A Survey of Web Cache Replacement Strategies

STEFAN PODLIPNIG AND LASZLO BÖSZÖRMENYI

University Klagenfurt

Web caching is an important technique to scale the Internet. One important performance factor of Web caches is the replacement strategy. Due to specific characteristics of the World Wide Web, there exist a huge number of proposals for cache replacement. This article proposes a classification for these proposals that subsumes prior classifications. Using this classification, different proposals and their advantages and disadvantages are described. Furthermore, the article discusses the importance of cache replacement strategies in modern proxy caches and outlines potential future research topics.

Categories and Subject Descriptors: A.1 [**Introductory and Survey**]; B.3.2 [**Memory Structures**]: Design Styles—*Cache memories*; C.2.4 [**Computer-Communication Networks**] Distributed Systems; C.2.5 [**Computer-Communication Networks**]: Local- and Wide-Area Networks—*Internet*

General Terms: Algorithms

Additional Key Words and Phrases: Web caching, replacement strategies

1. INTRODUCTION

Since its introduction, the Web has been constantly growing and so has the load on the Internet and Web servers. To overcome these obstacles, different techniques, like caching, have been introduced. Web caching has proven to be a valuable tool.

Three features of Web caching make it attractive to all Web participants, including users, network managers, and content creators [Davison 2001]:

—Caching reduces network bandwidth usage.

—Caching reduces user-perceived delays.

—Caching reduces loads on the origin server.

One central problem in Web caching is the cache replacement strategy. Cache replacement refers to the process that takes place when the cache becomes full and old objects[1] must be removed to make space for new ones.

Although the amount of literature and information on Web caching is staggering, there exist only a few survey articles and books. A concise survey of many different topics (including cache replacement) was given in Wang [1999]. A survey of techniques and trends in Web caching was given in Barish and Obraczka [2000]. There are a few books about Web caching. Wessels [2001] presented a practical introduction to Web caching. Krishnamurthy and Rexford [2001] gave a state-of-the-art

[1]We use the term *Web object* for all possible objects (HTML pages, images, videos, etc.) that can be stored in a proxy cache. It is more general than *Web document*, a denotation used in some papers.

Author's addresses: Institute of Information Technology, University Klagenfurt, Universitätsstrasse 65-67, A-9020 Klagenfurt, Austria, e-mail: {spodlipn,laszlo}@itec.uni-klu.ac.at.

overview of Web caching. Rabinovich and Spatscheck [2002] presented a very readable overview of Web caching with interesting critiques.

There exist no single in-depth survey of cache replacement strategies. Krishnamurthy and Rexford [2001] and Rabinovich and Spatscheck [2002] presented short discussions of cache replacement. They listed different arguments against the importance of cache replacement and argued that simple strategies are sufficient. We will comment on this in Section 4. Small overviews of replacement strategies have been presented, for example, in Aggarwal et al. [1999], Arlitt et al. [2000], Bahn et al. [2002], Cao and Irani [1997], Chang et al. [1999], Jin and Bestavros [2000], Tatarinov [1998b], and Williams et al. [1996]. A concise overview of some replacement strategies was given in Wang [1999]. In the following we present a comprehensive survey.

2. CLASSIFICATION OF REPLACEMENT STRATEGIES

To present the different proposals in a structured way, we want to give a classification of replacement strategies. Such classifications were also used by other authors. Before we describe the used classification, we summarize the important factors (characteristics) of Web objects that can influence the replacement process (most of these factors are described in Krishnamurthy and Rexford [2001]):

—recency: time of (since) the last reference to the object;

—frequency: number of requests to an object;

—size: size of the Web object;

—cost of fetching the object: cost to fetch an object from its origin server;

—modification time: time of (since) last modification;

—(heuristic) expiration time: time when an object gets stale and can be replaced immediately.

These factors can be incorporated into the replacement decision. Most of the proposals in the literature use the first four factors.

A first classification of replacement strategies was given in Aggarwal et al. [1999]. They proposed three categories:

—*Direct extensions of traditional strategies.* This category subsumes traditional strategies known from other areas (e.g., database buffer management, paging) and extensions thereof.

—*Key-based replacement strategies.* Replacement strategies in this category sort objects upon a primary key (factor). Ties are broken based on secondary key, tertiary key, etc.

—*Function-based replacement strategies.* The idea in function-based replacement is to use a potentially general function derived from the different factors described above.

A similar classification was given in Krishnamurthy and Rexford [2001]: one-level strategies that use one factor, two-level strategies that use a primary and secondary factor, and combination strategies that use a weighted approach for the combination of factors.

There are two major problems with these proposals. First, the first two classes could be merged as every traditional algorithm can be regarded as a key-based strategy using one key (factor). A number of strategies apply various additional techniques (more lists, etc.), that is, they are not only key-based. Second, randomized strategies can not be classified according to the above described classification. The pure random strategy can not be classified into any of these categories. It uses no key and no function. Some sophisticated random strategies can be combined with key-based decisions or function-based decisions. Therefore, they can be classified into more than one category.

A different classification was given in Jin and Bestavros [2000]:

—Recency-based strategies: strategies that incorporate recency (and size and/or cost) into the replacement process.

—Frequency-based strategies: strategies that incorporate frequency (and size and/or cost) into the replacement process.

—Recency/frequency-based strategies: strategies that consider both recency and frequency under fixed or variable cost/size assumptions.

This classification has the advantage that it distinguishes between recency and frequency being by far the most important factors. Furthermore, they represent different ways of describing the importance of a Web object. This classification still has a problem with randomized strategies. It also has a problem with strategies that do not use recency and frequency.

In the following, we propose a combined classification with the following classes:

—recency-based strategies;

—frequency-based strategies;

—recency/frequency-based strategies;

—function-based strategies;

—randomized strategies.

The following justifications can be given for the above classification:

—Recency and frequency are important but very different factors. Therefore, separate classes are advantageous for strategies that use these factors independently. This supports a more consistent critique of similar strategies. Furthermore, some strategies mix recency and frequency and should be treated in their own class.

—Some strategies evaluate a specific function incorporating different factors. Mostly they use weighting parameters for the different factors. Although they could be classified into one of the first three classes, a separate class supports a more consistent review and critique of this special sort of strategies.

—Randomized strategies constitute a different (nondeterministic) approach to cache replacement. Therefore, they should be treated in a separate class.

In the following, we use this classification to survey different proposals for cache replacement.

We do not explicitly deal with the place of replacement. Caches may reside at the client, at a proxy near to the client, or at the server. Replacement is an important issue in all these places. In the following, we will not distinguish the place as most replacement strategies were proposed for proxy caches. We will comment explicitly on this topic if necessary.

3. SURVEY OF REPLACEMENT STRATEGIES

This section presents a comprehensive overview of different proposals for cache replacement strategies. We assume the following simple replacement process. On a cache miss, the cache acquires and stores the requested object. If the size of all cached objects exceeds the given cache size, the cache evicts a certain number of objects. In practical implementations, the cache uses two marks H (high watermark) and L (low watermark), with $H > L$, to guide the replacement process. If the size of all cached objects exceeds H, the cache evicts objects until the size of the remaining objects is below L. It should be noted that some authors assume that the cache stores initially only descriptive information about the object. During the replacement process the cache decides whether this new object is accepted (physically) into the cache and some objects are removed or the new object is not accepted and the cache content remains unmodified.

To represent the replacement strategies in a consistent way, we use a set of uniform identifiers for often used keys (factors). These identifiers are given in Table I. Some replacement strategies use additional special identifiers.

In the following, we mainly survey the applied caching literature. This means that these proposals were evaluated empirically (in simulated or real environments). There exists another scientific camp that studies caching from a more theoretical point of view. The proposed

Table I. Often Used Identifiers

Identifier	Description
s_i	Size of object i
t_i	Last request time of object i
T_i	Time since last request to object i
f_i	Number of past requests to object i
l_i	Access latency for object i
c_i	Cost to fetch object i from its origin server; c_i has a more general meaning than l_i (e.g., number of networks hops, latency)

algorithms are compared to optimal algorithms via so-called competitive analysis. A large part of this literature handles known strategies (like LRU, FIFO) and their worst-case performance on all possible request sequences. We will comment on some results later on.

There are also new proposals for cache replacement in this theoretical research. Most of these algorithms were proposed for scenarios with equally sized objects (paging). Some algorithms were proposed for the more general case where objects can have different size and/or different costs, etc. In the following we will comment on some of these algorithms and present them if it is possible. Some algorithms require a more exhaustive description of the assumed model and the proposed procedure. Although their theoretical performance bounds are known and often better than the performance bounds of more simple algorithms, it is questionable if these algorithms will be implemented in practical environments.

We present for each replacement algorithm a short synopsis that describes the essence of the proposed algorithm. We will not try to describe any algorithm in detail because this goes beyond the scope of this survey. The interested reader can contact the original literature to get information about the incentives of the authors and extended aspects of the algorithms.

Note that we have included all proposals we are aware of. Clearly there will be further replacement algorithms we are not aware of, but we have tried to be as exhaustive as possible. Furthermore, there exist replacement algorithms that are used in other domains (e.g., database systems) that were not tested in the Web domain. We did not include such algorithms.

3.1. Recency-Based Strategies

These strategies use recency as a main factor. Most of them are more or less extensions of the well-known LRU strategy. LRU has been applied successfully in many different areas. LRU is based on the locality of reference seen in request streams. Locality of reference characterizes the ability to predict future accesses to objects from past accesses. There are two main types of locality: temporal and spatial. Temporal locality refers to repeated accesses to the same object within short time periods. It implies that recently accessed objects are likely to be accessed again in the future. Spatial locality refers to access patterns where accesses to some objects imply accesses to certain other objects. It implies that references to some objects can be a predictor of future references to other objects. Recency-based strategies exploit the temporal locality seen in Web request streams.

—*LRU.* This strategy removes the least recently referenced object. It is widely used in different areas (e.g., database buffer management, paging, disk buffers).

—*LRU-Threshold* [Abrams et al. 1995]. An object i is not cached when s_i exceeds a given threshold. Otherwise this strategy works like LRU.

—*Pitkow/Recker's strategy* [Pitkow and Recker 1994]. This uses LRU. Objects that are requested on the same day are differentiated by their size and the largest files are removed first.

