Performance Evaluation of Web Proxy Cache Replacement Policies

Martin Arlitt, Rich Friedrich, and Tai Jin
Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304
{arlitt, richf, tai}@hpl.hp.com

Abstract

The continued growth of the World-Wide Web and the emergence of new end-user technologies such as cable modems necessitate the use of proxy caches to reduce latency, network traffic and Web server loads. In this paper we analyze the importance of different Web proxy workload characteristics in making good cache replacement decisions. We evaluate workload characteristics such as object size, recency of reference, frequency of reference, and turnover in the active set of objects. Trace-driven simulation is used to evaluate the effectiveness of various replacement policies for Web proxy caches. The extended duration of the trace (117 million requests collected over five months) allows long term side effects of replacement policies to be identified and quantified.

Our results indicate that higher cache hit rates are achieved using size-based replacement policies. These policies store a large number of small objects in the cache, thus increasing the probability of an object being in the cache when requested. To achieve higher byte hit rates a few larger files must be retained in the cache. We found frequency-based policies to work best for this metric, as they keep the most popular files, regardless of size, in the cache. With either approach it is important that inactive objects be removed from the cache to prevent performance degradation due to pollution.

Keywords: World-Wide Web, performance evaluation, proxy caching, replacement policies, trace-driven simulation

1. Introduction

The World-Wide Web ("The Web") has grown tremendously in the past few years to become the most prevalent source of traffic on the Internet today. This growth has led to congested backbone links, overloaded Web servers and frustrated users. These problems will become more severe as new end-user technologies such as cable modems are deployed. One solution that could help relieve these problems is object caching [15][27].

In this paper we present a trace-driven simulation study of a Web proxy cache. Our goal in this study is to evaluate the effects of different workload characteristics on the replacement decisions made by the cache. The workload characteristics that we consider include object size, recency of reference, frequency of

reference and turnover in the active set of objects. These characteristics were identified in our Web proxy workload characterization study [2].

Our research on Web proxies has utilized measurements of an actual Web proxy workload. We collected data from a proxy cache that is located in an Internet Service Provider (ISP) environment. Subscribers to this ISP access the Web using high-speed cable modems. Measurements of this proxy were collected over a five month period (January 3rd - May 31st, 1997). In total more than 117 million requests were recorded.

Although other researchers have performed caching simulations of Web proxies these previous studies have been limited to either short-term traces of busy proxies [9][11][14] or long-term traces of relatively inactive proxies [19][25][26]. Our study is the first to examine a busy proxy over an extended period of time. Caching is more important in a busy environment as it reduces the demand on the shared external network link. The introduction of high-speed cable modems will significantly increase the demand on the shared external network link, making caches an even more valuable component of the Web architecture. Long-term traces are important in order to identify potential side effects of replacement policies. Furthermore, our study identifies which workload characteristics merit consideration in cache replacement decisions and discusses why these characteristics are important. We then use these characteristics to evaluate the achieved hit rates and byte hit rates of several existing replacement policies and to identify their strengths and weaknesses.

The remainder of this paper is organized as follows. Section 2 provides background information on the World-Wide Web and discusses related work. Section 3 describes the collection and reduction of the workload data set. Section 4 summarizes the results of our workload characterization study focusing on the characteristics that merit consideration when making cache replacement decisions. Section 5 provides the design of our trace-driven simulation study while Section 6 presents the simulation results. The paper concludes in Section 7 with a summary of our findings and a discussion of future work.

2. Background

The World-Wide Web is based on the client-server model [8]. Web browsers are used by people to access information that is available on the Web. Web servers provide the objects that are requested by the clients. Information on the format of Web requests and responses is available in the HTTP specification [13].

Figure 1 illustrates how a proxy can be used within the World-Wide Web. In Figure 1, clients 0-M are configured to use the proxy in order to resolve any requests for objects available on origin servers 0-N. All of the clients communicate with the proxy using the HTTP protocol. The proxy then communicates

with the appropriate origin server using the protocol specified in the URL of the requested object (e.g., http, ftp, gopher) [20].

