Meta Algorithms for Hierarchical Web Caches

Nikolaos Laoutaris, Sofia Syntila, Ioannis Stavrakakis
{laoutaris,stud1245,istavrak}@di.uoa.gr

Department of Informatics and Telecommunications,
University of Athens, 15784 Athens, Greece

***Abstract*— Large scale hierarchical caches for web content have been deployed widely in an attempt to reduce delivery delays and bandwidth consumption and also to improve the scalability of content dissemination through the world wide web. Irrespectively of the specific replacement algorithm employed in each cache, a de facto characteristic of contemporary hierarchical caches is that a hit for a document at an l-level cache leads to the caching of the document in all intermediate caches (levels $l-1,\ldots,1$) on the path towards the leaf cache that received the initial request. This paper presents various algorithms that revise this standard behavior and attempt to be more selective in choosing the caches that get to store a local copy of the requested document. As these algorithms operate independently of the actual replacement algorithm running in each individual cache, they are referred to as *meta algorithms*. Three new meta algorithms are proposed and compared against the de facto one and a recently proposed one by H. Che, Y. Tung, and Z. Wang [1] by means of synthetic and trace-driven simulations. The best of the new meta algorithms appears to be able to lead to improved performance under most simulated scenarios, especially under a low availability of storage. The latter observation makes the presented meta algorithms particularly favorable for the handling of large data objects such as stored music files or short video clips. Additionally, a simple load balancing algorithm that is based on the concept of meta algorithms is proposed and evaluated. The algorithm is shown to be able to provide for an effective balancing of load thus possibly addressing the recently discovered "filtering-effect" in hierarchical web caches (C. Williamson [2]).**

I. Introduction

Keeping temporary copies of valuable information on fast access memories, so as to avoid accesses to slow memories where the information is stored on a permanent basis, is a commonly employed technique to reduce the overall access time. This technique – known as *caching* – has been employed extensively in a variety of applications, most notably to speed up the communication between a CPU and the main memory or between the main memory and the secondary (permanent) storage (hard disks). The prevalence of the internet and the world wide web gave caching new fields of application. Initially, web browsers implemented local caches where frequently accessed pages were kept for fast access thus avoiding slow accesses to remote web servers. Soon it was realized that a higher gain would be achieved if multiple clients shared a larger cache thus requiring only a single copy of a document to service all the clients. This led to the development of dedicated cache (or proxy) servers that lay on the access point of a local area network to the internet, servicing the entire local client population. Dedicated proxy servers are known to be able to reduce the client perceived delay and also the bandwidth consumption on the backbone network. To be able to scale to large populations, and also to enhance the degree of sharing, local dedicated caches have federated thus creating large hierarchical caches such as the NLANR cache [3] and the UK JANET cache [4]. A hierarchical cache involves multiple proxies at different levels. Requests are first received at the leaf caches and are routed upwards until they reach a cache that stores a copy of the requested document. A hit is said to have occurred in that case. Following a hit, the requested document is sent on the reverse path to the client, and each cache on this path gets to store a local copy of the document so as to be able to service future requests.

The focus of this paper is to investigate whether caching a local copy in *all* intermediate caches on the reverse path is indeed a good idea, or are there reasons to revise it, and keep instead copies in a *subset* of intermediate caches. Notice that leaving copies everywhere (here after abbreviated LCE), has been considered as a de facto behavior. Despite the vast bibliography on web caching, it wasn't until very recently that a work appeared possing similar questions [1] and, in fact, providing some evidence that LCE can be improved.

The question of whether to cache a document at an intermediate cache is one that may be possed independently of the specific replacement algorithm operating on the cache. It may be seen as an admission mechanism, similar in conception to a Call Admission Control (CAC) mechanism on a multiplexor node; its sole purpose is to decide whether to keep a copy of a document coming from upwards in the hierarchy, leaving the decision of choosing a specific document to evict to make room for the new one to whichever replacement algorithm is in effect locally. For this reason, the algorithms that are studied here may be characterized as *meta algorithms* for hierarchical caches (or just meta algorithms) to differentiate them from the much discussed and well understood replacement algorithms and to stress the fact that they operate independently of the latter. Meta algorithms have been employed in the past in the different, but related to caching, domain of self-organizing linear search (see [5] and references). Apart from the apparently different application domain, the meta algorithms employed in this work operate on groups of caches organized in hierarchies, whereas earlier work on self-organizing linear search studied the operation of meta algorithms on isolated linear lists.

In the following, three new meta algorithms are described and compared against the standard LCE, and the recently

This work and its dissemination efforts have been supported in part by the IST Program of the European Union under contract IST-2001-32686 (Broadway).

proposed one in [1], by means of synthetic and trace-driven simulations. In all cases it is assumed that the Least Recently Used (LRU) replacement algorithm runs in all caches of the hierarchy. LRU is by far the most commonly used replacement algorithm and thus has been chosen as the basis over which the various meta algorithms are compared. Due to the aforementioned independent operation of the meta algorithm from the employed replacement algorithm, it is believed that the presented results and conclusions should apply to some extent to other replacement algorithms as well.

The results of the simulation study indicate that the new proposed algorithms (and most notably the best performing among them, called LCD) provide significant gains over LCE in most studied scenarios. It is also found that they can approximate and even exceed the performance of the more complex algorithm of [1]. The proposed meta algorithms are particularly suited to applications having a limited storage capacity. In such cases they manage to outperform the standard policy LCE by being more conservative in admitting documents to caches. This allows for the suppression of replacement error but also entails a cost in the form of reduced speed in tracking changing demand patterns. However, measured client workloads appear to be quite stable across time. In fact, a recently published measurement study [6] showed that the top 10% most popular documents on one day make up to around 80% of all requests for at least the following week. Thus the cost due to the reduced adaptability becomes much smaller as compared to the gain from the suppression of replacement error. Based on the derived results it is concluded that the new algorithms appear to be good, low complexity solutions, especially suitable for storage constrained applications. Although originally motivated by hierarchical web caching, it is later argued (see Sect. V) that the same meta algorithms may be applied to other applications such as the distribution of stored music files or short video clips where storage is much more limited as compared to the case of web content.

A second contribution of this work is a simple fully distributed load balancing scheme that is based on the concept of meta algorithms and can potentially address the recently discovered "filtering-effect" in hierarchical web caches (Williamson [2]). The proposed scheme is generic in nature and can be employed in conjunction with any of the studied meta algorithms. It prohibits the concentration of all popular documents at the leaf caches, as done by LCE, and instead spreads them more evenly, thus, achieving a smoother distribution of load (hits) among the caches of the hierarchy.

As a final note, it should be pointed out that this work is confined to the study of on-line, request driven, (meta) algorithms that may be employed with little, or no change, to the existing hierarchical caches. The idea is to try to improve these systems by enforcing only minor modifications. The reader is referred to [7] for the study of off-line co-operative object placement (replication) and to [8] for the study of co-operative replacement which, however, unlike the algorithms discussed here, employ a significant exchange of information between caches and incur a substantial complexity.