—*SIZE* [Williams et al. 1996]. This strategy removes the biggest object. The LRU strategy is applied to objects with the same size. One variant is LOG2-SIZE which uses $\lfloor log_2(s_i) \rfloor$ instead of s_i.

—*LRU-Min* [Abrams et al. 1995]. This is a variant of LRU that tries to minimize the number of documents replaced. Let L_o and T denote, respectively, a list and a threshold. (1) Set T to S, where S is the size of the requested document.

(2) Set L_0 to all documents whose size is equal to or larger than T. (L_o may be empty.) (3) Apply LRU to L_0 until the list is empty or the free cache space is at least T. (4) If the free cache space is not at least S, set T to $\frac{T}{2}$ and go to step (2).

—*EXP1* [Reddy and Fletcher 1998]. LRU uses the time period between the current time and the last request time to weight the importance of an object. This can be extended to an algorithm that uses more time periods. EXP1 chooses object i if $\frac{1}{\mu_i} = min\{\frac{1}{\mu_j}\}$ with $j = 1, \ldots, N$ (set of N objects). μ_i is calculated by applying exponential smoothing techniques to the successive time periods between different requests to object i. For further datails see Reddy and Fletcher [1998].

—*Value-Aging* [Zhang et al. 1999]. Value aging uses the following formula for replacement ($V_{new}(i)$ is updated at each request to object i):

$$V_{new}(i) = V_{old}(i) + C_t * \sqrt{\frac{C_t - t_i}{2}}, \quad (1)$$

where C_t is the current time.

—*HLRU* [Vakali 2000]. HLRU introduces a scheme to support the history of the number of references to a specific Web object. Let $r_1, r_2, \ldots, r_n$ be the requests for cached Web objects at the times $t_1, t_2, \ldots, t_n$. A history function is defined as follows:

$$hist(x,h) = \begin{cases} t_i & \text{if there are exactly } h-1 \text{ references between } t_i \text{ and } t_n, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

$hist(x,h)$ defines the time of the past hth reference to a specific cached object x. HLRU replaces the object with the maximum *hist* value. If there are many cached objects with $hist = 0$, the original LRU is considered for replacement.

—*PSS(Pyramidal Selection Scheme)* [Aggarwal et al. 1999]. This strategy makes a pyramidal classification of objects depending upon their size. All objects of class i have sizes ranging between 2^{i-1} and $2^i - 1$. Thus, there are $N = \lceil log_2(M+1) \rceil$ different classes, where M is the cache size. Each class has a separate LRU list. Whenever there is a replacement, the values of the least recently used objects in each class are compared. PSS chooses object i if its value $s_i \Delta T_i$ is the largest one among all values of these objects. ΔT_i is the number of accesses since the last time object i was requested.

—*LRU-LSC* [Hosseini-Khayat 1997].[2] This strategy uses a normal LRU list to determine the "activity" of the different cache objects. During replacement, objects with less activity (starting from the end of the LRU list) are placed in a second list as long as the total size of this new list is less than ΘB. B is the total cache size and $\Theta (0 < \Theta < 1)$ a threshold parameter that determines the fraction of the cache list to be moved to the new list. It can be set statically or dynamically. The new list is sorted according to some value $spc_i = c_i/s_i$. t_i (activity) is used as a tie breaker. Then objects are removed from this new list until their accumulated size subtracted from the size of all cached objects is lower than a specific mark.

—*Partitioned Caching* [Murta et al. 1998]. This strategy classifies all Web objects according to their size into three different groups (small, medium, large). The thresholds for this classification are derived from prior Web traces. Each class has its own cache space and is managed independently from the other two classes with the LRU strategy.[3] Let S_C be the size of the cache and S_{c1}, S_{c2}, S_{c3} (with $S_{c1} + S_{c2} + S_{c3} = S_C$) the size of the three partitions (small = 1, medium = 2, large = 3). Murta et al. [1998] showed experimentally that the

[2]Hosseini-Khayat [1997] discussed generalized caching in a very exhaustive way. Many theoretical aspects of different algorithms (on-line or off-line) were discussed. We limit our discussion to the most important algorithm that can be implemented in practical environments

[3]Note that this proposal is very general and could use any replacement strategy in each class. LRU is a simple and fast algorithm and therefore used in Murta et al. [1998].

following should hold: $S_{c1} < S_{c2} < S_{c3}$.

The advantages of recency-based strategies are as follows:

—They consider temporal locality, as a main factor. As Web request streams usually exhibit some sort of temporal locality, this is an advantageous procedure. Furthermore, these strategies are rather adaptive to workload changes (e.g., new very popular objects).

—They are simple to implement and fast. Most of these strategies use an LRU list. New requested objects are inserted at the head of the list. On a hit the object is removed from its current position and inserted at the head. Replacement takes place at the end of the list. So insertion and deletion add low complexity. Furthermore, searching can be supported by hashing techniques.

The disadvantages are as follows:

—Simple LRU variants do not combine recency and size in a useful, balanced way. As Web objects are usually of different size, size should be considered at every replacement. The SIZE strategy does this; however, it is too aggressive as it places too much emphasis on the size of objects. LRU-Min is also focused more on size. The PSS strategy is a laudable exception, as it is simple to implement, is fast, and combines size and recency in a more balanced way. Additionally, a properly parameterized partitioned LRU caching strategy can be very simple and fast.

—They do not consider frequency information. This could be an important indicator in more static environments.

Nevertheless recency-based strategies are often used in proxy caches. Many cache vendors use a simple LRU strategy or variants of it. We will comment on this in Section 4.

3.2. Frequency-Based Strategies

These strategies use frequency as a main factor. Frequency-based strategies are more or less extensions of the well known LFU strategy. They are based on the fact that different Web objects have different popularity values and that this popularity values result in different frequency values. Frequency-based strategies track these values and use them for future decisions. It should be noted that LFU (and its extensions) can be implemented in two different forms:

—*Perfect LFU*. Perfect LFU counts all requests to an object i. Request counts persist across replacements. On the one hand, this assures that the request counts represent all requests from the past. On the other hand, these statistics have to be kept for all objects seen in the past (space overhead).

—*In-Cache LFU*. With in-cache LFU, the counts are defined for cache objects only. Although this does not represent all requests in the past, it assures a simpler management (less space overhead).

In the following we assume in-cache LFU:

—*LFU.* Removes the least frequently requested object.

—*LFU-Aging* [Arlitt et al. 2000]. With LFU, objects that were very popular during one time period can remain in the cache even when they are not requested for a long time period. This is due to their high frequency count. To avoid this cache pollution, an aging effect can be introduced. LFU-Aging uses, therefore, a threshold. If the average value of all frequency counters exceeds this threshold, all frequency counters are divided by 2. Furthermore, this strategy uses a maximal value for the frequency counters.

—*LFU-DA* [Arlitt et al. 2000]. The performance of LFU-Aging depends heavily on the chosen parameters (threshold, maximal frequency value). LFU-DA tries to remove this problem. A request for object i triggers a (re)calculation of its value K_i:

$$K_i = f_i + L, \tag{3}$$

where L is an aging factor. L is initialized to zero. LFU-DA chooses the object with the smallest K_i-value. The value of this object is assigned to L.

—*α-Aging* [Zhang et al. 1999]. This is an explicit aging method with a periodic aging function

$$f(v) = \alpha * f_i \quad 0 \leq \alpha \leq 1. \tag{4}$$

Each virtual clock tick (e.g., 1 hr) the value of every object is decreased to α times its original value. Each hit causes $f(v)$ to be increased by one. Changing α from 0 to 1, one can obtain a spectrum of algorithms ranging from LRU ($\alpha = 0$) to LFU ($\alpha = 1$).

—*swLFU (Server-Weighted LFU)* [Kelly et al. 1999]. This strategy uses a weighted frequency counter. The weight w_i for an object i indicates how much the server of i appreciates the caching of object i. Therefore, the server can influence the caching of object i. The LRU strategy is applied to objects with the same weighted frequency value, that is, LRU is used as a tie breaker. One extension to this original strategy is A-swLFU (Aged-swLFU). This strategy evicts the LRU-object on every k replacement. $k = 0$ corresponds to the original swLFU strategy. $k = 1$ corresponds to LRU. $k > 1$ gives a mixture of recency and frequency.

The advantage of the frequency-based strategies is as follows:

—They consider the frequency of access. This is valuable in static environments where the popularity of objects does not change very much over a specific time period (day, week).

The disadvantages are as follows:

—*Complexity*. LFU-based strategies require a more complex cache management. LFU can be implemented, for example with a priority queue.

—*Cache pollution*. Frequency counts are to static for dynamic changes in the workload. Therefore, aging was introduced. But aging is nothing but a recency-based technique. It is questionable if sophisticated aging techniques are better than simple recency-based techniques in dynamically changing environments. Furthermore, they add complexity to the replacement process.

—*Similar values*. Many objects can have the same frequency count. In this case, a tie breaker factor is needed.

3.3. Recency/Frequency-Based Strategies

These strategies use recency and frequency and maybe further factors to find an object for replacement:

—*SLRU (Segmented LRU)* [Arlitt et al. 1999b]. The SLRU strategy partitions the cache into two segments: an unprotected segment and a protected segment (reserved for popular objects). On the first request for an object, this object is inserted into the unprotected segment. On a cache hit, the object is moved to the protected segment. Both segments are managed with the LRU strategy, but only objects from the unprotected segment are removed. Objects from the protected segment are moved back in the unprotected segment as the most recently used object. This strategy requires a parameter which determines what percentage of the cache space is allocated to the protected segment.

—*Generational Replacement* [Osawa et al. 1997]. All objects are stored in $n (n > 1)$ lists. Each list $i < n$ contains objects that were requested i times. List n contains all objects with n or more requests. A request to an object causes its deletion in its current list and its insertion in the next list (at the beginning). Objects in list n are inserted at the beginning of list n. Replacement takes place at the end of list 1.

—*LRU** [Chang et al. 1999]. All requested objects are stored in one LRU list. Each object has a request counter. When a cached document is hit, it is moved to the start of the list and its hit count is incremented by one. At each replacement run, the hit count of the least recently used object is checked. If it is zero, the object is discarded. Otherwise the hit count is decreased by one, and the object is moved to the beginning of the list.