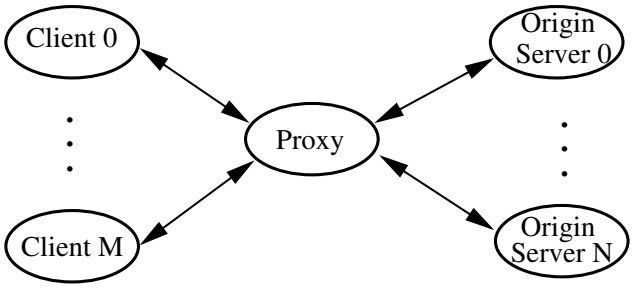


Figure 1. Using a Proxy within the World-Wide Web

When the proxy receives a request from a client the proxy attempts to fulfill the request from among the objects stored in the proxy's cache. If the requested object is found (a *cache hit*) the proxy can immediately respond to the client's request. If the requested object is not found (a *cache miss*) the proxy then attempts to retrieve the object from another location, such as a peer or parent proxy cache or the origin server. Once a copy of the object has been retrieved the proxy can complete its response to the client. If the requested object is *cacheable* (based on information provided by the origin server or determined from the URL) the proxy may decide to add a copy of the object to its cache. If the object is *uncacheable* (again determined from the URL or information from the origin server) the proxy should not store a copy in its cache.

Two common metrics for evaluating the performance of a Web proxy cache are *hit rate* and *byte hit rate*. The hit rate is the percentage of all requests that can be satisfied by searching the cache for a copy of the requested object. The byte hit rate represents the percentage of all data that is transferred directly from the cache rather than from the origin server. The results of our workload characterization study indicate that a trade off exists between these two metrics [2]. Our workload results show that most requests are for small objects, which suggests that the probability of achieving a high hit rate would be increased if the cache were used to store a large number of small objects. However, our workload results also revealed that a significant portion of the network traffic is caused by the transfer of very large objects. Thus to achieve higher byte hit rates a few larger objects must be cached at the expense of many smaller ones. Our characterization study also suggested that a wide-scale deployment of cable modems (or other high bandwidth access technologies) may increase the number of large object transfers. One of the goals of this study is to determine how existing replacement policies perform under these changing workloads.

A proxy cache that is primarily intended to reduce response times for users should utilize a replacement policy that achieves high hit rates. In an environ-

ment where saving bandwidth on the shared external network is of utmost importance, the proxy cache should use a replacement policy that achieves high byte hit rates. A proxy cache could also utilize multiple replacement policies. For example, a replacement policy that achieves high hit rates could be used to manage the proxy's memory cache in order to serve as many requests as quickly as possible and to avoid a disk I/O bottleneck. The proxy's much larger disk cache could be managed with a policy that achieves higher byte hit rates, in order to reduce external network traffic.

The tremendous growth of the World-Wide Web has motivated many research efforts aimed at improving Web performance and scalability. In this section we focus on Web proxy research.

Understanding Web proxy workloads is an important first step in proxy research. A quantitative way to understand these workloads is through workload characterization. A number of recent efforts [7][11][14], including our own [2], have identified numerous characteristics of proxy workloads. We use this knowledge of proxy workloads to help identify the strengths and weaknesses of different replacement policies.

Other researchers have focussed on managing proxy cache contents in order to improve hit rates (as well as other metrics). Currently there are two approaches to cache management. One approach attempts to use as few resources as possible by making good replacement decisions when the cache is full [9][19][25][26]. The alternative approach is to provide the cache with abundant resources so that few replacement decisions need to be made. In this paper we focus on the first approach. While some organizations may be willing to continuously add resources to their proxy cache, we feel that the majority of enterprises will be more interested in achieving the best possible performance for the lowest possible cost. Thus, throughout the remainder of this paper, we focus on maximizing either the hit rate or the byte hit rate of a proxy cache that has a limited amount of cache space.

Several research efforts have evaluated the performance of existing Web proxy server software. Almeida and Cao created a benchmark in order to directly compare the performance of different proxy servers [1]. Maltzahn *et al.* examined the performance of several proxy servers in a live environment [22], and proposed several methods for reducing disk I/O in proxy caches [21].

3. Data Collection and Reduction

In order to characterize the workload of a Web proxy and to conduct a trace-driven simulation of a Web proxy cache, measurements of an actual Web proxy workload were collected. Section 3.1 presents background information on the data collection site. Section 3.2 discusses the data that was collected. Section 3.3

describes how the collected data was reduced into a more manageable form and summarizes the assumptions we made to address the limitations of our data set.