II. Meta algorithms for hierarchical caches

This section presents some new meta algorithms along with the design principles that guide their operation. Some older meta algorithms, which are used for comparison, are also presented.

A. Description

This section describes three new meta algorithms, Prob, LCD, MCD, as well as the currently employed one, LCE, and a recently proposed one, Filter.

1) Leave Copy Everywhere (LCE): This is the standard mode of operation currently in use in most hierarchical caches. When a hit occurs at a level l cache or the origin server, a copy of the requested document is cached in all intermediate caches (levels $l-1,\ldots,1$) on the path from the location of the hit down to the requesting client.

2) Prob: Prob is a randomized version of LCE. Each intermediate cache on the path from the location of the hit down to the requesting client is eligible for storing a copy of the requested document. An intermediate cache keeps a local copy with probability p, thus invoking the replacement algorithm, and does not keep a copy with probability $1-p$. Prob with $p=1$ is identical to LCE.[1]

3) Leave Copy Down (LCD): Under LCD a new copy of the requested document is cached only at the $(l-1)$-level cache, i.e., the one that resides immediately below the location of the hit on the path to the requesting client. LCD is more "conservative" than LCE as it requires multiple requests to bring a document to a leaf cache, with each request advancing a new copy of the document one hop closer to the client.

4) Move Copy Down (MCD): Similar to LCD with the difference that a hit at level l *moves* the requested document to the underlaying cache. This requires that the requested document be deleted[2] from the cache where the hit occurred. No deletion of course takes place when the hit occurs at the origin server. The idea behind MCD is to reduce the number of replicas for the same document on the path between the requesting client and the origin server.

The operation of the above mentioned algorithms is illustrated in an example in Fig. 1. Note that the three new meta algorithms require a very small amount of extra co-operation other than the minimum required to implement a hierarchical cache, i.e., each cache to know its immediate ancestor so that it can forward requests upstream, and its immediate descendants so that it can forward documents downstream. Prob is oblivious to where the original hit occurred thus runs solely on local information. LCD needs to know where the original hit occurred so that it can decide whether to keep a

[1] Probabilistic algorithms have been recently employed in the domain of web caching, but in different contexts. The authors of [9] show that replacements algorithms may be efficiently implemented by utilizing only samples from the cache (instead of the entire cache content) in order to identify a good eviction candidate. In [10] randomized algorithms are used in order to handle documents with varying costs (fetch distance) and sizes.

[2] The document does not have to be physically deleted from the cache. A better strategy is to set its timestampt to a very small value thus marking it for eviction upon the next miss. This has the advantage that in the case that the next request refers to this document, a hit will occur, whereas a miss would have occurred if physical deletion had taken place.

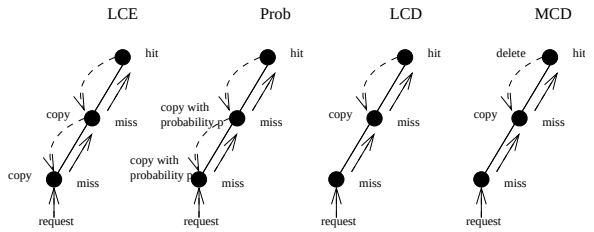

Fig. 1. Operation of LCE, Prob, LCD, and MCD.

local copy or not. This can be done by piggybacking the IP address of the hit location along with the document that is sent downstream. Then an intermediate cache checks whether the IP address of the hit location matches its ancestor's IP and only in that case it keeps a local copy. MCD can be implemented likewise. It also requires to delete the local copy of the document at the location of the hit after sending it downwards, which of course requires no information from other caches.

5) Filter: a non-memoryless meta algorithm: A major distinctive factor of replacement algorithms is whether they are memoryless or non-memoryless. The LRU algorithm is characterized as memoryless because each permutation of the cache, following a request, utilizes no additional information from the memory; the permutation is executed by simply bringing the requested document to the top of the LRU list. Frequency based algorithms, however, need to maintain additional information in the memory to keep track of the number of requests for each document. Such algorithms are characterized as non-memoryless. The Least Frequently Used, LFU, replacement rule is the most popular non-memoryless replacement algorithm. Non-memoryless algorithms take advantage of the extra information and thus generally achieve a better hit ratio.

The aforementioned characterization may be applied to the field of meta algorithms as well. All the aforementioned meta algorithms may be characterized as memoryless because they do not require any extra information other than the actual state of the hierarchy. Recently, H. Che, Y. Tung, and Z. Wang have proposed a new meta algorithm for hierarchical caches, aiming at improving the overall hit ratio [1]. This algorithm, that will here after be referred to as Filter, appears to be non-memoryless. We have implemented it, and used it to evaluate its performance against the memoryless Prob, LCD, and MCD. It is briefly outlined below.

Under the Filter algorithm, a hit for document i at level l on behalf of client k leads to the caching of i in an intermediate cache m on the path to k, when m satisfies the following condition: $\tau_m^{-1} < \lambda_{ki}$. τ_m is said to be the *characteristic time* of cache m. It is equal to the difference between the current time and a timestamp that indicates the time of last access to the document that would be replaced to make room for the caching of i if LCE was to be used. λ_{ki} is the frequency that client k requests document i. Each cache m is seen as a low pass filter with a cutoff frequency equal to τ_m^{-1} and, thus, a path of the hierarchy constitutes a line in tandem of low pass filters having different cutoff frequencies. Documents that have a low request frequency are allowed to pass from a cache without storing a local copy with the aim of saving cache resources since these documents stand a good chance of being replaced before being requested again. Additionally, when a document is evicted from a cache at level l the algorithm forces its caching at level $l+1$ if not already cached there. Filter incurs an additional complexity as compared to the previous memoryless meta algorithms as it needs to estimate the request frequency of each requested document in all clients and disseminate this information to all the caches. This extra processing and the information that must be maintained make Filter a non-memoryless meta algorithm.

B. Design Principles

The three new meta algorithms, Prob, LCD, and MCD, aim at improving the performance of a hierarchical cache in terms of the expected distance to reach a cache hit. To achieve this goal they take advantage of the following three design principles:

1) Avoid the amplification of replacement errors: When focusing on an isolated cache, a replacement algorithm is said to have committed an error if it chooses to evict a document i while there exists a document j that if evicted in place of i would lead to an improved hit ratio. The exact definition of a replacement error depends on the assumptions of the problem. For example, under a given sequence of requests, the optimal replacement algorithm is the one that in each replacement evicts the document that has the maximum forward distance (in number of requests) until the next request for the same document [11]. Any algorithm that violates this strategy will be committing replacement errors. The LRU replacement algorithm has a hit ratio that has been proved to be no more than K times worse than the hit ratio of the optimal offline algorithm, where K is the capacity of the cache in unit sized documents [12]. Obviously this optimal offline algorithm cannot be implemented as it requires knowledge of future requests which is generally not available (except for in-advance "reservation" systems).