—*LRU-Hot* [Menaud et al. 2000]. LRU-Hot manages two LRU lists: one for hot (popular) objects and one for cold (not so popular) objects. An object is hot, if its request frequency at the original server is above a threshold. This information is sent with the object to the client (and proxy). According to this information, the object is inserted into the corresponding list. These lists are treated differently. LRU-Hot uses two reference counters: a base counter and a counter for hot objects. Both counters are initialized to zero. Each request increases the base counter by one. The hot counter is increased by one at every $\alpha (\alpha > 1)$ requests. When an object is requested (miss or hit), it is stored at the beginning of the corresponding list, and it is assigned an access value that is equal to the actual base counter value. Upon a replacement, the cache recomputes values for the two objects at the end of the two lists. Let *tail_hot* and *tail_cold* be the values of these objects. Then the reaccessibility values for these two objects are given as

$$\begin{aligned} &hot\_value = \\ &\quad \text{tail\_hot} - \text{hot reference counter}, \\ &cold\_value = \\ &\quad \text{tail\_cold} - \text{cold reference counter}. \end{aligned} \tag{5}$$

The object with the smaller value is removed from the cache. The algorithm iterates until there is enough space for the new incoming object.

—*HYPER-G* [Williams et al. 1996]. This strategy combines LRU, LFU, and SIZE. At first the least frequently used object is chosen. If there is more than one object that meets this criterion, the cache chooses the least recently used among them. If this still does not give a unique object to replace, the largest object is chosen.

—*CSS(Cubic Selection Scheme)* [Tatarinov 1998a]. CSS is an extension of the PSS strategy. CSS is based on a cube-like data structure. The CSS cube is formed by a matrix whose elements are lists of objects that correspond to one class. Objects in one class have size $\lfloor log\, s_i \rfloor$ and request frequency $\lfloor log\, f_i \rfloor$. Let *MaxF* be the maximum possible value for f_i. Then the height of the matrix is given by $\lfloor log\ MaxF \rfloor + 1$ and the width by $\lfloor log\, M \rfloor + 1$. The consideration of size and frequency requires more cache management. A modification of the frequency value can cause a reorganization of the lists, because the corresponding object has to be inserted in a new list. CSS also uses a more sophisticated replacement procedure, where the LRU objects of the diagonals of the CSS matrix are considered. Furthermore, CSS applies an aging mechanism to the frequency counters.

—*LRU-SP* [Cheng and Kambayashi 2000]. LRU-SP is another extension of PSS. Similarly to PSS, LRU-PS uses different classes. The class of object i is determined by $\lfloor log(s_i/f_i) \rfloor$. Each class maintains a separate LRU list. A request to a cached object i can cause its rearrangement into another list. On a replacement, the values of the least recently used objects in each class are compared. LRU-SP chooses object i if its value $(\Delta T_i s_i)/f_i$ is the largest one among all values of this objects. ΔT_i is the number of accesses since the last time object i was requested (like in PSS).

The advantage of these strategies is as follows:

—They combine recency and frequency. If designed properly, such strategies can avoid the problems of recency- and frequency-based strategies described above.

The disadvantage is as follows:

—Due to special procedures, most of these strategies introduce additional complexity. Only Generational Replacement and LRU* try to combine the simple implementation of LRU with frequency counts. However, they do not consider size.

3.4. Function-Based Strategies

These strategies use a potentially general function to calculate the value of an object. In the following we assume (if not stated otherwise) that the strategy chooses the object with the smallest value.

—*GD(Greedy Dual)-Size* [Cao and Irani 1997]. GD-Size maintains for each object a characteristic value H_i. A request for object i (new request or hit) requires a recalculation of H_i. H_i is calculated as

$$H_i = \frac{c_i}{s_i} + L. \tag{6}$$

L is a running aging factor, similar to LFU-DA, which is initialized to zero. GD-Size chooses the object with the smallest H_i-value. The value of this object is assigned to L.

GD-Size can be seen as a modification of the so-called Landlord algorithm proposed in Young [1998]. This simple deterministic on-line algorithm generalizes many well-known caching (paging) strategies. Young [1998] showed that the worst case performance for any request sequence is bounded.

—*GDSF* [Arlitt et al. 2000]. GDSF calculates H_i as

$$H_i = f_i \frac{c_i}{s_i} + L. \tag{7}$$

A generalized form of GDSF was described in Cherkasova and Ciardo [2001]. It calculates H_i as

$$H_i = \frac{f_i^\alpha}{s_i^\beta} + L. \tag{8}$$

c_i is set to 1. α and β are weighting parameters, which indicate the importance of the used factors.

—*GD** [Jin and Bestavros 2000]. GD* calculates H_i as

$$H_i = \left(\frac{(f_i * c_i)}{s_i}\right)^{\frac{1}{\beta}} + L. \tag{9}$$

β is a weighting factor, which is estimated in an online fashion. β characterizes reference correlation. Reference correlation is measured by the distribution of reference interarrivals for equally popular objects.

—*Server-assisted cache replacement* [Cohen et al. 1998]. This strategy enhances replacement policies by providing the proxy with the distribution function of the next request time for each object. The distribution is estimated by collecting per-object statistics on interrequest times. In server-assisted replacement, the server generates a histogram of interrequest times by observing its request logs. These statistics are incorporated into the calculation of the profit of an object. For further details concerning these and additional calculations, see Cohen et al. [1998].

—*TSP(Taylor Series Prediction)* [Yang et al. 2001]. TSP calculates H_i as

$$H_i = \frac{f_i * c_i}{s_i * T_T}. \tag{10}$$

T_T describes the temporal "acceleration" of requests to object i. $T_T = t_p - t_c$, where t_p is the predicted time for the next request and t_c is the current time. t_p is determined with a second-order Taylor series that uses the last and the next to last request times.

—*Bolot/Hoschka's strategy* [Bolot and Hoschka 1996]. This strategy uses the following function for calculating the value of object i:

$$f(i) = W_1 l_i + W_2 s_i + \frac{W_3 + W_4 s_i}{T_i}, \tag{11}$$

where W_1, W_2, W_3, and W_4 are tuning parameters. The values of the tuning parameters depend on the performance metric that should be maximized.

—*MIX* [Niclausse et al. 1998]. The MIX strategy uses the following function for calculating the value of object i:

$$f(i) = \frac{l_i^{r_1} f_i^{r_2}}{T_i^{r_3} s_i^{r_4}}. \tag{12}$$

$r_i (i = 1, \ldots, 4)$ are tuning parameters, that should be determined through experimental cache runs. There are no defined ranges for these tuning parameters. The authors of the algorithm used $r_i = 1$ for $i = 2, 3, 4$ and $r_1 = 0.1$.

—*M-Metric* [Wessels 1995]. The value of an object i is given by:

$$M - Metric = f_i^f T_i^r s_i^s. \qquad (13)$$

f, s, and r are weighting parameters. f should have a positive value to weight a higher number of requests. r should be negative to give more weight to recent requests. s could be positive or negative. A positive value gives more weight to larger object, a negative value gives more weight to smaller objects.

—*HYBRID* [Wooster and Abrams 1997]. The function for an object i from server s depends on the following parameters: c_s (time to contact server s), b_s (available bandwidth to server s). The function is defined as

$$f(i) = \frac{\left(c_s + \frac{W_b}{b_s}\right) f_i^{W_n}}{s_i}, \qquad (14)$$

where W_b and W_n are weighting parameters. Estimates for c_s and b_s are based on the time to fetch documents from server s in the recent past.

—*LNC-R-W3* [Scheuermann et al. 1997]. The so called profit $p(i)$ of an object i is calculated as

$$p(i) = \frac{f_i l_i}{s_i}. \qquad (15)$$

For this calculation, f_i is calculated in consideration of the last K ($K > 1$) request times (sliding window of K requests):

$$f_i = \frac{K}{(t - t_k) s_i^b}, \qquad (16)$$

where t corresponds to the actual time and t_k is the time of the oldest request time in the sliding window. b is used to weight the importance of different object sizes.

—*LRV* [Rizzo and Vicisano 2000]. This strategy chooses the object with lowest relative value which is a function of the probability that the document is accessed again. This probability P_r is calculated as

$$P_r(f_i, T_i, s_i) = \begin{cases} P(1, s_i)(1 - \tilde{D}(T_i)), & \text{if } f_i = 1, \\ P(f_i)(1 - \tilde{D}(T_i)) & \text{otherwise.} \end{cases} \qquad (18)$$

$P(f_i)$ is the probability that object i is requested $f_i + 1$ times given that it is requested f_i times. For objects with only one request, this probability depends on the size of the object and is given by $(P(1, s_i))$. $\tilde{D}(T_i)$ denotes the distribution of the interaccess times. Rizzo and Vicisano [2000] showed effective ways to compute adaptively $P(f_i)$, $P(1, s_i)$ and the parameters that influence $\tilde{D}(T_i)$ (based on the history of previous accesses).

—*LUV* [Bahn et al. 2002]. This strategy calculates for every object i a value $V(i)$ as

$$V(i) = W(i)p(i). \qquad (19)$$

$W(i)$ is the relative cost to fetch the object from its original server and is defined as $W(i) = c_i/s_i$. $p(i)$ is the "probability" that object i is referenced in the future. $p(i)$ is calculated as

$$p(i) = \sum_{k=1}^{f_i} F(t_c - t_k), \qquad (20)$$

where t_c is the actual time and t_k is the oldest request time in a sliding window of k request times. $F(x)$ should be decreasing, to give more weight to more recent references. One possibility is

$$F(x) = \left(\frac{1}{2}\right)^{\lambda x} \qquad (0 \le \lambda \le 1). \quad (21)$$

Bahn et al. [2002] showed how to calculate $V(i)$ in an efficient way even if all request times are used for $p(i)$.