3.1. Data Collection Site

The site under study provides interactive data services to residential and business subscribers using cable modems. The services available to the subscribers include email, network news and the World-Wide Web. Figure 2 shows a simplified view of the system under study. To access the available services a subscriber uses a cable modem to connect to the server complex through the Signal Conversion System (SCS). The SCS routes all requests for Web objects (i.e., HTTP, FTP, and Gopher requests) to the Web proxy. This proxy includes an object cache so some of the client requests can be satisfied within the server complex. On a cache miss the proxy retrieves the object from an origin server on the Internet. The access logs of this proxy were collected. Customer requests for other services such as Email and News are forwarded to a separate server; the workload of the Email and News server was not measured and is not used in this study.

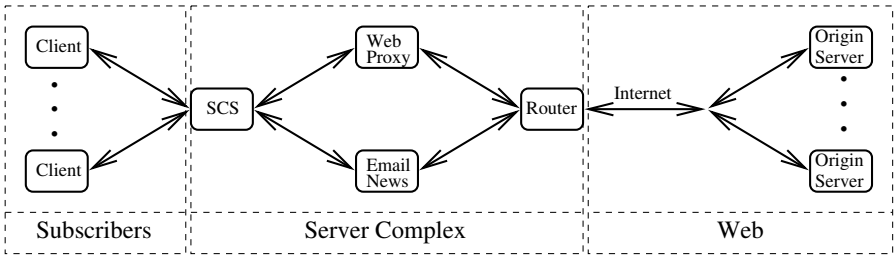


Figure 2. Diagram of the System Under Study

3.2. Data Collection

The access logs of the Web proxy described in Section 3.1 were collected for this study. These access logs contain information on all client requests for Web objects from January 3rd, 1997 until May 31st, 1997[1]. Each entry in an access log contains information on a single request received by the Web proxy from a client. Each entry includes the client IP address (dynamically assigned), the time of the request, the requested URL, the status codes for both the proxy and origin server

1. The access logs were collected on a daily basis. The access logs were not available on 13 days and were incomplete on 4 other days. Despite these gaps in the data set we have a relatively complete view of the proxy workload for an extended period of time.

responses, the size of the response (in bytes) and the time required to complete the response. A summary of the amount of raw data collected is given in Table 1.

Table 1. Summary of Access Log Characteristics (Raw Data Set)

Access Log Duration	January 3rd - May 31st, 1997
Total Requests	117,652,652
Total Content Data Transferred	1,340 GB

3.3. Data Reduction

Due to the extremely large access logs created by the proxy (nearly 30 GB of data) we found it necessary to create a smaller, more compact log due to storage constraints and to ensure that the workload analyses and caching simulations could be completed in a reasonable amount of time. We performed these reductions in several ways while still maintaining as much of the original information as possible. One very effective method of reduction is to represent the access log information in a more efficient manner (e.g., map the unique URLs to distinct integer identifiers). We also removed information that we felt would be of little or no value in either the workload analysis or the simulation study (e.g., we kept only GET requests which accounted for 98% of all requests and 99.2% of all content data). After reducing the access logs the overall statistics were recalculated. The results are shown in Table 2. The reduced data set is 4.5 GB (1.5GB compressed). This represents not only a tremendous space savings but also a time savings as the log is in a format that dramatically improves the efficiency of our analysis tools and cache simulator.

Table 2. Summary of Access Log Characteristics (Reduced Data Set)

Access Log Duration	January 3rd - May 31st, 1997
Total Requests	115,310,904
Total Content Bytes	1,328 GB
Unique Cacheable Requests	16,225,621
Total Uncacheable Requests	9,020,632
Unique Cacheable Content Bytes	389 GB
Total Uncacheable Content Bytes	56 GB

Unfortunately, not all information of interest is available in the access logs. One problem that we faced was trying to correctly identify object modifications and user aborted connections. To address this problem we assumed that modifica-

tions and aborts could be identified by a change in the size of the object. If the size changed by less than 5% we hypothesized that the object had been modified[2]; otherwise, we speculated that a user abort had occurred (either during the current request or on a previous one). If no change in size occurred, we assumed that the object had not been modified.

4. Workload Characterization

In this section we present a summary of our workload characterization study [2]. In particular we focus on the characteristics that we feel could impact proxy performance and cache replacement decisions.