A more practical model for the evaluation of replacement algorithms is the so called independent reference model (IR), under which requests are i.i.d. random variables following a certain popularity distribution over a given document universe. The optimal caching strategy under IR demands the static replication of the most popular documents up to the capacity of the cache [13]. Let this optimal caching strategy under IR be called *Highest Popularity First* (HPF) strategy. A replacement algorithm that operates under the IR model may evict one of the most popular documents in order to cache a less popular document, thus committing a replacement error which eventually leads to a reduced steady-state hit ratio as compared to that achieved under the HPF strategy. The LRU replacement algorithm under the IR model has a worst case hit ratio that has been proved to be at least $O(\log K)$ worse than the hit ratio of the HPF strategy, K denoting the capacity of the cache in unit sized documents [14].

From the previous it should be clear that any causal replacement algorithm is committing errors as compared to the optimal strategy (under the assumed model) and these errors

lead to inferior performance. This situation becomes even more critical when considering a hierarchical rather than an isolated cache. In an L-level hierarchical cache that operates under the LCE meta algorithm, a request for an unpopular document may lead to its caching in all L caches on the path from the requesting client up to the root cache, and by doing so commit up to L replacement errors. Leaving a copy in all the intermediate caches is, in effect, leading to the *amplification of replacement errors*. The proposed algorithms try to reduce the extent of this amplification by reducing the number of copies that are cached with each request.

2) Filter-out one-timer documents: A prominent characteristic of measured proxy workloads is a very high percentage of documents that are requested only once, despite of the duration of the studied workload. These so called $one-timer$ documents usually amount up to 45% of the total requests and 75% of the total distinct documents present in the measured workloads [15], [16]. Caching an one-timer document is the worse type of replacement error that can occur as it is guaranteed that the one-timer will not be requested again thus leading to waste of storage capacity. The insertion of a one-timer in the cache is equivalent to operating with a cache of a reduced capacity, equal to $K-1$, for the time duration that is required to have K other distinct documents being requested which lead to the eviction of the one-timer and the release of the temporarily wasted storage. The aforementioned high percentages of one-timers clog an entire hierarchical cache that operates under LCE with useless documents, which deprive useful popular documents of valuable storage capacity; this leads to poor hit ratios. The proposed LCD and MCD meta algorithms guarantee that the one-timers cannot affect any cache other than the root cache. Thus they completely filter-out one-timers for all but one caches in the hierarchy. Prob, likewise, filters out most of the one-timers by using a small cache probability p.

3) Rationalize the degree of replication: A request for a document under LCE may lead to the caching of a local copy in all caches on the path to the root, if the document was not previously cached on that path. Despite the location of the hit, LCE guarantees that a copy will be stored at the leaf cache that services the requesting client. Since this leaf cache is the one closest to the client, the upper copies could have been omitted without affecting the specific client. LCE "proactively" places copies at the upper levels to achieve the following two goals: (1) have a nearby copy to service other clients connected to leaf caches that do not have a copy of the document; (2) have a "backup" copy for the requesting client in case its leaf copy is evicted from the leaf cache. Situation (1) targets the so called *first access* or *cold* misses, i.e., the inavoidable misses occurring when requesting a document that has not been requested before, thus no copy of it exists at the leaf cache. Figure 2 gives an example of how the copies at the upper levels reduce the hit distance for cold misses at other leaves. Request 1 incurs 3 misses as the requested document initially resides only at the origin server. Request 2 is for the same document but this time there is only one miss and the hit occurs at a level-2 cache. The shorter hit distance owes to the upper level copies stored by request 1. Request 3 follows,

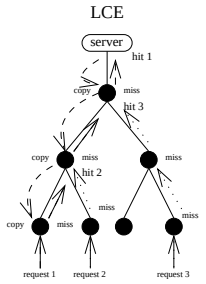

Fig. 2. Example of how LCE reduces the hit distance for *cold* misses by proactively storing multiple copies at upper level caches. Request 1 incurs 3 misses but the following requests incur a smaller number of misses, 1,2 respectively, owing to the upper level copies stored by the first request.

originating at the far right leaf cache, and it too experiences a smaller number of misses, due to the upper copies stored by the first request.

For the aforementioned reasons, LCE leads to a large number of document replicas being stored with each request. There are situations, however, that the appropriateness of this high degree of replication may be challenged. Consider the extreme case that each client references a distinct set of documents without any overlap in the document sets referenced by different clients. Under such a scenario, goal (1) – reducing the hit distance for cold misses – ceases to exist because leaving a copy at an upper level serves no other client. The same applies to a smaller degree when there is only a partial overlap in the various document sets. Goal (2) may also be questioned. Storing a "backup" copy at an intermediate cache may not be beneficial, especially under a limited storage capacity. Indeed, a popular document at an intermediate level will probably receive very few requests and be replaced quickly since copies of it will probably exist lower at the hierarchy thus filtering the majority of the requests. Prob, LCD, and MCD create relatively fewer copies for each requested document thus leading to a smaller degree of replication for each cached document, allowing for more distinct documents to be cached.

III. Synthetic simulations

For the purpose of evaluating the performance of the proposed meta algorithms, simulations were conducted using synthetically generated Zipf-like document popularity distributions to model client requests, which are assumed to be i.i.d. random variables, i.e., the independent reference (IR) model is assumed. Under a Zipf-like distribution the probability of requesting the ith most popular document is given by $p(i) = C/i^a$, where $C = (\sum_{j=1}^{N} \frac{1}{j^a})^{-1}$. N denotes the number of document in the universe and a is the skewness parameter of the Zipf-like distribution, indicating the degree of concentration of requests. Values of a close to 1 indicate that few distinct documents attract the majority of the requests while values close to 0 indicate almost uniform document popularities. A Zipf-like distribution has been reported to be a good model of actual measured workloads [17], [18], [15], [16], [2], with typical values of a in the range [0.6,0.9]. The assumption that successive requests are independent has been used extensively by many researchers and has been shown to

be able to capture asymptotic behaviors that are consistent with the experimental observations [18].

The simulated hierarchical cache is a regular Q-ary tree with L levels. Q-ary trees have been used extensively for the study of performance issues in hierarchical caches [19]. All the available documents are assumed to originate from an origin server that resides outside the hierarchy, at a conceptual level $L+1$. Each client is co-located with one leaf cache and represents the population of an entire organization. A client j has a request rate λ_j (requests/unit of time) that captures the volume of traffic requested by the local organization and a document popularity distribution p_j. A request follows the unique path from a leaf cache towards the location of the hit, either an intermediate cache or the origin server. The distance from a client is 0 hops for a leaf cache, $l-1$ hops for a l-level cache, and L hops for the origin server. The average number of hops to achieve a request hit is used to compare the performance of the various meta algorithms. For simplicity only hierarchical request forwarding has been used; see [20] for the effect of request peering between sibling caches. As the focus of this work is on the meta algorithms and not on the replacement algorithms themselves, the standard LRU replacement is assumed to run in all caches. The meta algorithm is independent of the employed replacement algorithm and, thus, it is expected that the presented results and observations should apply to some extent to other replacement algorithms as well.