—*LR(Logistic Regression)-Model* [Foong et al. 2000]. The goal of this model is to express the outcome of a dependent variable Y, in terms of its predictors, $(1, X_1, \ldots, X_k)$ and their respective coefficients $(\beta_0, \beta_1, \ldots, \beta_k)$. The LR probability is given by

$$\begin{aligned} P_{LR} &= P(Y = 1 \quad or \quad 0|1, X_1, \ldots, X_k) \\ &= \frac{1}{1 - e^{-z}}, \end{aligned}$$

where (22)

$$z = \sum_{j=0}^{k} \beta_j X_j \qquad -\infty < z < +\infty.$$

Given a set of observed data, the coefficients are obtained using the method of maximum likelihood estimation (learning phase). Once these coefficients are known, the LR probability can be calculated for other objects (prediction phase). In the caching context, the event of interest is whether a document is reaccessed at least once in the next N accesses. The predictors are factors like recency, frequency, size, and cost.

This class contains most of the proposed replacement strategies. Many of these strategies use similar factors and weighting schemes. The advantages of these proposals are as follows:

—They do not assume a fixed combination of factors or fixed usage of data structures. There exists no built-in bias for some objects. Through the proper choice of weighting parameters, one can try to optimize any performance metric.

—They consider a number of factors for handling different workload situations.

The disadvantages are as follows:

—Choosing appropriate weights is a difficult task. Some proposals assume that the weights are derived from Web trace studies. This is a simple but errorprone approach. Web workloads change over time and require some adaptive setting of the parameters. Adaptivity adds new complexity to the replacement process. Furthermore, there exist no exhaustive treatment of this sort of adaptivity in the literature.

—Using latency in the value calculation can introduce some problems. Recency and frequency information can be attained easily from the past request stream at the proxy. Latency is sampled at the proxy but influenced by many components on the path between server and proxy (or client). Statistical fluctuations complicate any accurate decision making. Furthermore, new technologies that support the movement of Web objects (content distribution networks) introduce further sources for inaccurate latency estimations. Therefore, using latency can lead to inferior replacement decisions.

3.5. Randomized Strategies

These strategies use randomized decisions to find an object for replacement. The following strategies were proposed:

—*RAND*. This strategy removes a random object.

—*HARMONIC* [Hosseini-Khayat 1997]. Whereas RAND uses equal probability for each object, HARMONIC removes from cache one item at random with a probability inversely proportional to its specific cost $cost = c_i/s_i$.

—*LRU-C and LRU-S* [Starobinski and Tse 2001]. LRU-C is a randomized version of LRU. Let $c_{\max} = \max\{c_1, c_2, \ldots, c_N\}$ be the maximum of the access costs of all N objects of a request sequence. Let $\tilde{c_i} = c_i/c_{\max}$ be the normalized cost for object i. When an object i is requested, it is moved to the head of the cache with probability $\tilde{c_i}$; otherwise, nothing is done.

LRU-S uses the size instead of the cost. Let $s_{\min} = \min\{s_1, s_2, \ldots, s_N\}$ be the size of the smallest objects among the N documents, and $d_i = s_{\min}/s_i$ be the normalized density of object i. LRU-S acts as LRU with probability d_i; otherwise the cache state is left unmodified.

Furthermore, Starobinski and Tse [2001] proposed an algorithm which deals with both varying-size and varying-cost objects. The following quantities were defined:

$$\beta_i = \frac{c_i}{s_i}; \quad \beta_{\max} = \max_i \{\beta_i\};$$
$$\tilde{\beta_i} = \frac{\beta_i}{\beta_{\max}}. \tag{23}$$

Upon a request for object i, this algorithm performs the same operation as LRU with probability $\tilde{\beta_i}$ and with $1 - \tilde{\beta_i}$ will leave the cache state unmodified.

—*Randomized replacement with general value functions* [Psounis and Prabhakar 2001]. This strategy draws N objects randomly from the cache and evicts the least useful object in the sample. The usefulness of a document can be determined by any utility function. After replacing the least useful object, the next $M(M < N)$ least useful objects are retained in memory. At the next replacement, $N - M$ new samples are drawn from the cache and the least useful of these $N - M$ and M previously retained is evicted. The M least useful of the remaining are stored in memory and so on.

Randomization presents a different approach to cache replacement. Randomized strategies try to reduce the complexity of the replacement process without sacrificing the quality too much. The advantages are as follows:

—Randomized strategies do not need special data structures for inserting and deleting objects. Searching can be supported by special data structures.
—Randomized strategies are simple to implement.

The disadvantage is as follows:

—Randomized strategies are more cumbersome to evaluate. For example, different simulation runs on the same Web request trace will give slightly different results.

4. ON THE IMPORTANCE OF CACHE REPLACEMENT STRATEGIES

Cache replacement was an important issue in the early days of caching. Nowadays cache replacement is often considered less important [Krishnamurthy and Rexford 2001; Rabinovich and Spatscheck 2002]. The following arguments are presented:

—Steadily falling costs of storage lead to caches of sizes large enough to hold most of the requested objects.
—The fraction of traffic that is cacheable steadily decreases.
—There exist "good-enough" algorithms that satisfy most situations in which cache replacement is used. Algorithms such as GD-Size are in the good-enough category.
—Changes to objects over time reduce the value of having a large cache that can store them longer.

We will comment on these arguments in the following sections.

4.1. Large Caches

This is the main argument against an exhaustive discussion of replacement strategies. It is based on numbers often found in the literature [Gray and Shenoy 2000]. The most striking argument is, that disk space doubles every 18 months (100 $\times$ in 10 years). Therefore the capacity of caches grows steadily. So replacement strategies are not seen as a limiting factor of the progress of proxy caching. This argument is underpinned with an example in Rabinovich and Spatscheck [2002]. We give a similar example in the following. Assume a cache that handles 1000 requests per second (average rate). At an average size of 10 kbytes, this request rate produces approximately 82 Mbps of data sent to the clients. Assuming 40% of uncacheable data that is not stored and a 40% overall byte hit rate, we get 16.4 Mbps (2.05 MBps) of incoming data. The aforementioned cache has a total disk capacity of 200 Gbytes. At 2.05 MBps, it would require about 28 h to fill the cache. Once it is filled, the cache will hold about 21 millions 10-kbyte-sized objects. Although the opinions vary on how long it should take a cache to collect a working set for its clients, a study of a large proxy cache [Arlitt et al. 1999a] has found the median stack distance to be 60,000, with a mean of 640,000 and a standard deviation of 1.5 million. Even the maximum stack distance was just over 16 million. These numbers indicate that even a basic LRU replacement strategy would be sufficient for this cache. Furthermore, cache vendors never differentiate themselves based on replacement strategies because they assume that disk space is currently not the limiting factor in proxy performance.

Although these arguments seem to be convincing, one has to bear in mind the following aspects:

—The example assumes a proxy cache with large disk capacity. Using replacement strategies in more limited systems can still be an important issue. Furthermore, main memory caches can benefit from advanced replacement strategies summarized in this survey.

—Due to new content (especially multimedia content), the average size of Web objects will grow with orders of magnitude in the future.

4.2. Reduction of Cacheable Traffic and Rate of Change

A response to a Web request is called *cacheable* if it is allowed to be stored in a cache and used for future requests; otherwise, it is uncacheable. Similarly a request is cacheable if it is satisfied by a cache, and uncacheable otherwise. A request to an uncacheable object is always uncacheable. Normally proxy caches do not cache dynamically generated objects, requests with a cookie header and responses with a set-cookie header (cookies are often used to personalize Web objects to a particular client or even particular request, e.g., no reuse possible), requests using methods other than GET or HEAD, and responses with certain codes, although they could be cached under certain circumstances (defined in HTTP 1.1.). Overall, around 40% of all requests are uncacheable [Rabinovich and Spatscheck 2002]. This clearly presents a big obstacle to caching and not only to cache replacement.

A cache has to deliver fresh information (Web objects) on each hit. If an object changes at the origin server, the cached data is stale. Therefore, the cache has to implement a consistency policy. Normally a certain amount of objects changes over time, that is, these objects are fresh only for a specific time period. A larger cache that can store these objects for a longer time period is not very useful, that is, the cache will have a moderate number of fresh objects during each time interval only.

4.3. Good-Enough Algorithms

In the early days of caching, simple replacement strategies were used. Therefore, research for more sophisticated replacement strategies was an important issue. Nowadays there exist a multitude of strategies. Although most of the papers in the literature provide evidence that their actually proposed strategies are the best of all, one can find some strategies that give good results in different evaluations. Such algorithms are considered as "good-enough" for general Web cache replacement but it is questionable if they are good enough for future requirements (e.g., multimedia, QoS). We comment on this topic in Section 5.

Examples of algorithms that are considered as good enough are: GD-Size and extensions thereof, LUV, PSS and extensions thereof. A definite performance ranking of the different replacement strategies is not possible as there exists no best strategy for different workload situations. Furthermore, replacement strategies give different results for different metrics. Often used metrics are hit-rate, byte-hit-rate, and delay-savings-ratio. Let h_i be the number of hits for video i, s_i the size of video i, d_i the latency seen when downloading object i from its original server, and r_i the number of requests to video i. The hit-rate of a cache for a request sequence with n different videos is given by $\sum_{i=1}^{n} h_i / \sum_{i=1}^{n} r_i$, the byte-hit-rate by $\sum_{i=1}^{n} s_i h_i / \sum_{i=1}^{n} s_i r_i$, and the delay-savings-ratio by $\sum_{i=1}^{n} d_i h_i / \sum_{i=1}^{n} d_i r_i$. The following statements concerning these performance metrics can be made:

—*Hit-rate*. By far the most often used cache performance metric. Strategies that favor smaller objects (like SIZE, LRU-Min, or function-based strategies with a strong weighting of object size) optimize for hit-rate. This performance metric is interesting for users that want to have a high percentage of local hits.

—*Byte-hit-rate*. Strategies that tend to remove bigger objects improve the hit-rate and decrease the byte-hit-rate. The different weighting of object sizes influences this tradeoff. This performance metric is interesting for ISPs when they try to minimize the download volume from the Internet.

—*Delay-savings-ratio*. Intuitively this metric seems to be appropriate for interactive systems as it tries to minimize the expected download time. It was introduced to show the improved performance of algorithms using download delays, in their calculations (like Hybrid). Due to the statistical fluctuations of download delays, performance results based on this metric can be unstable. Therefore, this metric is used seldom.

Good-enough algorithms give good performance for more than one performance metric. They represent a solution to the tradeoff introduced by diverse performance metrics. Furthermore some function-based strategies with weighting parameters can be optimized for any performance metric.