Cacheable Objects. In order for Web caching to improve performance it is vital that most objects be cacheable. Our analysis of the data set under study revealed that 92% of all requested objects (96% of the data transferred) were cacheable.

Object Set Size. In Table 2 we reported that there were over 16 million unique cacheable objects requested during the measurement period. This is several orders of magnitude larger than the number of unique objects seen in Web server workloads [4]. Due to the extremely large object set size the proxy cache must be able to quickly determine whether a requested object is cached to reduce response latency. The proxy must also efficiently update its state on a cache hit, miss or replacement.

Object Sizes. One of the obstacles for Web caching is working effectively with variable-sized objects. While most of the requested objects are small (the median object size in this data set was 4 KB) there are some extremely large objects available. The largest object requested during the measurement period was a 148 MB video. We speculate that the higher access speeds available to the clients are increasing the number of large transfers as well as the maximum size of transfers. The issue for the proxy cache is to decide whether to cache a large number of small objects (which could potentially increase the hit rate) or to cache a few large objects (possibly increasing the byte hit rate).

Recency of Reference. Most Web proxy caches in use today utilize the Least Recently Used (LRU) replacement policy (or some derivative of LRU). This policy works best when the access stream exhibits strong temporal locality or recency of reference (i.e., objects which have recently been referenced are likely to be re-referenced in the near future). In our workload characterization study [2] we found that one-third of all re-references to an object occurred within one hour of the previous reference to the same object. Approximately two-thirds (66%) of re-references occurred within 24 hours of the previous request. These results suggest that recency is a characteristic of Web proxy workloads.

2. We chose 5% as a threshold after an in-depth analysis of object size changes [2].

Frequency of Reference. Several recent studies [4][10] have found that some Web objects are more popular than others (i.e., Web referencing patterns are non-uniform). Our characterization study of the Web proxy workload revealed similar results [2]. These findings suggest that popularity, or frequency of reference, is a characteristic that could be considered in a cache replacement decision. We also found that many objects are extremely unpopular. In fact, over 60% of the distinct objects (i.e., unique requests)[3] seen in the proxy log were requested only a single time (we refer to these objects as "one-timers" [4]). Similar observations have been made by other researchers [6][19]. Obviously there is no benefit in caching one-timers. Thus, a replacement policy that could discriminate against one-timers should outperform a policy that does not.

Turnover. One final characteristic that could impact proxy cache replacement decisions is turnover in the active set of objects (i.e., the set of objects that users are currently interested in). Over time the active set changes; objects that were once popular are no longer requested. These inactive objects should be removed from the cache to make space available for new objects that are now in the active set.

5. Experimental Design

This section describes the design of the Web proxy cache simulation study. Section 5.1 introduces the factors and levels that are examined. Section 5.2 presents the metrics used to evaluate the performance of each replacement policy. Section 5.3 discusses other issues regarding the simulation study.

5.1. Factors and Levels

Cache Sizes. The cache size indicates the amount of space available for storing Web objects. We examine seven different levels for this factor: 256 MB, 1 GB, 4 GB, 16 GB, 64 GB, 256 GB and 1 TB. Each level is a factor of four larger than the previous size; this allows us to easily compare the performance improvement relative to the increase in cache size. The smaller cache sizes (e.g., 256 MB to 16 GB) indicate likely cache sizes for Web proxies. The larger values (64 GB to 1 TB) indicate the performance of the cache when a significant fraction of the total requested object set is cached. The largest cache size (1 TB) can store the entire object set and thus indicates the maximum achievable performance of the cache. The other cache sizes can hold approximately 0.06% (256 MB), 0.25% (1 GB), 1% (4 GB), 4% (16 GB), 16% (64 GB) and 64% (256 GB) of the entire object set.

3. The number of distinct objects represents an upper bound on the number of objects that could be cached; the size of these objects (i.e., the unique bytes transferred) indicates the maximum useful cache size.

Cache Replacement Policies. The second factor that we investigate in this simulation study is the replacement policy used by the cache. The replacement policy determines which object(s) should be evicted from the cache in order to create sufficient room to add a new object. There are many proposed cache replacement policies, too many to focus on in this study. We examine six different, previously proposed replacement policies in this study: two "traditional" policies (Least Recently Used and Least Frequently Used), two replacement policies recommended for Web proxy caches (Size [25] and GreedyDual-Size [9]) and two policies designed for other computer systems (Segmented LRU [16] and LRU-K [23]). We chose these six policies because each one considers at least one of the proxy workload characteristics when making a replacement decision.

