4) At level m cache $(1 < m \leq L)$, if no entry is found which matches the requested document, let $m = m + 1$, and go back to the beginning of Step 4. Otherwise, the document and λ_{ki} are sent via the reverse path to the requesting host. At each intermediate level cache m', if $\tau_{m'}^{-1} < \lambda_{ki}$, a copy of the document is cached before it is sent to the next lower level cache.

Note that since the document found at level m $(m = 2, \ldots, L, L+1)$ will not cause the document caching in all the lower level caches, especially for one-time documents, the cache efficiencies are improved in all level caches. Preventing those documents which will for sure not incur any hits from being cached not only saves cache memory but also CPU power. As we shall see in the following section, one-time document requests constitute a large portion of the total document requests and consequently the proposed algorithm results in great reduction of cache memory size and CPU power consumption.

A major added complexity of our algorithm is the need for the management of an access frequency table at each leaf cache. This added complexity is, however, minimum compared with the complexities for regular cache searches and LRU updates. Besides, by setting the timeout value at ΔT, both frequent [1] and infrequent requests are quickly timeout from this table, resulting in a small table size to be managed.

C. Performance Evaluation

A performance analysis is performed for the proposed hierarchical caching architecture compared with the uncooperative caching architecture described at the beginning of Section II. 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.

1) Model Based Simulation Study: Again, consider a two-level hierarchical caching system with $M =$ four 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 $C_0 = 1200$ based on simulation and the uncooperative architecture at $C_0 = 1200$ and 2400 based on numerical analysis.

Fig. 5 presents the results. Parts (a) and (b) of the figure give the results for the homogeneous and inhomogeneous cases, respectively. For both cases, the overall cache miss rate as defined in (3), for the proposed architecture at $C_0 = 1200$ are found to be very close to the cache miss rate for the uncooperative architecture at $C_0 = 2400$. This means the proposed architecture

[1]Since only those requests which receive cache misses will cause caching in the access frequency table, frequent requests will not appear in this table simply because they are in the leaf cache without being replaced.

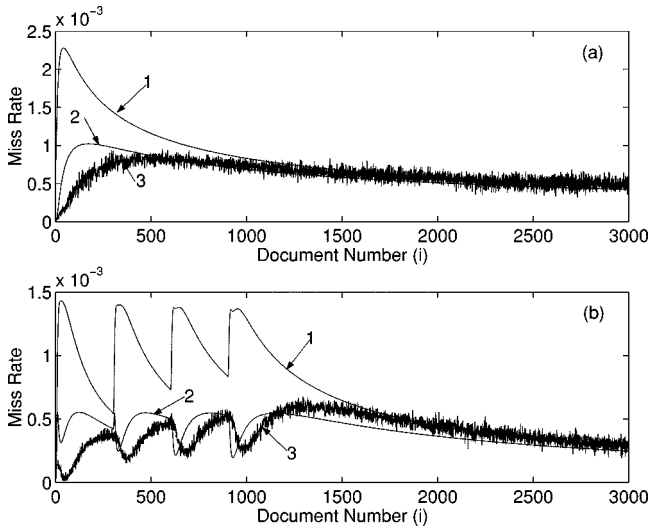

Fig. 5. Cache miss rates for individual documents at the root cache for cooperative (curve 3) and uncooperative (curves 1 and 2) architectures. (a) Homogeneous case. (b) Inhomogeneous case. Curves 1 and 3: cache miss rates at $C_0 = 1200$, and curve 2: cache miss rate at $C_0 = 2400$.

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 $C_0 = 1200$ for both subplots and focusing on the first 1300 popular documents, one observes that an improvement by a factor of two or more is achieved by using our proposed architecture. The proposed architecture leads to a small loss of performance 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, which, in general, improves customer satisfactions using the web service.

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 $C_0 = 1200$, $\tau_0 = 332.4$ for uncooperative case and $\tau_0 = 1875$ for cooperative case, increasing by about six times.

2) Real Trace Simulation Study: Our real trace simulation study focuses on the following two issues. First, we want to verify the assumption and the approximations made in the previous sections, i.e., the assumption of the Zipf-like distribution and the two approximations which lead to the concept of characteristic time. Second, we further test the effectiveness of the proposed architecture compared with the traditional one.