4.4. Strategy Usage

The previous section described some general aspects of so-called good-enough algorithms. Although it is difficult to give a definite list of good-enough algorithms, in the following section we give some guidelines for the proper selection.

4.4.1. Product-Based Approaches. Most of the caching products use a form of LRU replacement. There are two main reasons for that:

—It is easy to implement. An LRU implementation is less complex than the implementation of more elaborated strategies. Furthermore, LRU imposes less overhead than strategies that sort objects to a certain criterion. Inserting and removing have a complexity of $O(1)$ (for original LRU or strategies that preserve the LRU procedure of inserting and removing, e.g., LRU-S, LRU-C).

—Most cache vendors assume that disk space is not the limiting factor of the performance of their caching products. We discussed this topic in Section 4.1.

One popular caching product realized in software is the freely available Squid proxy cache. In its original implementation Squid uses LRU with some modifications. The replacement algorithm is not triggered on demand but runs periodically every second. Squid has a low and a high water mark. When the disk usage is close to the low water mark, the replacement is less aggressive (fewer objects removed). When the usage is close to the high water mark, the replacement is more aggressive (more objects removed). The replacement depends among other things on an LRU-threshold that is dynamically calculated, based on the current cache size and the low and high water marks. An object is removed if the time since last access is greater than this threshold. Furthermore, Squid supports a version of LFU-DA and GDSF. For specific implementation details, see Dilley et al. [1999].

4.4.2. Considerations Based on Statistical Properties. As stated earlier, it is not possible to identify the best replacement strategy. First, different replacement strategies optimize for different performance metrics. Second, different workload situations can result in different performance rankings for a specific set of replacement strategies. It is this workload sensitivity that produces the different results described in the literature.

There are typical workload factors that influence the performance of replacement strategies. The following factors are important in a object request stream (succession of object requests) to a proxy cache:

—*Object size*. This is the mainspring for a big part of replacement research. The size of Web objects can vary significantly. The size of Web objects is often modeled with a heavy-tailed distribution, a combination of light-tailed (body of the distribution) and heavy-tailed (tail of the distribution)(e.g., Barford et al. [1998]), or with a log-normal distribution (e.g., Downey [2001]).

—*Temporal locality*. Temporal locality describes the characteristic that an object just referenced has an increased probability of being referenced in the near future. Temporal locality has two aspects: popularity and correlation. Popularity describes the long-term probability of an object seen in a request stream. The skewed distribution of the popularity of a set of objects is often characterized by a Zipf distribution [Breslau et al. 1999] or recently by the empirical entropy of the request stream [Fonseca et al. 2003]. Temporal correlation focuses on the way in which references to a given object are separated by references to other objects. It is often characterized by the LRU-Stack distance model [Almeida et al. 1996; Mahanti et al. 2000] or recently by the coefficient of variation of the interreference gap (number of references to other objects between two consecutive requests for a certain object)[Fonseca et al. 2003].

These factors have a decisive influence on the performance of replacement strategies. Therefore, any evaluation should consider the interplay between workload and replacement strategy.

Considering these factors and results described in the literature, the following general rules for choosing a replacement strategy can be derived:

—Replacement strategies should contain some size differentiation. This size differentiation can be incorporated into a value function (like in most function-based strategies). This complicates the management of a sorted list of cached objects. Therefore, if the proxy has fewer CPU resources, such a differentiation could be too heavy-weight. A simple alternative is to separate the cache into a small number of caches. Each cache handles objects of a specific size range. Objects are inserted in their corresponding cache and managed with a simple replacement strategy inside this cache (examples for this procedure are PSS and variations thereof or partitioned caching).

—As described above, the complexity of the (re)calculation of object values (not only size-based) can be a decisive factor against a replacement strategy. Proxies that are CPU-bounded should not integrate such complex strategies (e.g., function-based replacement). Less complex alternatives are LRU and certain variations thereof.

—If the cache space is very small compared to the overall size of all Web objects seen in a workload, the cache replacement strategy should favor recency over frequency. This is due to the fact that temporal correlations between requests to the same object exist in the short term.

—If the cache space is large enough (few Gigabytes for actual workloads), long-term frequency should be incorporated into the caching decision because document popularity is the main contributor to temporal locality. Frequency should always be combined with an aging mechanism to reduce the effect of cache pollution.

These rules can be seen as general guidelines. Indeed, the performance of a cache replacement strategy depends heavily on the seen workload. Therefore, it is a valuable question how caches can be made adaptive to different workloads. We will comment on this topic in the following section about future research topics.

4.5. Alternative Models for Cache Management

Normally cache management is associated with cache replacement. If the cache fills up, some objects are removed to make space for further objects. A different approach to cache management constitutes static caching. With static caching the content of the cache is updated periodically. The popularity of objects is determined by information derived from prior periods. Although this approach is easy to implement and puts load on the cache at predetermined time points, it is only suitable for specific systems with a well-defined (prior known) number of objects. Therefore such approaches are used in caches for Web servers [Liu et al. 1998; Tatarinov et al. 1997].

An even simpler approach is to cache objects for a certain time period. If all objects are stored for the same time period T, the objects can be stored in a FIFO queue. Periodically the cache checks the end of the queue if some objects have exceeded their time limit and removes them from the cache. This approach is simple to implement and allows a simple analysis. The problem of this approach is that the used storage space increases strongly with increasing T. One remedy to this situation is to store only objects that were requested $n (n > 1)$ times. This filtering policy removes objects that do not contribute very much to the hit-rate but waste storage space. A similar technique for a slightly different approach was described in Rodriguez [2000].

4.6. Potentials for Capacity Improving

Proxy caching has a substantial impact on the average load of servers and networks. Therefore, this load reduction is often described as the main advantage of proxy caching. Although the reduction of the average load is useful to users due to the reduction in the average response time, it is less effective for servers and networks that are typically provisioned based on the high percentile of the load distribution. Interestingly, proxy caches have a diminishing impact on the tail of the load distribution. Raunak et al. [2002] described three key results for typical Web workloads:

—Proxy caches have a diminished impact on the tail of the load distribution. The reduction in the tail of the load distribution and the corresponding capacity savings depend on the percentile of the distribution chosen for provisioning: the higher the percentile, the smaller the savings.

—Although proxy caches smooth out some of the burstiness in the load, the resulting traffic continues to be bursty and heavy-tailed.

—Intense bursts in the load are caused by an increase in the request rate as well as the size of requested objects. Requests for large objects diminish the impact of caching because the locality exhibited by these objects is often poor.

It should be noted that these results are workload-dependent but they are based on typical workloads seen in the Web. Although some parameters of these workloads change over years, the basic aspects (e.g., heavy-tailed object sizes) will not change.

5. FUTURE RESEARCH TOPICS

In the previous section we have discussed the importance of cache replacement strategies. Although Web cache replacement in its general form seems to be a solved topic, there are new areas that need further investigation. The following areas can be identified:

—*Adaptive replacement*. Evaluations show that there exists no "best" replacement strategy. Depending on the workload, different strategies can give the best result. As workloads on the Web can change, it is a valuable question how a proxy can use different replacement strategies in an adaptive manner. Furthermore, function-based strategies can be made adaptive by changing the weighting parameters.

—*Coordinated replacement*. Replacement decisions affect the state of one proxy. In a caching hierarchy or distributed caching scheme, a coordination of replacement decisions at different proxies could give a superior performance. Furthermore, coordinated placement of Web objects could improve the performance further.

—*Combination of cache replacement and cache coherence*. Traditionally cache replacement and cache coherence are treated as separate topics. Nevertheless, there is a relationship between these topics because a cache should not return stale Web objects. The interplay between replacement and coherence has not been studied in detail.

—*Multimedia cache replacement*. Multimedia caches pose a new problem for replacement strategies. First, multimedia objects (especially videos) are very large and place a heavy load on caches. Therefore, partial caching techniques (the cache contains only important parts) were introduced. Second, multimedia caches can use adaptation techniques to augment replacement strategies.

—*Differentiated cache replacement*. Original caching does not support quality of service (QoS), that is, caching can be seen as a best-effort service. All objects are handled equally. Introducing some sort of differentiation can make the replacement process QoS aware.

The following sections provide a more thorough discussion of these topics.

5.1. Adaptive Replacement

The efficiency of replacement strategies depends on the actual workload. Differences in workloads can lead to different results for replacement strategies. Therefore, a cache should be adaptive with respect to different workloads. We distinguish between the following forms of adaptivity:

—*Adaptive replacement strategies*. Many function-based replacement strategies use weighting parameters to guide the replacement process. These parameters can be changed according to workload changes.

—*Adaptive application of different replacement strategies*. Original replacement strategies that are good enough for different workload scenarios can be used as building blocks for a combined replacement strategy.

Examples for the first category are proposed in Bolot and Hoschka [1996], Kelly et al. [1999], Niclausse et al. [1998], Wessels [1995], and Wooster and Abrams [1997]. A change of the proposed parameters will result in a different behavior. To what extent this can be done dynamically is an open question. Parameter adaptation was not discussed in any of the above-mentioned papers. Therefore, the online adaptation constitutes an interesting research topic.

Two proposals in the category of function-based replacement strategies use some sort of adaptation. The so-called LR-model in Foong et al. [2000] is adaptive in principle. The used coefficients are learned from past workloads periodically. Therefore, this strategy adapts to new workload situations. The problem is the additional overhead introduced by this machine learning technique. Another simple adaptive calculation is proposed for the LUV strategy proposed in Bahn et al. [2002]. The weighting parameter λ can adopt a value between 0 and 1. A simple strategy for adapting this parameter according to the actual workload was described in Lee et al. [1999].[4]

Aspects of the second category were discussed in Aguilar and Leiss [2001] and Aubert and Beugnard [1999]. Aubert and Beugnard [1999] discussed general conditions for adaptation in Web caches. They did not propose certain procedures but discussed some problems that can arise from this form of adaptivity. Aguilar and Leiss [2001] proposed a simple semi-adaptive procedure. By means of some rules, the cache determines according to the given workload the used replacement strategy.