The Least Recently Used (LRU) policy removes the object which has not been accessed for the longest period of time. This policy works well in workloads which exhibit strong temporal locality (i.e., recency of reference). LRU is a very simple policy requiring no parameterization.

The Least Frequently Used (LFU) policy maintains a reference count for every object in the cache. The object with the lowest reference count is selected for replacement. If more than one object has the same reference count a secondary policy can be used to break the tie (our implementation uses LRU as the secondary policy). One potential drawback of LFU is that some objects may accumulate large reference counts and never become candidates for replacement, even if these objects are no longer in the active set (i.e., the cache could become polluted with inactive objects). To alleviate this problem an aging policy can be implemented [24]. This aging policy requires two parameters: $\mathbf{A_{Max}}$, which places an upper limit on the average reference count for all objects in the cache; and $\mathbf{M_{Refs}}$, which imposes an upper limit on the reference count that can be obtained by a single object. Whenever the average reference count for objects in the cache surpasses $\mathbf{A_{Max}}$, the reference count of each object in the cache is reduced by a factor of two.

The Size policy, designed by Williams *et al.* [25] specifically for Web proxy caches, removes the largest object(s) from the cache when space is needed for a new object. This policy requires no parameterization.

The GreedyDual-Size policy proposed by Cao and Irani [9] considers both the size of the object and its recency of reference when making a replacement decision. Cao and Irani have proposed several variations of this policy [9]. We examine two of these policies: GreedyDual-Size (Hits), which attempts to maximize the hit rate of the proxy cache, and GreedyDual-Size (Bytes) which attempts to maximize the byte hit rate. Neither GreedyDual-Size policy requires parameterization.

The Segmented LRU (SLRU) policy was originally designed for use in a disk cache [16]. We include it in this study because it considers both frequency and

recency of reference when making a replacement decision. The SLRU policy partitions the cache into two segments: an unprotected segment and a protected segment (reserved for popular objects). On the initial request for an object, the object is added to the unprotected segment. When a cache hit occurs, the object is moved to the protected segment. Both segments are managed with the LRU policy. However, only objects in the unprotected segment are eligible for replacement. When objects are removed from the protected segment they are added to the most recently used position in the unprotected segment. This allows the once popular objects to remain in the cache for a longer period of time in case they regain their popularity. If space is needed to add these objects, the least recently used objects in the unprotected segment are removed. This policy requires one parameter, which determines what percentage of the cache space to allocate to the protected segment.

The LRU-K replacement policy proposed by O'Neil *et al.* [23] also considers both frequency and recency of reference when selecting an object for replacement. In an attempt to improve performance this policy retains historical information (the last K reference times) on objects even if they have been removed from the cache. This policy requires two parameters: **K** and **RP**. The LRU-K policy retains the **K** newest reference times for each object. Objects with fewer than **K** references are the first candidates for replacement, followed by the object with the oldest reference time. The parameter **RP** is used to limit the length of time (i.e., the retaining period) for objects that are no longer in the cache. **RP** is needed to prevent the policy from accumulating too much historical data.

5.2. Performance Metrics

In this study two metrics are used to evaluate the performance of the proxy cache: *hit rate* and *byte hit rate*. The hit rate is the percentage of all requests that can be satisfied by searching the cache for a copy of the requested object. The byte hit rate represents the percentage of all data that is transferred directly from the cache rather than from the origin server.

A third metric of interest is response time or latency. We do not use this metric in this study for several reasons. High variability in transfer times for the same object make replacement decisions more difficult. Inaccuracies in our recorded response times also factored in our decision to not use this metric. Furthermore, Cao and Irani found that maximizing the hit rate reduced latency more effectively than policies designed to reduce response times [9].

5.3. The Simulator

This section discusses several remaining topics. Section 5.3.1 discusses the design and validation of our simulator. Section 5.3.2 describes how we determined the length of simulation warm-up. Section 5.3.3 summarizes the assump-

tions we made regarding cacheable and uncacheable requests, cache consistency and other issues involved with object caching.