The National Laboratory for Applied Network Research (NLANR) manages a hierarchical Internet cache system built upon Squid caches. NLANR is one of the few organizations which provide the real web traces for public access. The traces used for this study are downloaded from the NLANR web site [17]. Following the naming convention by the trace owner, the names of the traces studied and the corresponding proxy locations are listed as follows:

- bo: Boulder, CO;
- pa: Palo Alto, CA;
- pb: Pittsburgh, PA;

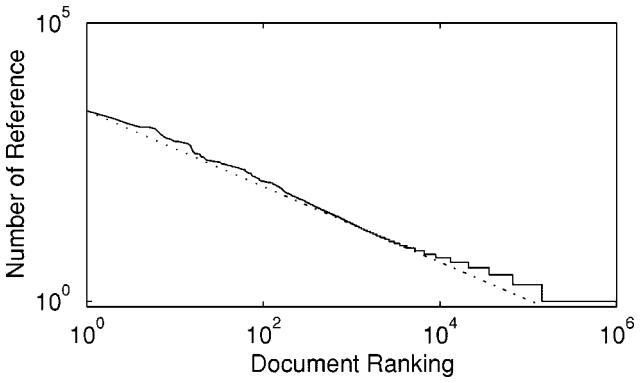

Fig. 6. Frequency of document accesses versus document ranking.

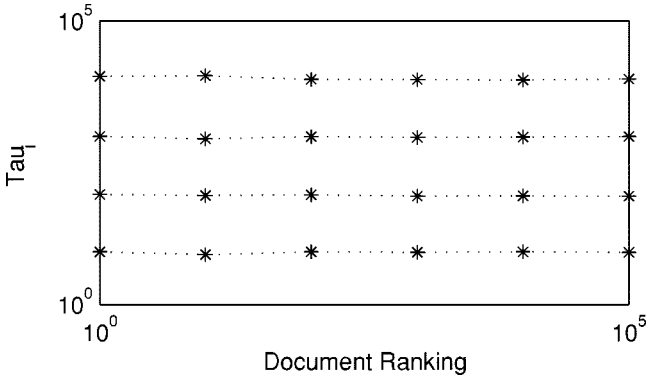

Fig. 7. τ_i versus document ranking for different cache size.

- rtp: Research Triangle Park, NC;
- sd: San Diego, CA,;
- st: STARTAP, the international connection point in Chicago, IL;
- sv: Silicon Valley, CA (FIX-West);
- uc: Urbana-Champaign, IL.

These traces were collected during the period from December, 2000 through April, 2001. Each trace contains a one day log. The traces were preprocessed to extract the information containing timestamp and URL from each entry.

First, we test whether these traces follow the Zipf-like distribution well. The distribution of all the traces are calculated and the results show that the distribution for any of these traces follows the Zipf-like distribution reasonably well, expect for the tail portion. The value of z_k varies from one proxy to another, ranging from 0.64 to 0.75. As an example, Fig. 6 shows the distribution for trace sv (dated 4/12/01) together with the Zipf-like distribution at $z_k = 0.68$. One can see that the curve fits the Zipf-like distribution reasonably well, especially for the middle portion. There appears to be some flattening at the most popular end and least popular end (i.e., the tail portion), mainly unfit at the least popular end where one-time documents account for more than 70% of the total document references.

Second, we test whether Approximation 1 made in Section II holds well. τ_i in a wide range of i values (1 to 100 000) are sampled at various cache sizes. The results show that without exception, the τ_i samples take value within 15% around its mean values for all the traces tested. For instance, for trace sd (dated 2/19/01) at cache size of 1000, the 1000th document has an average τ of 82.3 with variation from 77.6 to 93.7.