Although adaptation constitutes an interesting aspect for cache replacement, one should not ignore the possible problems. Introducing adaptation in function-based replacement strategies adds additional complexity to the replacement process. Furthermore, the adaptation procedure should be intelligent enough to avoid unnecessary adaptations that could deteriorate the performance of the replacement strategy. A similar problem exists when the cache uses different replacement strategies adaptively. The changeover between different strategies should be done prudently to avoid wrong decisions. Furthermore, this kind of adaptivity adds another form of complexity

[4]This adaptation procedure was described for a strategy that forms the basis for LUV.

to the replacement process. The cache has to manage data (and data structures) for all the different replacement strategies. It is questionable if this is reasonable in a Web cache. Future research should therefore concentrate on the following two topics: design and evaluation of intelligent adaptation procedures (or rules) and verification of the applicability of these procedures to Web caches in real environments (e.g., high loads, bursty request sequences).

5.2. Coordinated Replacement

Most of the cache replacement research has concentrated on cache replacement strategies for one proxy. The decisions are derived for one proxy. In cooperating caching architectures (caching hierarchies, distributed caches), such local decisions could be inferior to coordinated replacement decisions. One example was described in Korupolu and Dahlin [1999]. They modified GD-Size for caching hierarchies. If an object is evicted from the cache, it is passed up to a parent cluster for possible inclusion in one of its caches. A specific admission control test checks if this object should be included into the parent cluster replacing a less valuable object.

Besides this simple replacement enhancement, Korupolu and Dahlin [1999] proposed optimal and heuristic coordinated placement algorithms. Such algorithms attempt to solve the following problem: Given a set of cooperating caches, the network distance between the caches and predictions of the access rates from each cache to each object determine where to place each object in order to minimize the average access cost. Another approach to coordinated placement was described in Ramaswamy and Liu [2002] which takes into account the contentions at individual caches in order to limit the replication of documents within a cache group and increase document hit rate. The concept of cache expiration age (average cache expiration age) is used to measure the contention of individual caches.

In contrast to cache replacement, coordinated cache replacement and placement has not attracted very much attention. Therefore, there is still some place for algorithmic research. On the one hand, this can mean enhancements of the approaches described above. On the other hand, this can mean new proposals. Besides this, an exhaustive performance evaluation is needed.

5.3. Cache Replacement and Cache Coherence

Caching creates numerous copies of Web objects distributed throughout the Internet. If an object is updated at the origin server, copies of that object become stale. Therefore, a proxy should apply special mechanisms to enforce the freshness of cached objects. While this problem of cache coherence is a widely studied topic in different areas of (distributed) computing science, it is especially difficult in the Web. This is due to its scale and loose coupling between servers and proxies and clients.

There are two approaches to cache consistency [Rabinovich and Spatscheck 2002]: validation and invalidation. With validation, the clients verify the validity of their cached objects with the origin server. This is typically done periodically and therefore this approach ensures weak consistency only. With invalidation, the origin server notifies clients if a cached object has changed. This approach has a potential for providing strong consistency.

Less work exists investigating the interplay between cache coherence techniques and cache replacement. Shim et al. [1998] described an enhancement of the LNC-R-W3 replacement strategy that considers the TTL (Time-to-live)-values of cached objects. They showed an increased performance over their previous algorithm and a simple enhanced version of LRU but did not compare it with better replacement algorithms. Krishnamurthy and Wills [1999] compared various combinations of coherence and replacement strategies. They found among other things that the inclusion of cache coherency issues in cache replacement yields little improvement in over-all performance. Belloum and Hertzberger [2002] described the impact of cache coherence techniques on different cache replacement strategies with respect to hit-rate. They showed among other things that a simple coherence technique can improve the performance in terms of hit-rate and that certain coherence techniques can result in an decreasing hit-rate for increasing cache sizes.

Over-all, these are initial results that need more investigation. Considering replacement and coherence together introduces more complexity into the evaluation (e.g., more factors to consider during the evaluation process).

5.4. Multimedia Cache Replacement

Due to the increasing amount of multimedia data, multimedia caching has increased in importance. In contrast to HTML pages, the size of multimedia objects can be reduced to a certain point without sacrificing too much quality. Quality reduction was first used for images in original Web caches. In the so-called Soft Caching approach [Kangasharju et al. 1998], unpopular images are not removed but recoded to a lower resolution. The idea is to provide a client with a lower image resolution until she asks explicitly for the original version. Menaud et al. [2000] described an LRU enhancement (LRU-QoS) that uses a separate LRU list for degradable objects and recompressable videos. Furthermore, some authors tried to combine transcoding of (multimedia) objects with cache replacement. Acharya et al. [1999] proposed to create different versions of an object, store them in the cache, and replace them independently. Under specific workload situations, this can give better results than simple cache replacement. Another proposal for a combination of cache replacement and transcoding can be found in Yeung et al. [2001]. Furthermore, there exist some proposals for combining transcoding and caching of streaming videos in heterogeneous client environments [Shen and Lee 2002; Tang et al. 2002].

Especially video data will increase in the near future and pose a heavy load on the network and caches. To overcome this problem, different proposals for video caching exist in the literature. Most of the proposals assume that video files are huge and that normal sized caches cannot store the many videos necessary to achieve a certain hit rate. Therefore, partial caching strategies have been proposed. Examples are caching of a prefix [Sen et al. 1999], caching of prefix and selected frames [Miao and Ortega 1999; Ma and Du 2000], prefix-assisted periodic broadcast of popular videos [Guo et al. 2002], optimal proxy prefix allocation integrated with server-based reactive transmission (batching, patching, stream-merging) [Wang et al. 2002], caching of bursty parts of a video [Zhang et al. 2000], caching of hotspot segments [Fahmi et al. 2001], popularity-based prefix caching [Park and E. J. Lim 2001], segment-based prefix caching [Wu et al. 2001], variable-sized chunk-based video caching [Balafoutis et al. 2002], and distributed architectures for partial caching [Acharya and Smith 2000; Bommaiah et al. 2000] of a video. Some of these caching schemes do not use any dynamic replacement but use periodic cache decision. Some use simple replacement (LRU) combined with partial caching decisions.

Other authors proposed to cache the whole video but adapt the quality of the videos according to some criteria. Examples are periodic caching of layered coded videos [Kangasharju et al. 2001], a combination of replacement strategies and layered coded videos [Paknikar et al. 2000], quality-adjusted caching of GoPs (group of pictures) [Sasabe et al. 2001], adaptive caching of layered coded videos in combination with congestion control [Rejaie and Kangasharju 2001], simple replacement strategies (patterns) for videos consisting of different quality steps [Podlipnig and Böszörmenyi 2002], or a combination of transcoding and cache replacement as described in a prior section.

Future multimedia caching research will be dominated by video caching research because of the following reasons:

—Videos are by far the biggest objects in a multimedia workload.
—Images and audio data (another source for load) are not so big and especially audio does not allow major adaptation.

The main problem of video caching research is not the lack of sufficient different proposals but the lack of practical experiences in real environments. Most of the proposals are evaluated by simulation. Some proposals are implemented in proprietary systems. Therefore future research should concentrate on the application of different proposals in real environments. Especially two aspects should be worked out in more detail:

—Cooperation of caching proposals and modern communication standards like RTP and RTSP. One example can be found in Gruber et al. [2000].
—Applicability of (quality-based) video caching to modern multimedia standards like MPEG-4 or MPEG-7. One example is described in Schojer et al. [2002].

Furthermore, the implemented systems should be evaluated with different metrics. Besides conventional metrics, newly defined specialized metrics should be evaluated. From prior evaluations, it is not clear which metric is suitable for video caching.

5.5. Differentiated Cache Replacement

Traditional caching (cache replacement) does not not support QoS (Quality of Service). Caching can be seen as a best-effort service. The inevitability of cache misses implies that caches cannot be relied upon to deliver the most popular objects in a consistent and predictable way. To support QoS, cache replacement has to be enhanced or replaced by other cache management techniques.

One discussion of the combination of network storage and QoS was presented in Chuang and Sirbu [1999]. They proposed the stor-serv framework, which was inspired by the intserv and diffserv frameworks in the data transmission domain. The framework supports multiple service classes, each with varying degrees of QoS. The proposed classes can be summarized as follows:

—*Best-effort caching*. This is original caching that does not support custom placement or custom replacement.
—*Differential caching*. This supports no custom placement but adds custom replacement. It does not provide any service guarantee, but offers preferential treatment to certain objects while they are in cache.
—*Push caching*. Push caching is the opposite of differential caching. The content provider specifies the cache nodes to which objects are pushed (custom placement). Once they arrive at the cache, they are subject to the local replacement strategy.
—*Replication*. Replication allows the content provider to specify custom placement and replacement strategies for its objects.

Replication corresponds to a guaranteed service, for example, it provides performance guarantees such as object lifetime and latency bounds. Resource reservation and admission control are necessary for this kind of service. From the network domain it is known that reservation adds additional complexity that deteriorates scalability. Therefore, we will concentrate on the simpler approach of differential caching (which we will call *differentiated caching* in the following). On the one hand, it is easier to implement and adds (hopefully) less complexity to the replacement process. Similar arguments hold for the differentiated service approach in the network domain. On the other hand, it is the second real QoS approach described above. Push caching does not realize any QoS guarantee at the local caching node.

The idea of differentiated caching is not new. There are a few proposals that try to introduce preferential treatment of certain cached objects. Examples are LRU-Hot and swLFU described in Section 3. They assume that the server provides additional information identifying the privileged objects, for example, some "marking" of objects. A different approach is presented in Lu et al. [2001]. Digital feedback control theory is used to manage cache resources (cache space) for different classes (HTML pages, images) in a way that guarantees bounded convergence time to a specified performance (specified hit-rate for each class). At fixed time intervals the resource allocations are corrected based on the measured performance error.

These different proposals for differentiation can be divided into two classes: the first class (LRU-Hot, swLFU) tries to transfer the differentiation principle from the network domain to the caching domain. Servers mark specific objects, and caches use this information for differentiated replacement. The second class (differentiation via feedback control) does not depend on any information sent by the server. The differentiation approach is handled at the proxy.

Although transferring known techniques from one domain to another could be an interesting idea one has to bear in mind the following problems:

—Any form of differentiation needs some accompanying pricing policy. Otherwise content providers will try to get the best service for all objects. These principles are well studied in the network domain but not in the caching domain.