The next results depict the average hit distance under the studied meta algorithms for a storage capacity of S unit sized documents, equally allocated to the n caches of a hierarchy with each cache taking S/n units of storage ($n = \frac{Q^L-1}{Q-1}$ for a full Q-ary tree with L levels). The equal allocation of storage has been employed as a baseline storage allocation strategy, often applied in practice, under which the different meta algorithms are compared; see [20] for algorithms that optimize the allocation of S to the nodes of the hierarchy. Before comparing the different meta algorithms we present a graph that pertains to the effect of the parameter p of the Prob meta algorithm (Prob is the only parametric new algorithm). As it may be seen from Fig. 3 smaller values of p lead to a smaller average hit distance as they filter out more effectively the one-timer references and reduce the amplification of errors. The cost paid for the improved performance is a slower convergence to steady-state. In all the following results we will be using $p = 0.2$ whenever referring to Prob.

Figure 4 compares the different meta algorithms under the assumed Zip-like demand. The following may be noted: (1) LCE has the worse performance across all S; (2) Prob is ranking second across all S; (3) MCD and LCD yield almost equal performance which is always better than LCE and Prob; (4) Filter, although non-memoryless, is outperformed by LCD and closely matched by MCD.[3]

[3]It is believed [21] that Filter's performance could be improved by performing per cache request frequency estimation, thus being able to use the exact aggregate miss stream for a document at an upper level cache to govern local caching decisions. The original version of the algorithm presented in [1] makes local caching decisions based only on the request frequency of a document at the leaf cache that received the original request. This request frequency at the leaf is, however, potentially different from the actual miss stream for this document reaching an upper level cache due to the aggregation of miss streams from multiple leaf caches.

The next results provide some insight into the effects of non-stationary document sets, such as those that are common in the web [22], [23], and are attributed to the creation of new documents and the extinction of old ones. The simulation scenario is the same as the previously described one, with the difference that every W requests, M documents out of the total N in the current document universe are considered extinct (not requested anymore) and are replaced by M new ones. A Zipf-like popularity distribution is used over the current document set. Figure 5 shows the performance of the different algorithms for $W = 1000$ and M in the range [1000,10000]. These values are selected somewhat arbitrarily and it is not suggested that they represent realistic situations; they aim at revealing qualitatively the behavior of the various algorithms under non-stationary document sets. A more realistic evaluation of this effect is obtained by the trace-driven simulations in Sect. IV. Figure 5 show that LCE, which is the worst performer under a stationary document set, progressively approaches the performance of the other algorithms, and eventually outperforms them, as the volatility of the document set increases (with larger M). This behavior owes to the fact that LCE is able to track the new demand more quickly than Prob, MCD, LCD, and Filter, which require multiple requests to bring a copy of a new document to a leaf cache. Naturally, the average hit distance increases, under all algorithms, with increasing values of M. The rate of increase for LCE, however, is smaller than for the other algorithms and, thus, LCE eventually outperforms them.

IV. Trace-driven simulations

To gain a clearer perception of the expected performance of the new meta algorithms, trace driven simulations were conducted using traces from actual operating caches. The trace-based workloads reflect features that are not captured by the employed synthetically generated workloads and, thus, enhance the understanding of the new algorithms and the confidence on the obtained results. Specifically, the traces capture the temporal correlation – also known as locality of reference – which is known to exist in client request patterns [17], [18], [16]. Additionally, embedded in the traces are the volatility of the available document set, which is attributed to the creation of new documents and the extinction of old ones, and the non-stationarity of user access patterns.

A. Description of the traces

The trace-based workloads were created by processing the access logs maintained by the Squid proxy server [24]. These traces were filtered to keep only the requests for cacheable documents. Two types of caches were studied: (1) leaf caches, servicing the client population of an individual organization; (2) root caches of the NLANR hierarchy [3], which receive requests originating from a diverse mixture of connected organizations. The two types of workloads are known to have different properties which are caused by the progressive filtering of requests that pass through successive caches while progressing upwards in the hierarchy [2].

The two leaf caches included in the study belong to the University of Athens (UoA) and the National Technical University

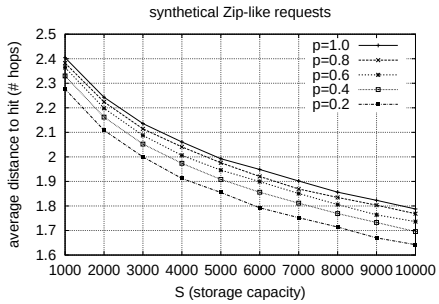

Fig. 3. Average hit distance (in number hops) for Prob with different parameters p, under Zipf-like(0.9) requests and equal request rates, $\lambda_j = 1$ for every client j. LRU runs locally in each cache of the hierarchy. $N = 100000$, $L = 3$, $Q = 2$.

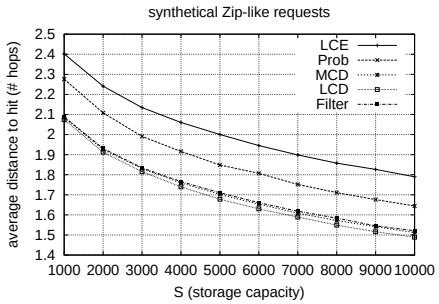

Fig. 4. Average hit distance (in number hops) for LCE, Prob, MCD, LCD, and Filter, under Zipf-like(0.9) requests and equal request rates, $\lambda_j = 1$ for every client j. LRU runs locally in each cache of the hierarchy. $N = 100000$, $L = 3$, $Q = 2$.

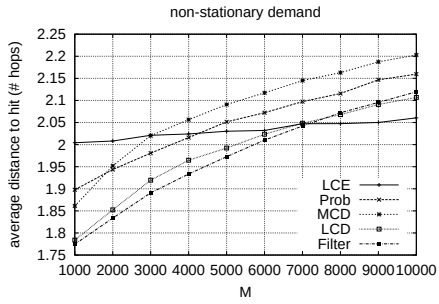

Fig. 5. Average hit distance under non-stationary demand. The document popularity distribution is Zipf-like(0.9) and the client request rates are equal, $\lambda_j = 1$ for every client j. LRU runs locally in each cache of the hierarchy. $N = 100000$, $L = 3$, $Q = 2$.