TABLE I
SIMULATION RESULTS FOR A LEAF CACHE

Trace Index (4/26/01)	Reference Number	Distinct Doc. No.	One-time Doc. No.	Cache Miss Ratio	Cache Size (LRU)	Cache Size (New Approach)
bo	130164	96052	90564	0.88	1000	277
pa	275351	163363	135639	0.85	1500	355
pb	355758	225461	174892	0.90	1500	371
rtp	789246	469007	424970	0.82	4000	1450
sd	1160607	601315	479411	0.85	5000	1874
st	135385	93009	86728	0.87	1000	357
sv	1065379	814762	779023	0.89	4000	1887
uc	747867	419818	345852	0.63	4000	1555

Next, we test whether Approximation 2 made in Section II is a good one. The simulation studies for all the traces show that this approximation is very accurate. As an example, Fig. 7 gives the average τ_i for $i = 1, 10, 100, 1000, 10000, 100\,000$ at cache sizes $C = 10, 100, 1000, 10\,000$ for trace pa (dated 1/26/01). One can see that average τ_i is pretty close to a constant for different i's at any given cache size. These two tests validate the use of the characteristic time as a well-defined parameter to characterize the cache bahavior.

Finally, we study the performance of the proposed architecture compared with the traditional uncooperative architecture. Due to the fact that the available traces are filtered traces by the lower level caches and they are collected from widely different locations all over the country, it would be rather artificial to use these traces as leaf cache input to simulate a complete hierarchical system. Instead, we focus on the performance study of a single leaf cache using these traces as input. The focal point is placed on getting a rough estimate of the potential performance gain for the proposed architecture over the traditional one.

Assume $\Delta T = 1.2\tau$ where τ is the average characteristic time of the leaf cache. Table I lists the trace name, the number of document references, the total number of distinct documents, the total number of one-time document references, the cache miss ratio, the cache sizes for both the traditional approach, and the proposed approach.

First, one observes that one-time document requests represent a large portion of the total document requests for all the traces. This makes the average number of per document references very small. For instance, for trace bo, the one-time references represent 70% (90564/130 164) of the total references which makes the average number of per document references as low as 1.36 (130 164/96 052). This translates into rather high cache miss ratio of 0.88.

Since the traditional approach does not make an attempt to prevent the one-time documents from being cached in the cache table, the cache efficient is extremely low. A large portion of the proxy resources including cache memory and CPU power are used for caching and replacing one-time documents without a cache hit. In contrast, the proposed algorithm results in tremendous cache memory savings, e.g., 72.3% [(1000-277)/1000] for trace bo. Moreover, since only frequently referenced documents are cached, most of the time the CPU is performing pointer exchanges for documents which receive cache hits, rather than frequently adding new documents to and removing old documents from the cache due to cache misses. Hence, the proposed algorithm substantially reduces the search complexity due to its much reduced cache size.

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 algorithms, such as LRU variants or GD-Size. 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

$$\begin{aligned}\phi_{ki}(s) &= \sum_{n=1}^{\infty} P_{ki}(n) \int e^{-st} f_{ki}(t \mid n)\, dt \\ &= \sum_{n=1}^{\infty} [(1-q_{ki})M_1(s)]^{n-1} q_{ki} M_2(s) \\ &= \frac{q_{ki} M_2(s)}{1-(1-q_{ki})M_1(s)} \end{aligned} \tag{25}$$

where

$$M_1(s) = \int_0^{\tau_{ki}} e^{-st} \frac{\lambda_{ki} e^{-\lambda_{ki} t}}{1 - e^{-\lambda_{ki}\tau_{ki}}} dt = \frac{\lambda_{ki}\left(e^{(-s-\lambda_{ki})\tau_{ki}} - 1\right)}{(-s-\lambda_{ki})(1 - e^{-\lambda_{ki}\tau_{ki}})} \quad (26)$$

and

$$M_2(s) = \int_{\tau_{ki}}^{\infty} e^{-st} \frac{\lambda_{ki} e^{-\lambda_{ki} t}}{e^{-\lambda_{ki}\tau_{ki}}} dt = \frac{\lambda_{ki} e^{(-s-\lambda_{ki})\tau_{ki}}}{(\lambda_{ki}+s) e^{-\lambda_{ki}\tau_{ki}}}. \quad (27)$$

Substituting (26) and (27) into (25), we get (8).