—In contrast to the network domain, content providers have to mark individual objects. A priori knowledge of the importance of an object is sometimes difficult to attain.

Therefore the second approach where the proxy determines the differentiation without any server information is easier to implement. Furthermore, it seems to be the more intuitive approach for this domain, as the proxy can rely on the actual local workload.

The proposed feedback approach constitutes an intelligent way for adaptation but adds some overhead to the implementation and performance of the replacement strategy. It is an interesting future research question how differentiation can be applied with simpler techniques. One example is given in Feldman and Chuang [2002], where the well-known LRU strategy is enhanced to consider different classes of objects.

6. CONCLUSIONS

This article has given an exhaustive survey of cache replacement strategies proposed for Web caches. We concentrated on proposals for proxy caches that manage the cache replacement process at one specific proxy. A simple classification scheme for these replacement strategies was given and used for the description and general critique of the described replacement strategies. Although cache replacement is considered as a solved problem, we showed that there are still numerous areas for interesting research.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments that helped to improve the quality of this paper.

REFERENCES

ABRAMS, M., STANDRIDGE, C. R., ABDULLA, G., WILLIAMS, S., AND FOX, E. 1995. Caching proxies: Limitations and potentials. In *Proceedings of the 4th International World Wide Web Conference*.

ACHARYA, S., KORTH, H. F., AND POOSALA, V. 1999. Systematic multiresolution and its application to the World Wide Web. In *Proceedings of the 15th International Conference on Data Engineering*. IEEE Computer Society, Piscataway, NJ, 40–49.

ACHARYA, S. AND SMITH, B. 2000. Middleman: A video caching proxy server. In *Proceedings of the 10th International Workshop on Network and Operating System Support for Digital Audio and Video*.

AGGARWAL, C. C., WOLF, J. L., AND YU, P. S. 1999. Caching on the World Wide Web. *IEEE Trans. Knowl. Data Eng. 11*, 1 (Jan.), 94–107.

AGUILAR, J. AND LEISS, E. 2001. A Web proxy cache coherency and replacement approach. In *Proceedings of the 1st Asia-Pacific Conference on Web Intelligence*.

ALMEIDA, V., BESTAVROS, A., CROVELLA, M., AND DE OLIVEIRA, A. 1996. Characterizing reference locality in the WWW. In *Proceedings of* *the IEEE Conference on Parallel and Distributed Information Systems*. IEEE Computer Society, Piscataway, NJ, 92–103.

ARLITT, M., FRIEDRICH, R., AND JIN, T. 1999a. Workload characterization of a Web proxy in a cable modem environment. *ACM SIGMETRICS Perform. Eval. Rev. 27*, 2, 25–36.

ARLITT, M. F., CHERKASOVA, L., DILLEY, J., FRIEDRICH, R. J., AND JIN, T. Y. 2000. Evaluating content management techniques for Web proxy caches. *ACM SIGMETRICS Perform. Eval. Rev. 27*, 4 (Mar.), 3–11.

ARLITT, M. F., FRIEDRICH, R. J., AND JIN, T. Y. 1999b. Performance evaluation of Web proxy cache replacement policies. Tech. rep. HPL-98-97(R.1), Hewlett-Packard Company, Palo Alto, CA.

AUBERT, O. AND BEUGNARD, A. 1999. Towards a fine-grained adaptivity in Web caches. In *Proceedings of the 4th International Web Caching Workshop*.

BAHN, H., KOH, K., MIN, S. L., AND NOH, S. H. 2002. Efficient replacement of nonuniform objects in Web caches. *IEEE Comput. 35*, 6 (June), 65–73.

BALAFOUTIS, E., PANAGAKIS, A., LAOUTARIS, N., AND STAVRAKAKIS, I. 2002. The impact of replacement granularity on video caching. In *IFIP Networking 2002*. Lecture Notes in Computer Science, vol. 2345. Springer-Verlag, Berlin, Germany, 214–225.

BARFORD, P., BESTAVROS, A., CROVELLA, M., AND BRADLEY, A. 1998. Changes in Web client access patterns: Characteristics and caching implications. *World Wide Web J.: Special Issue on World Wide Web Characterization and Performance Evaluation 2*, 15–18.

BARISH, G. AND OBRACZKA, K. 2000. World Wide Web caching: Trends and techniques. *IEEE Commun. Mag. 38*, 5 (May), 178–185.

BELLOUM, A. S. Z. AND HERTZBERGER, L. O. 2002. Concurrent evaluation of Web cache replacement and coherence strategies. *Simulation 78*, 1 (Jan.), 28–35.

BOLOT, J. AND HOSCHKA, P. 1996. Performance engineering of the World Wide Web: Application to dimensioning and cache design. In *Proceedings of the 5th International World Wide Web Conference*. Elsevier, Amsterdam, The Netherlands.

BOMMAIAH, E., GUO, K., HOFMANN, M., AND PAUL, S. 2000. Design and implementation of a caching system for streaming media over the internet. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*. IEEE Computer Society, Piscataway, NJ.

BRESLAU, L., CAO, P., PHILIPS, G., AND SHENKER, S. 1999. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the IEEE INFOCOM*. IEEE Computer Society, Piscataway, NJ, 126–134.

CAO, P. AND IRANI, S. 1997. Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*. 193–206.

CHANG, C.-Y., MCGREGOR, T., AND HOLMES, G. 1999. The LRU* WWW proxy cache document replacement algorithm. In *Proceedings of the Asia Pacific Web Conference*.

CHENG, K. AND KAMBAYASHI, Y. 2000. A size-adjusted and popularity-aware LRU replacement algorithm for Web caching. In *Proceedings of the 24th International Computer Software and Applications Conference (COMPSAC)*. IEEE Computer Society, Piscataway, NJ, 48–53.

CHERKASOVA, L. AND CIARDO, G. 2001. Role of aging, frequency and size in Web caching replacement strategies. In *Proceedings of the 2001 Conference on High-Performance Computing and Networking (HPCN'01)*. Lecture Notes in Computer Science, vol. 2110. Springer-Verlag, Berlin, Germany, 114–123.

CHUANG, J. AND SIRBU, M. 1999. Adding quality-of-service to network storage. In *Proceedings of the Workshop on Internet Service Quality Economics*.

COHEN, E., KRISHNAMURTHY, B., AND REXFORD, J. 1998. Evaluating server-assisted cache replacement in the Web. In *Proceedings of the 6th European Symposium on Algorithms*. Lecture Notes in Computer Science, vol. 1461, Springer-Verlag, Germany, 307–319.

DAVISON, B. D. 2001. A Web caching primer. *IEEE Internet Comput. 5*, 4 (July), 38–45.

DILLEY, J., ARLITT, M., AND PERRET, S. 1999. Enhancement and validation of the Squid cache replacement policy. In *Proceedings of the 4th International Web Caching Workshop*.

DOWNEY, A. B. 2001. The structural cause of file size distributions. In *Proceedings of the 9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE Computer Society, Piscataway, NJ.

FAHMI, H., LATIF, M., SEDIGH-ALI, S., GHAFOOR, A., LIU, P., AND HSU, L. H. 2001. Proxy servers for scalable interactive video support. *IEEE Comput. 43*, 9 (Sept.), 54–60.

FELDMAN, M. AND CHUANG, J. 2002. Service differentiation in Web caching and content distribution. In *Proceedings of the IASTED International Conference on Communications and Computer Networks*. IASTED/ACTA Press, Calgary, Alta., Canada.

FONSECA, R., ALMEIDA, V., CROVELLA, M., AND ABRAHAO, B. 2003. On the intrinsic locality properties of Web reference streams. In *Proceedings of the IEEE INFOCOM*. IEEE Computer Society, Piscataway, NJ.

FOONG, A. P., HU, Y., AND HEISEY, D. M. 2000. Essence of an effective Web caching algorithm. In *Proceedings of the International Conference on Internet Computing*. 269–276.

GRAY, J. AND SHENOY, P. 2000. Rules of thumb in data engineering. In *Proceedings of the 16th International Conference on Data Engineering*. IEEE Computer Society, Piscataway, NJ, 3–12.

GRUBER, S., REXFORD, J., AND BASSO, A. 2000. Protocol considerations for a prefix-caching proxy for multimedia streams. In *Proceedings of the 9th World Wide Web Conference*.

GUO, Y., SEN, S., AND TOWSLEY, D. 2002. Prefix caching assisted periodic broadcast for streaming popular videos. In *Proceedings of ICC (International Conference on Communications)*. IEEE Computer Society, Piscataway, NJ.

HOSSEINI-KHAYAT, S. 1997. Investigation of generalized caching. Ph.D. dissertation. Washington University, St. Louis, MO.

JIN, S. AND BESTAVROS, A. 2000. GreedyDual*: Web caching algorithms exploiting the two sources of temporal locality in Web request streams. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*.

KANGASHARJU, J., HARTANTO, F., REISSLEIN, M., AND ROSS, K. W. 2001. Distributing layered encoded video through caches. In *Proceedings of the IEEE INFOCOM*. IEEE Computer Society, Piscataway, NJ, 1791–1800.

KANGASHARJU, J., KWON, Y. G., AND ORTEGA, A. 1998. Design and implementation of a soft caching proxy. *Comput. Netw. ISDN Syst. 30*, 22-23 (Nov.), 2113–2121.

KELLY, T., JAMIN, S., AND MACKIE-MASON, J. K. 1999. Variable QoS from shared Web caches: User centered design and value-sensitive replacement. In *Proceedings of the MIT Workshop on Internet Service Quality Economics*.

KORUPOLU, M. P. AND DAHLIN, M. 1999. Coordinated placement and replacement for large-scale distributed caches. In *Proceedings of the IEEE Workshop on Internet Applications*. IEEE Computer Society, Piscataway, NJ, 62–71.

KRISHNAMURTHY, B. AND REXFORD, J. 2001. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*. Addison-Wesley, Reading, MA.

KRISHNAMURTHY, B. AND WILLS, C. E. 1999. Proxy cache coherency and replacement—towards a more complete picture. In *Proceedings of the 19th International Conference on Distributed Computing Systems*. IEEE Computer Society, Piscataway, NJ, 332–339.