Trace	Type	Requests	Tot. distinct	1-timers/ Tot. distinct	1-timers/ Requests
uc	root	815194	279375	72%	25%
sv	root	1299024	726075	82%	45%
bo	root	698691	365060	81%	43%
pb	root	709180	405680	84%	48%
sd	root	193769	94457	83%	40%
pa	root	273511	137497	76%	38%
UoA	leaf	282540	41088	71%	10%
NTUA	leaf	580460	234432	73%	30%

TABLE I
TRACE CHARACTERISTICS.

of Athens (NTUA), Greece. UoA is a large metropolitan university including schools of science, medicine, law, and philosophy totalling to more than 45000 students, faculty and staff, while NTUA is a large polytechnic university including a wide variety of engineering departments that amount to around 15000 students, faculty and staff. Both traces include one week worth of requests gathered during the third week of March, 2003.

The root caches correspond to six root NLANR proxy servers that are abbreviated following the NLANR naming conventions as follows: Boulder, Colorado (bo), Palo Alto, California (pa), Pittsburgh, Pennsylvania (pb), Urbana-Champaign, Illinois (uc), San Diego, California (sd), Silicon Valley, California (sv). Each NLANR trace includes one day's worth of requests gathered on Wednesday, 19 February, 2003. A smaller time duration was selected for the root servers as opposed to the leaf servers because the first tend to receive many more requests. Table I summarizes the properties of the various traces used in the simulations.

B. Simulation results

Figure 6 show the simulated performance of the various meta algorithms on the traces of Table I, following an initial warm-up period. A regular hierarchy with $L = 3$, $Q = 2$ and a storage capacity S in the range [1000,10000] were used in all simulations.

The relative ranking of the various meta algorithms changes under different traces, whereas it remains almost stable under the synthetic simulations. The following observations follow from Fig. 6:

- The relative ranking in terms of average hit distance of the three new meta algorithms remains unchanged under all traces and all storage capacities S. The ranking is Prob, MCD, LCD. The LCD meta algorithm appears to be the best performing new one across all experiments, as was also the case with the synthetic simulations.
- LCE, that was constantly the worst performing one under stationary synthetic input (Fig. 4), while it improved under non-stationary demand (Fig. 5) has a relative ranking that changes among different traces. As compared to LCD – the best performing new meta algorithm – LCE is constantly inferior under all six NLANR traces, it is almost as good under the trace UoA, and slightly better under the trace NTUA. Its relative ranking, as compared to the other meta algorithms improves under the various traces, following the order uc, sv, bo, pb, sd, pa, UoA, NTUA, and with increasing S. The behavior may be attributed to two reasons: (1) LCE that occupies more storage by storing copies everywhere is expected to be performing better when a substantial percentage of the requested documents fit in the hierarchy (i.e., with high S/N ratios); (2) as discussed earlier, LCE, improves with the degree of non-stationarity as it catches up to the new demand more quickly than the other algorithms that may require multiple requests to bring a document to a leaf cache. It appears that under a high availability of storage, the gain that LCD achieves by filtering out one-timers and reducing the amplification of replacement errors, is surpassed by the cost paid for requiring multiple requests to bring a document to a leaf cache. With regard to one-timers, the relative performance of LCE/LCD seems to depend on the ratio of (number of one-timers) to (total number of requests), with LCD improving with higher ratios. E.g., in the UoA trace this ratio is relatively small, 10% (see Table I), thus the gain from the filtering of one-timers is rather limited and LCD performs nearly as good as LCE.
- The Filter algorithm has a performance that is relatively close to the best one in some cases (sv) but can also deviate significantly (NTUA). It appears to be constantly inferior to the best performing new one, LCD, across all traces (except for the trace sv and small S).

Based on these observations it may be concluded that

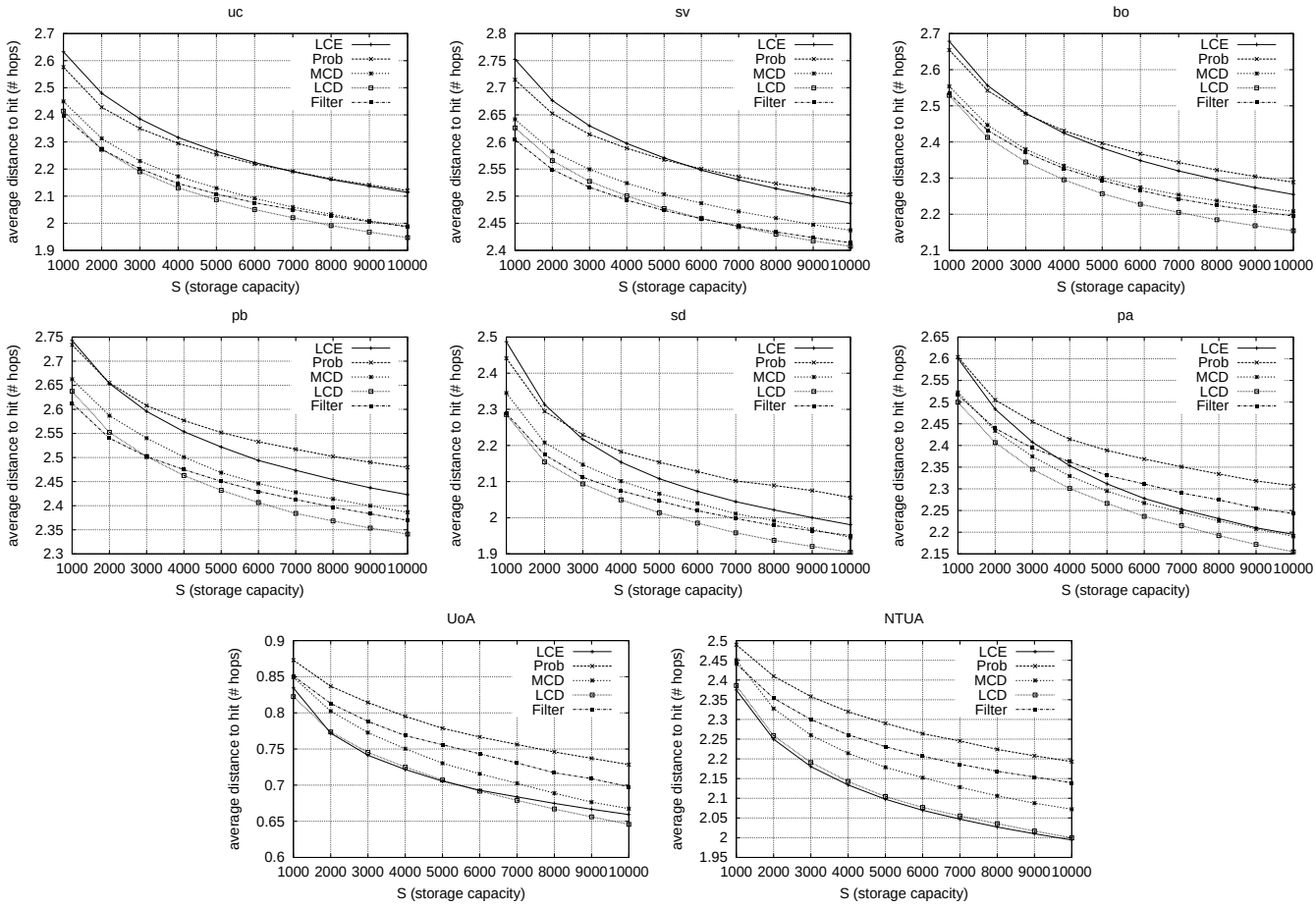

Fig. 6. The average hit distance under the various meta algorithms for the traces of Table I. The simulated topology is a regular hierarchy with $L = 3$ (levels) and $Q = 2$ (node degree).