APPENDIX B

From (8) and (10), we have

$$\phi_{ki}(s) = \frac{1}{1 + (\lambda_{ki}^0 s)^{-1} e^{-s\tau_{ki}}} = \sum_{n=1}^{\infty} (-1)^{n+1} \left(\lambda_{ki}^0\right)^{-n} s^{-n} e^{-ns\tau_{ki}}. \quad (28)$$

Making use of the inverse Laplace transform

$$\frac{e^{ks}}{s^{\mu}} \Leftrightarrow \frac{(t-k)^{\mu-1}}{\eta(\mu)} u(t-k), \quad \mu > 0 \quad (29)$$

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. 37–45, Nov. 1997.

[2] H. Che, Z. Wang, and Y. Tung, "Analysis and design of hierarchical web caching systems," *Proc. IEEE INFOCOM'01*, vol. 3, pp. 1416–1424, Apr. 2001.

[3] J. Zhang, R. Izmailov, D. Reininger, and M. Ott, "Web caching framework: Analytical models and beyond," in *IEEE Workshop Internet Applications 1999*, July 1999, pp. 132–141.

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

[5] S. G. Dykes, C. L. Jeffery, and S. Das, "Taxonomy and design analysis for distributed web caching," in *Proc. HICSS-32*, 1999, pp. 286–295.

[6] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," in *Proc. 1997 USENIX Symp. Internet Technology and Systems*, Dec. 1997, pp. 193–206.

[7] C. R. Cunha, A. Bestavros, and M. E. Crovella, "Characteristics of WWW Client-Based Traces," Boston University, CS Dept., Boston, MA 02 215, Tech. Rep.BUCS-TR-1995-010, Apr. 1995.

[8] A. Belloum and L. O. Hertzberger, "Dealing with one-timer-documents in web caching," in *Proc. 24th Euromicro Conf.*, vol. 2, 1998, pp. 544–550.

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

[10] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," *Proc. IEEE INFOCOM'99*, vol. 1, pp. 126–134, Apr. 1999.

[11] M. R. Korupolu and M. Dahlin, "Coordinated placement and replacement for large-scale distributed caches," in *Proc. IEEE Workshop on Internet Applications*, June 1999, pp. 62–71.

[12] J. Wang, "A survey of web caching schemes for the internet," *ACM Comp. Commun. Rev.*, vol. 29, no. 5, pp. 36–46, Oct. 1999.

[13] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox, "Caching proxies: Limitations and potentials," in *Proc. 4th Int. World-Wide Web Conf*, Boston, MA, Dec. 1995.

[14] F. P. Kelly, "The clifford Paterson lecture, 1995: Modeling communication networks, present and future," in *Proc. Royal Society, London A*, vol. 444, 1995, pp. 1–20.

[15] D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[16] G. R. Grimmett and D. R. Stirzaker, *Probability and Random Processes*. New York: Oxford Univ. Press, 1992.

[17] Anonymized Access Log [Online]. Available: ftp://ircache.nlanr.net/Traces/

Hao Che received the B.S. degree from Nanjing University, Nanjing, China, the M.S. degree in physics from the University of Texas at Arlington, TX, in 1994, and the Ph.D. degree in electrical engineering from the University of Texas at Austin, TX, in 1998.

He was an Assistant Professor of Electrical Engineering at the Pennsylvania State University, University Park, PA, from 1998 to 2001. Since July 2000, he has been a System Architect with Santera Systems, Inc., Plano, TX . He has also served as an Adjunct Assistant Professor at the Pennsylvania State University and a Visiting Professor in the school of Information Science and Engineering, Yunnan University, China, since 2001. His current research interests include network architecture and design, network resource management, multiservice switching architecture, and network processor design.

Ye Tung (S'95–M'01) received the B.S. degree from Xian Jiaotong University, Xian, P. R. China, in 1990, and the Ph.D. degree from the Pennsylvania State University, University Park, PA, in 2001.

Since January 2002, he has been an Assistant Professor at the Department of Electrical and Computer Engineering, University of South Alabama, Mobile, Alabama. His current research interests include communication systems, computer networking, and server technology.

Zhijun Wang received the M.S. degree in physics from Huazhong University of Science and Technology, Wuhan, China, in 1992, and the M.S. degree in electrical engineering from the Pennsylvania State University, University Park, PA, in 2001.

He is working toward the Ph.D. degree in computer science and engineering at the University of Texas at Arlington, TX. His current research interests include wireless web cache management, mobile computing, and wireless networks.