5.3.1. Simulator Description

Due to the extremely large data set that we use in our simulation study it was necessary to implement the simulator as efficiently as possible. Our focus was on reducing the complexity of the actions performed by the simulator. We began with the simulator used by Arlitt and Williamson [5]. Their simulator utilizes a large array that maintains metadata on each unique object in the data set. By preprocessing the data set and mapping all of the unique object names to distinct integers (as discussed in Section 3.3) their simulator is able to determine in O(1) time if a cache hit or miss has occurred. Their simulator uses a linked list data structure for sorting the objects in the cache according to the given replacement criteria. By combining the linked list with the array of metadata their simulator is able to locate objects in the cache in O(1) time. The bottleneck in their simulator is the time required to update the linked list following a cache hit or miss. The linked list works very well for policies such as LRU, requiring only O(1) time to update the cache contents. Only 30 minutes was required to simulate all 115 million requests when the LRU policy was utilized. However, most policies require O(n) time to perform updates when using a linked list. This is a significant problem when there are millions of unique objects in the cache.

To lessen the effects of this bottleneck we replaced the linked list data structure with a heap. This data structure requires only O(log n) time to perform the necessary reordering of the cache contents following a hit or miss. In order to locate objects in the cache in O(1) time we maintain a pointer from the metadata array to the corresponding object in the heap. It is also necessary to have a pointer from the object back to it's position in the metadata array. As a result of these changes our simulator requires only 45 minutes to simulate all 115 million requests when using the more complicated replacement policies.

An important aspect of any simulation study is validation of the simulator. We took numerous precautions to ensure correctness. For example, the simulator was initially tested using short traces (e.g., 100 requests) which could be verified by hand. The results obtained from our simulator are repeatable. Furthermore, the performance of various policies using our simulator is similar to the results reported in other studies [9][19].

5.3.2. Simulation Warm-up

When monitoring a system only the steady-state behaviour is of interest. During the initial or transient state of a cache simulation, many of the cache misses occur simply because the cache is empty (i.e., cold misses). To identify the transient state we categorized the cache misses that occurred during each day in the trace-

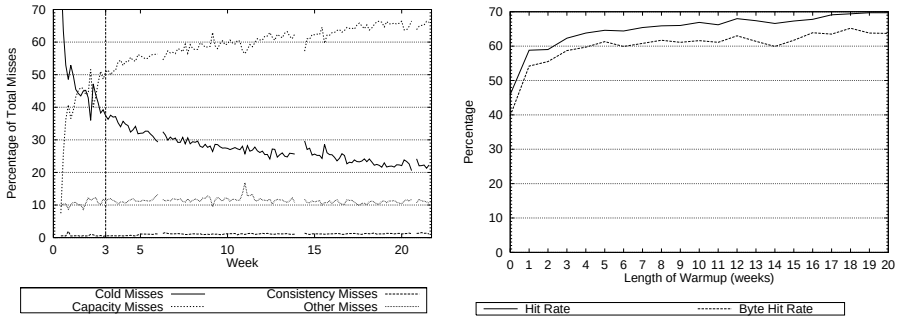
driven simulation. Figure 3(a) indicates the misses that occurred in a 1 GB cache that utilized the LRU replacement policy. The four categories are:

1. **cold miss:** the initial request for an object
2. **capacity miss:** a request for an object that was in the cache but has since been evicted
3. **consistency miss:** a request for an object in the cache that has been modified; due to the modification the object must be transferred from the origin server
4. **other misses:** requests that do not fit in one of the above categories (e.g., requests for cgi-bin objects)

Figure 3(a) confirms that initially most misses occur because the cache is empty. Once the cache has had time to add objects the percentage of cold misses decreases rapidly. Due to the limited cache space capacity misses increase dramatically. After several weeks capacity misses have become the dominant source of all cache misses. The continual increase in capacity misses over time indicates that the cache size is too small, at least when the LRU replacement policy is used. Even after five months cold misses still account for a significant percentage of all misses (over 20% of the misses in a 1 GB LRU cache). This phenomenon is due to the continuous growth in the number of Web objects and the changing interests of users. Throughout the trace the percentage of consistency misses and other misses stays relatively constant. Finally, Figure 3(a) illustrates the need for long duration traces in supporting useful proxy cache performance analyses.