LCD appears to be an attractive alternative over the currently employed one LCE. As it has been discussed earlier the implementation of LCD requires only minor modifications over the current scheme. It also appears to be performing better than Filter, despite the fact that is much simpler.

V. Other applications: stored music and video over content distribution or peer-to-peer networks

Although originally motivated by hierarchical web caching, the presented meta algorithms have applications to CDN or even P2P networks. Indeed, typical CDN nodes that cache content according to the received request, employ replacement algorithms to self organize [6]. Similarly, some recent research into the evolution of P2P systems proposes P2P nodes that lend storage capacity to the P2P community, allowing it to operate under a replacement algorithm [25]. Although CDN and P2P networks are flat structured, it is conceivable that multi-hop search paths might be formed by forwarding misses towards the direction of a node that is known to be holding the requested document. Such multi-hop search paths on the flat structured overlays are a lot similar to multi-hop branches in hierarchical caches, like the ones studied here. LCD, MCD or Probe could potentially be applied to these multi-hop paths to move content more cautiously.

The presented meta algorithms become particularly relevant when the CDN or P2P application is used for the transfer of stored music files or video clips. A recent large scale characterization of http traffic from Saroiu et al. [26] has shown that more than 75% of internet traffic is generated by P2P applications that employ the http protocol, such as KaZaa and Gnutella. The median document size of these P2P systems is 4MB which represents a thousand-fold increase over the 4KB median size of typical web documents. Furthermore, the access to these documents is highly repetitive and skewed towards the most popular ones thus making them highly amenable to caching as demonstrated by the authors of [26].

Similar is the case with stored audio and video files distributed by either standard web servers or dedicated video on demand (VoD) servers. Such document are usually short videos (advertisements or news) and amount to around 1 MB [27]. Although a 1997 study [28] found that such traffic represents a rather limited percentage of the total aggregate traffic in the internet, the percentage was shown to have been increased significantly by a follow-up measurement study just a few years later [29]. This trend seems to have persisted, as recent studies (2001) have shown that stored video and audio files have by now become almost pervasive [27].

Handling such content in large quantities is challenging. The thousand-fold increase in size as compared to html content,

may lead to the exhaustion of the storage capacity of a cache or a CDN node, even under a low price of storage that allows for the deployment of multi-gigabyte disks. This situation resembles much the presented simulation scenarios that assume that only a fraction of the requested content fits in the cache. It is exactly in this storage-limited scenario that the presented meta algorithms (most notably LCD) clearly outperform the storage consuming LCE policy. Unfortunately, a more realistic study of such applications is inhibited by the unavailability of publicly released P2P and video workload traces.

VI. Load Balancing

Recently there has been some controversy regarding the effectiveness of hierarchical caches [30], owing mainly to the poor hit ratios observed at upper level caches and attributed to the "filtering effect" [2] (also known as "trickle-down effect" [31]). The filtering effect is a natural consequence of the currently employed LCE algorithm. Under LCE the most popular documents gather at the leaf caches, where they filter the great majority of requests. Upper level caches store much less popular documents and also some of the popular ones which, however, receive only very few requests due to the filtering of requests by lower level copies. To connect with the observed low hit ratios, one must note that the hit ratio depends strongly on the skewness of the demand. The filtering effect transform a skewed Zipf-like demand into a progressively uniform demand as progressing upwards in the hierarchy [2]; this leads to poor hit ratios at upper levels.

From the perspective of the *load* that is imposed on each cache, the filtering effect leads to the servicing (hits) of most of the requests at the lower level caches, leaving upper level caches severely underutilized. We count as load only the requests that lead to a hit and disregard the relative smaller load imposed from a miss.[4] A nearby hit at a low level cache is of course desirable since it is expected to yield a small propagation delay thus avoiding accessing caches that are further away in the hierarchy. This, however, does not always lead to a small total delivery delay since the processing at a low level cache (involves accessing a document from main memory or disk and transmitting it) might take too long due to the overloading of low level caches, as is the case with the filtering effect. Performance studies of real web proxies (including Squid, Harvest, CERN httpd) have shown that the processing delay remains constant only under small to medium load whereas it increases fast under high load as the system approaches a thrashing state [32], [33], [34].

To address the filtering effect, and the imbalance in the handling of load that stems from it, we present a simple load balancing mechanism that is based on the idea of meta algorithms and runs solely on local information. The fact that the algorithm is fully distributed gives a significant advantage over centralized algorithms that require the exchange of messages (to report the current level of load, redirect requests, etc.). The idea is to have each cache monitor its load and accept new copies of documents only when this load is below a predetermined threshold that marks the start of the overload region. Not accepting a new document means that the requests for this document will have to be serviced by an upper level cache, thus, effectively allowing for some load to be removed from the overutilized leaf caches and flow towards the underutilized upper level caches. This simple mechanism does not allow all popular documents to gather at the leaf caches. Instead it requires that some of them be cached only at upper level caches, thus, prohibiting the development of the filtering effect. A cache starts caching new documents again as soon as its load falls beneath the threshold. This can either happen due to the stochastic nature of request arrival times or can be a result of changing demand patterns. For example a cache that reaches the threshold after having cached enough popular documents will return below the threshold when its documents cease to be popular (e.g., have expired or substituted by new ones). Then the cache will start accepting new copies again.

The following experiments evaluate the effectiveness of the load balancing mechanism using data traces. Although it may be applied to all discussed meta algorithms, we have chosen to present results that assume the LCE algorithm, as this is the one currently employed in practice. We define the LCE-LB algorithm as a variation of LCE that keeps copies at all intermediate caches on the reverse path provided that a cache has not reached its load threshold TH. In an actual implementation, the system load would be queried[5] upon the reception of a new document and a copy would be stored only if this load didn't exceed TH. The exact selection of TH depends on the architecture/implemetation of each proxy server but can generally be identified (see the previously cited works [32]-[34]).

In our simulation environment we let each cache estimate its load by measuring the number of hits that have occurred in the recent past. Specifically the load of cache v on the mth time slot is estimated from the following linear recursive filter: $load_v(m) = g \cdot load_v(m-1) + (1-g) \cdot h(m)$, where $h(m)$ denotes the number of hits in the current time slot m. The "gain" of the filter g determines the effect of previous hits on the current load (i.e., accounts for hits that have occurred in the past but are still under service); $g = 0.9$ is used in the following experiments.