LEE, D., CHOI, J., KIM, J.-H., NOH, S., MIN, S. L., CHO, Y., AND KIM, C. S. 1999. On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies. In *Proceedings of ACM SIGMETRICS*. ACM Press, New York, NY, 134–143.

LIU, Z., NAIN, P., NICLAUSSE, N., AND TOWSLEY, D. 1998. Static caching of Web servers. In *Multimedia Computing And Networking*, vol. 3310, SPIE Press, Bellingham, WA, 179–190.

LU, Y., SAXENA, A., AND ABDELZAHER, T. F. 2001. Differentiated caching services: A control-theoretical approach. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, Piscataway, NJ, 615–624.

MA, W.-H. AND DU, D. H.-C. 2000. Reducing bandwidth requirement for delivering video over wide area networks with proxy server. In *Proceedings of the IEEE International Conference on Multimedia and Expo*. IEEE Computer Society, Piscataway, NJ, 991–994.

MAHANTI, A., EAGER, D., AND WILLIAMSON, C. 2000. Temporal locality and its impact on Web proxy cache performance. *Perform. Eval. 42*, 2–3 (Sept.), 187–203.

MENAUD, J.-M., ISSARNY, V., AND BANATRE, M. 2000. Improving effectiveness of Web caching. In *Recent Advances in Distributed Systems*. Lecture Notes in Computer Science, vol. 1752. Springer-Verlag, Berlin, Germany, 375–401.

MIAO, Z. AND ORTEGA, A. 1999. Proxy Caching for Efficient Video Services over the Internet. In *Proceedings of the 9th International Packet Video Workshop* (PVW'99).

MURTA, C. D., ALMEIDA, V. A. F., AND MEIRA, W. 1998. Analyzing performance of partitioned caches for the WWW. In *Proceedings of the 3rd International WWW Caching Workshop*.

NICLAUSSE, N., LIU, Z., AND NAIN, P. 1998. A new efficient caching policy for the World Wide Web. In *Proceedings of the Workshop on Internet Server Performance*. 119–128.

OSAWA, N., YUBA, T., AND HAKOZAKI, K. 1997. Generational replacement schemes for a WWW proxy server. In *High-Performance Computing and Networking (HPCN'97)*. Lecture Notes in Computer Science, vol. 1225. Springer-Verlag, Berlin, Germany, 940–949.

PAKNIKAR, S., KANKANHALLI, M., RAMAKRISHNAN, K. R., SRINIVASAN, S. H., AND NGOH, L. H. 2000. A caching and streaming framework for multimedia. In *Proceedings of ACM Multimedia*. ACM Press, New York, NY, 13–20.

PARK, S. H., LIM, E. J., AND CHUNG, K. D. 2001. Popularity-based partial caching for VOD systems using a proxy server. In *Proceedings of the Workshop on Parallel and Distributed Computing in Image Processing, Video Processing and Multimedia*. IEEE Computer Society, Piscataway, NJ.

PITKOW, J. AND RECKER, M. 1994. A simple yet robust caching algorithm based on dynamic access patterns. In *Proceedings of the 2nd International World Wide Web Conference*. 1039–1046.

PODLIPNIG, S. AND BÖSZÖRMENYI, L. 2002. Replacement strategies for quality based video caching. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*. Vol. 2. IEEE Computer Society, Piscataway, NJ, 49–53.

PSOUNIS, K. AND PRABHAKAR, B. 2001. A randomized Web-cache replacement scheme. In *Proceedings of the IEEE INFOCOM*. IEEE Computer Society, Piscataway, NJ, 1407–1415.

RABINOVICH, M. AND SPATSCHECK, O. 2002. *Web Caching and Replication*. Addison-Wesley, Reading, MA.

RAMASWAMY, L. AND LIU, L. 2002. A new document placement scheme for cooperative caching on the internet. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, Piscataway, NJ, 95–103.

RAUNAK, M. S., SHENOY, P., GOYAL, P., RAMAMRITHAM, K., AND KULKARNI, P. 2002. Implications of proxy caching for provisioning servers and networks. *IEEE J. Sel. Areas Commun. (JSAC): Special Issue on Internet Proxy Services 20*, 7 (Sept.), 1276–1289.

REDDY, M. AND FLETCHER, G. P. 1998. Intelligent Web caching using document life histories: A comparison with existing cache management techniques. In *Proceedings of the 3rd International Web Caching Workshop*.

REJAIE, R. AND KANGASHARJU, J. 2001. Mocha: A quality adaptive multimedia proxy cache for Internet streaming. In *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM Press, New York, NY, 3–10.

RIZZO, L. AND VICISANO, L. 2000. Replacement policies for a proxy cache. *IEEE/ACM Trans. Netw. 8*, 2 (Apr.), 158–170.

RODRIGUEZ, P. 2000. Scalable content distribution in the Internet. Ph.D. dissertation, Ecole Polytechnique Fédérale de Lausanne (EPFL) Lausanne, Switzerland.

SASABE, M., WAKAMIYA, N., MURATA, M., AND MIYAHARA, H. 2001. Proxy caching mechanisms with video quality adjustment. In *Proceedings of the SPIE Conference on Internet Multimedia Management Systems*, vol. 4519, SPIE Press, Bellingham, WA, 276–284.

SCHEUERMANN, P., SHIM, J., AND VINGRALEK, R. 1997. A Case for delay-conscious caching of Web-documents. In *Proceedings of the 6th International WWW Conference*.

SCHOJER, P., BÖSZÖRMENYI, L., AND HELLWAGNER, H. 2002. An adaptive MPEG-4 proxy cache. In *Proceedings of DAPSYS 2002*. Kluwer, Dordrecht, The Netherlands.

SEN, S., REXFORD, J., AND TOWSLEY, D. 1999. Proxy prefix caching for multimedia streams. In *Proceedings of IEEE INFOCOM'99*. IEEE Computer Society, Piscataway, NJ, 1310–1319.

SHEN, B. AND LEE, S.-J. 2002. Transcoding-enabled caching proxy for video delivery in heterogeneous network environments. In *International Conference on Internet and Multimedia Systems and Applications (IMSA)*. IASTED/ACTA Press, Calgary, Alta, Canada, 360–365.

SHIM, J., SCHEUERMANN, P., AND VINGRALEK, R. 1998. A unified algorithm for cache replacement and consistency in Web proxy servers. In *International Workshop on the Web and Databases*. Lecture Notes in Computer Science, vol. 1590, Springer-Verlag, Berlin, Germany, 1–13.

STAROBINSKI, D. AND TSE, D. 2001. Probabilistic methods for Web caching. *Perform. Eval. 46*, 2–3 (Oct.), 125–137.

TANG, X., ZHANG, F., AND CHANSON, S. 2002. Streaming media caching algorithms for transcoding proxies. In *Proceedings of the International Conference on Parallel Processing*. IEEE Computer Society, Piscataway, NJ.

TATARINOV, I. 1998a. An efficient LFU-like policy for Web caches. Tech. Rep. NDSU-CSORTR-98-01, Computer Science Department, North Dakota State University, Wahpeton, ND.

TATARINOV, I. 1998b. Performance analysis of cache policies for Web servers. In *Proceedings of the 9th International Conference on Computing and Information*.

TATARINOV, I., ROUSSKOV, A., AND SOLOVIEV, V. 1997. Static caching in Web servers. In *Proceedings of the IEEE International Conference on Computer Communications and Networks*. IEEE Computer Society, Piscataway, NJ.

VAKALI, A. 2000. LRU-based algorithms for Web cache replacement. In *International Conference on Electronic Commerce and Web Technologies*. Lecture Notes in Computer Science, vol. 1875. Springer-Verlag, Berlin, Germany, 409–418.

WANG, B., SEN, S., ADLER, M., AND TOWSLEY, D. 2002. Optimal proxy cache allocation for efficient streaming media distribution. In *Proceedings of the IEEE INFOCOM*. IEEE Computer Society, Piscataway, NJ, 1726–1735.

WANG, J. 1999. A survey of Web caching schemes for the internet. *ACM Comput. Commun. Rev. 29*, 5 (Oct.), 36–46.

WESSELS, D. 1995. Intelligent caching for World-Wide-Web objects. M.S. thesis, University of Colorado at Boulder, Boulder, CO.

WESSELS, D. 2001. *Web Caching*. O'Reilly, Sebastopol, CA.

WILLIAMS, S., ABRAMS, M., STANDRIDGE, C. R., ABDULLA, G., AND FOX, E. A. 1996. Removal policies in network caches for World-Wide Web documents. In *Proceedings of ACM SIGCOMM*. ACM Press, New York, NY, 293–305.

WOOSTER, R. P. AND ABRAMS, M. 1997. Proxy caching that estimates page load delays. In *Proceedings of the 6th International World Wide Web Conference*.

WU, K.-L., YU, P. S., AND WOLF, J. L. 2001. Segment-based proxy caching of multimedia streams. In *Proceedings of the 10th International World Wide Web Conference*. ACM Press, New York, NY, 36–44.

YANG, Q., ZHANG, H. H., AND ZHANG, H. 2001. Taylor series prediction: A cache replacement policy based on second-order trend analysis. In *Proceedings of the 34th Hawaii International Conference on Systems Sciences*. IEEE Computer Society, Piscataway, NJ.

YEUNG, K. H., WONG, C. C., AND WONG, K. Y. 2001. A cache replacement policy for transcoding proxy servers. In *Proceedings of World Multiconference* *on Systemics, Cybernetics and Informatics*. Vol. 12, 234–237.

YOUNG, N. E. 1998. On-line file caching. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM Press, New York, NY, 82–86.

ZHANG, J., IZMAILOV, R., REININGER, D., AND OTT, M. 1999. Web caching framework: Analytical models and beyound. In *Proceedings of the IEEE Workshop on Internet Applications*. IEEE Computer Society, Piscataway, NJ.

ZHANG, Z.-L., WANG, Y., DU, D. H. C., AND SHU, D. 2000. Video staging: A proxy-server-based approach to end-to-end video delivery over wide-area networks. *IEEE/ACM Trans. Netw. 8*, 4 (Aug.), 429–442.

Received October 2002; revised August 2003; accepted August 2003