A common load threshold is used for all caches. The threshold is selected relatively to the total request rate from all clients. Specifically, its value is set to: $TH = \frac{\sum_j \lambda_j}{k \cdot n}$ where k is a parameter controlling the intensity of the desired load balancing, and n is the number of caches in the hierarchy. The definition of the threshold is justified as follows. The maximum load that could be imposed on a single cache can not exceed the sum of request rates from all clients, $\sum_j \lambda_j$; this would happen if every request resulted in a hit and all hits occurred at the root cache. Setting the threshold to $\frac{\sum_j \lambda_j}{n}$ would require a perfect load balancing over this extreme

[4]The processing imposed by a miss relates to the establishment of TCP connections and the lookup for the requested document. The same work applies however to a hit which, in addition, must retrieve the requested document from either the main memory or the disk. The additional processing for accessing and streaming a document dominates all other processing, especially for large documents.

[5]In the UNIX operating system this could be easily done by using the `ps` command.

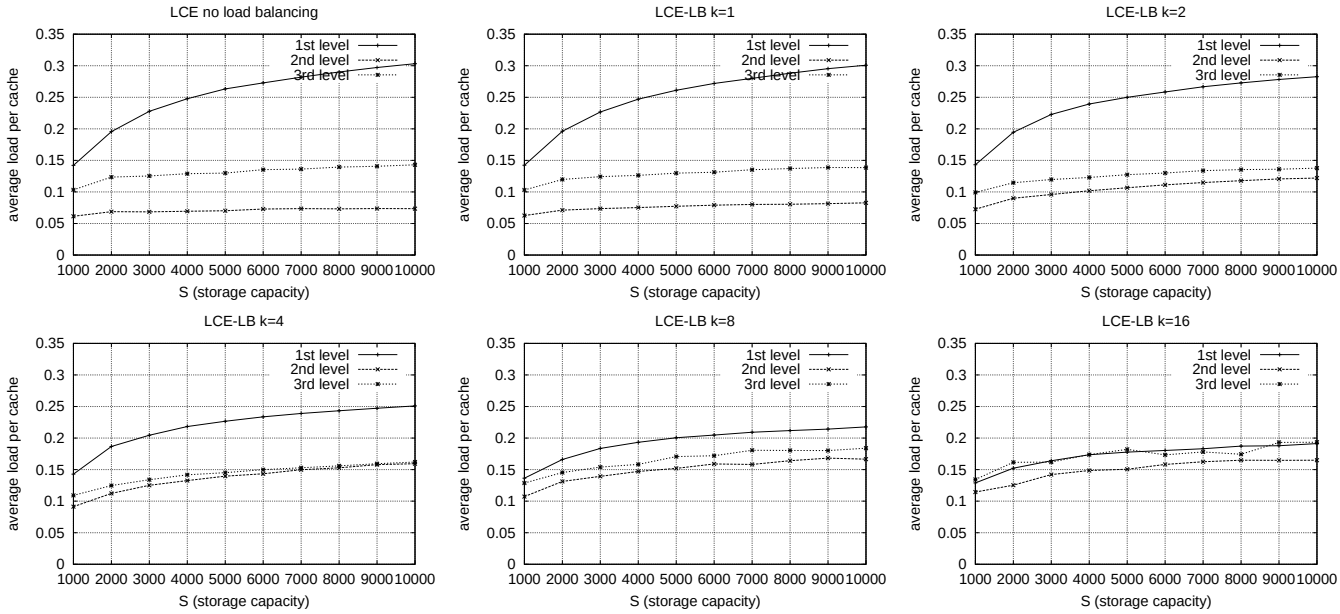

Fig. 7. Average load per cache under the trace sd for the following cases: (1) LCE without load balancing; (2) LCE-LB with $k = 1, 2, 4, 8, 16$. The simulated topology is a regular hierarchy with $L = 3$ (levels) and $Q = 2$ (node degree).

scenario. Such a threshold, however, would be too loose for more realistic cases because: (1) all requests do not lead to hits (this would require too much storage); (2) hits occur in all caches not only the root. Thus we further divide by k to define more useful load thresholds.

Figure 7 shows the average load per level under the trace sd[6] for the following cases: (1) LCE without load balancing; (2) LCE-LB with $k = 1, 2, 4, 8, 16$. One may verify from the plots that as k increases the load tends to be more evenly distributed among different levels. Specifically, as compared to the LCE algorithm that may assign to level-1 caches as much as four times the load than is assigned to level-2 caches (for $S = 10000$), the various LCE-LB examples limit the asymmetry in the handling of load, achieving an almost even distribution under $k = 16$, where level-1 caches get no more that 15% extra load as compared to level-2 caches. Notice that the distribution of load under LCE-LB with $k = 1$ is almost identical to the one under LCE. This is due to the fact that the load constraint under $k = 1$ is too loose, almost equal to the maximum load that is assigned under LCE; the load balancing mechanism becomes effective for values of k larger than 2.

Figure 8 illustrates the effect of various degrees of load balancing on the average hit distance. As it may be anticipated, the average hit distance increases with the intensity of load balancing (with larger k) as more requests get serviced by caches upper in the hierarchy. The maximum performance gap between the load unconstrained LCE and the most load constrained LCE-LB with $k = 16$ is almost 10%. This means that a 10% penalty in average hit distance (which translates to an equally increased propagation delay) may yield an almost even distribution of load among the different caches. In terms of user perceived delay, the 10% increase in the propagation delay may easily be outweighted by a substantially reduced

[6]Similar results were obtained under the other traces as well, but are not presented due to space limitations.

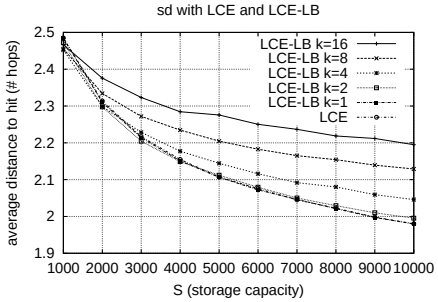

Fig. 8. Average hit distance under the trace sd for the following cases: (1) LCE without load balancing; (2) LCE-LB with $k = 1, 2, 4, 8, 16$. The simulated topology is a regular hierarchy with $L = 3$ (levels) and $Q = 2$ (node degree).

processing delay as a result of accessing uncongested caches. Thus, networks that may tolerate the 10% increase in network traffic may achieve a reduced overall delay by employing the load balancing mechanism. These results appear to be consistent with our previous results pertaining to the tradeoff between load balancing and hit distance, for the case of object replication [20].

VII. Conclusions and future work

This work has studied the performance of various meta algorithms that are responsible for deciding whether a new document will be accepted in a cache following a cache miss. Three new meta algorithms have been proposed and compared against the de facto one that always accepts copies and a recently proposed one from the literature. The design principles of the new algorithms as well as the presented numerical results suggest that these algorithms may prove useful in a variety of situations. In fact, the best performing new meta algorithm, LCD, seems to be performing very well under all studied scenarios. We are currently working towards the development of appropriate analytical models for

the performance of the various meta algorithms. Hopefully, such models will enhance the understanding of the new algorithms gained from this experimental study and help in identifying more clearly their performance under the numerous parameters.

A second contribution of this work has been the description and evaluation of a simple load balancing mechanism for hierarchical caches that utilizes the concept of meta algorithms. It has been shown that with only minor (local) modifications to the currently employed caching strategy, it is possible to limit the extent of the so called "filtering effect" that may overutilize leaf caches while underutilizing higher level caches. The concept of meta algorithms could have been used for other interesting purposes as well. Different meta algorithms could have been used to provide service differentiation to clients that have different service agreements with a content distributor (e.g., a CDN). One example would involve the use of Prob with different parameters p depending on the service level of a client; large p would be used to allow premium clients to maintain multiple copies of their working sets as opposed to non premium clients that would be offered a lower p.

Finally, as the presented algorithms are generic in nature, they might be employed for the distribution of content other than the standard web content (html, images), such as large stored audio and video files. In fact, it has been suggested that the employed algorithms need not be confined to hierarchical caching but could potentially extend to CDN or P2P networks. When handling large content, the judicious filtering of one-timer references and the reduction of amplification error offered by the proposed meta algorithms might prove even more important than in the case of web content as media files may exhaust the storage capacity of even a large node. We plan on investigating such issues in the future.

REFERENCES

[1] Hao Che, Ye Tung, and Zhijun Wang, "Hierarchical web caching systems: Modeling, design and experimental results," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, Sept. 2002.

[2] Carey Williamson, "On filter effects in web caching hierarchies," *ACM Transactions on Internet Technology*, vol. 2, no. 1, Feb. 2002.

[3] *The IRCache project*, `http://www.ircache.net/`.

[4] *The JANET web cache*, `http://wwwcache.ja.net/index.html`.

[5] James H. Hester and Daniel S. Hirschberg, "Self-organizing linear search," *ACM Computing Surveys*, vol. 17, no. 3, pp. 295–311, Sept. 1985.

[6] Yan Chen, Lili Qiu, Weiyu Chen, Luan Nguyen, and Randy H. Katz, "Efficient and adaptive web replication using content clustering," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 6, Aug. 2003.

[7] Madhukar R. Korupolu and Michael Dahlin, "Coordinated placement and replacement for large-scale distributed caches," in *Proceedings of the IEEE Workshop on Internet Applications*, June 1999, pp. 62–71.

[8] Xueyan Tang and Samuel T. Chanson, "Coordinated en-route web caching," *IEEE Transactions on Computers*, vol. 51, no. 6, pp. 595–607, June 2002.

[9] Konstantinos Psounis and Balaji Prabhakar, "Efficient randomized web-cache replacement schemes using samples from past eviciton-times," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 441–454, Aug. 2002.

[10] David Starobinski and David N. C. Tse, "Probabilistic methods for web caching," *Performance Evaluation*, vol. 46, no. 2-3, pp. 125–137, 2001.

[11] E. G. Coffman and P. J. Denning, *Operating Systems Theory*, Prentice-Hall Inc., 1980.

[12] Daniel D. Sleator and Robert E. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, vol. 28, no. 2, pp. 202–208, 1985.

[13] A.V. Aho, P.J. Denning, and J.D. Ullman, "Principles of optimal page replacement," *Journal of the ACM*, vol. 18, no. 1, pp. 80–93, Jan. 1971.

[14] P.A. Franaszek and T.J. Wagner, "Some distribution-free aspects of paging algorithm performance," *Journal of the ACM*, vol. 21, no. 1, pp. 31–39, Jan. 1974.

[15] Anirban Mahanti, Carey Williamson, and Derek Eager, "Traffic analysis of a web proxy caching hierarchy," *IEEE Network Magazine*, vol. 14, no. 3, pp. 16–23, May 2000.

[16] Anirban Mahanti, Derek Eager, and Carey Williamson, "Temporal locality and its impact on web proxy cache performance," *Performance Evaluation*, vol. 42, pp. 187–203, 2000.

[17] Martin F. Arlitt and Carey Williamson, "Internet web servers: Workload characterization and performance implications," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 631–645, Oct. 1997.

[18] Lee Breslau, Pei Cao, Li Fan, Graham Philips, and Scott Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, New York, Mar. 1999.

[19] Pablo Rodriguez, Christian Spanner, and Ernst W. Biersack, "Analysis of web caching architectures: Hierarchical and distributed caching," *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, Aug. 2001.

[20] Nikolaos Laoutaris, Vassilios Zissimopoulos, and Ioannis Stavrakakis, "Storage capacity allocation algorithms for hierarchical content distribution," [submitted work].

[21] Hao Che, *personal communication, April 2003.*

[22] F. Douglis, A. Feldman, B. Krishnamurthy, and J.C. Mogul, "Rate of change and other metrics: a live study of the world wide web," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.

[23] B.E. Brewington and G. Cybenko, "How dynamic is the web?," *WWW9/Computer Networks*, vol. 33, no. 1-6, pp. 257–276, 2000.

[24] Duane Wessels and K. Claffy, "ICP and the Squid web cache," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 3, Apr. 1998.

[25] Jussi Kangasharju, Keith W. Ross, and David A. Turner, "Optimal content replication in P2P communities," 2002, in submission.

[26] Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy, "An analysis of internet content delivery systems," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Dec. 2002.

[27] Maureen Chesire, Alec Wolman, Geoffrey M. Voelker, and Henry M. Levy, "Measurement and analysis of a streaming-media workload," in *Proceedings of USITS*, 2001.

[28] S. D. Gribble and E.A. Brewer, "System design issues for internet middleware service: Deductions from a large client trace," in *Proceedings of the First USENIX Symposium on Internet Technologies and Systems*, Dec. 1999.

[29] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy, "Organization-based analysis of web-object sharing and caching," in *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems*, Oct. 1999.

[30] Sandra G. Dykes and Kay A. Robbins, "Limitations and benefits of cooperative caching," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, Sept. 2002.

[31] R. Doyle, J. Chase, S. Gadde, and A. Vahdat, "The trickle-down effect: Web caching and server request distribution," in *Proceedings of the 6th Int. Workshop on Web Caching and Content Distribution*, Boston, MA, June 2001.

[32] Jussara Almeida and Pei Cao, "Measuring proxy performance with the wisconsin proxy benchmark," *Computer Networks and ISDN Systems*, vol. 30, pp. 2179–2192, 1998.

[33] P. Barford and M. E. Crovella, "Measuring web performance in the wide area," *Performance Evaluation Review*, Aug. 1999.

[34] Alex Rousskov and Valery Soloviev, "A performance study of the Squid proxy on http/1.0," *World Wide Web*, vol. 2, no. 1, pp. 47–67, 1999.