Figure 3(b) shows the performance of an infinite-sized cache. With an infinite-sized cache no capacity misses occur. Thus instead of categorizing the misses that occur we monitor the effects of different warm-up periods on cache performance. In Figure 3(b) the first x weeks of the trace are used to warm the cache; following the warm-up period the cache is simulated for an additional week during which performance statistics were collected. Figure 3(b) shows that both the hit rate and byte hit rate of the cache are quite poor during the initial few weeks, but are increasing rapidly (as more objects get added to the cache the probability of a miss occurring decreases). After the first few weeks of simulation the performance of the cache is relatively stable. The continued growth in the hit and byte hit rates is due to the increased sharing of object in the cache by an expanding user base.

After analyzing the results from Figure 3(a) and (b) we chose to use the first three weeks of trace data (8% of all requests) to warm the cache. During this period the simulated cache operates in its usual manner but no statistics are collected. Statistics are only collected once the warm-up period has finished. We use the same warm-up period in all experiments.



(a) 1 GB LRU Cache (b) Infinite-sized Cache

Figure 3. Determining the Simulation Warm-up

5.3.3. Cacheability and Consistency

In our study all requests except for aborted (i.e., incomplete) transfers are used to drive the simulation. We divide the completed transfers into two groups: cacheable requests and uncacheable requests. The cacheable requests are identified by the response status code recorded in the data set. According to the HTTP Specification, responses with a status code of 200 (Successful), 203 (Non-authoritative Information), 300 (Multiple Choices), 301 (Moved Permanently) and 410 (Gone) (except for dynamic requests) are considered to be cacheable [13]. Some of these responses could potentially be uncacheable. Feldmann *et al*. [12] found that many HTTP requests and responses may be uncacheable due to the inclusion of various header fields (e.g., cookies). Unfortunately the data we collected did not include the headers that would indicate which of these responses were uncacheable. By assuming that all of these responses are cacheable we are determining an upper bound on the achievable hit rates and byte hit rates. We also consider status 304 responses to be cacheable even though no data is transferred. We use status 304 responses to update the state information maintained by the proxy cache on the object being validated. We believe that this information helps the replacement policy in determining the active set of objects. This does not imply that the cache would not forward Get-If-Modified requests to the origin server should such actions be necessary. All remaining responses are considered to be uncacheable. All requests for uncacheable responses result in cache misses.

Our simulator does not perform cache consistency functions such as asynchronous validations. We do update the state information on objects that we know have changed. Since our data set does not include Time-to-Live information we do not collect statistics on the number of validation messages that would occur. Also, we do not consider issues like security or authentication. These issues, along with consistency, require more in-depth coverage than we can provide in this study. Since our simulator does not provide all of the functionality of a real proxy cache we expect our results (i.e., hit rates and byte hit rates) to be somewhat optimistic.

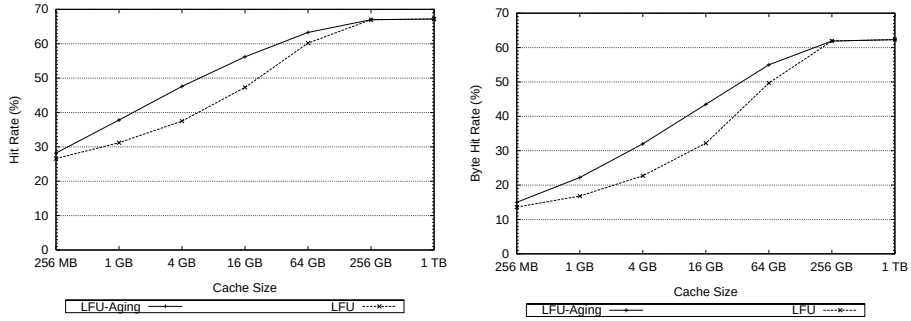
6. Simulation Results

This section provides the simulation results of the proxy cache replacement policy study. Section 6.1 examines the effects of parameterization on the performance of the replacement policies. Section 6.2 compares the performance of the different cache replacement policies. All of our results are shown in graphical format (Figure 4 - Figure 7). Each figure consists of two graphs, with the graph on the left indicating the achieved hit rate for a cache of a particular size while the graph on the right shows the achieved byte hit rate for a similarly configured cache.

6.1. Parameterization

Three of the replacement policies under study (LFU, SLRU and LRU-K) require parameterization in order to function properly. We examine each policy individually in an attempt to determine the effects of each parameter on the performance of the replacement policy.

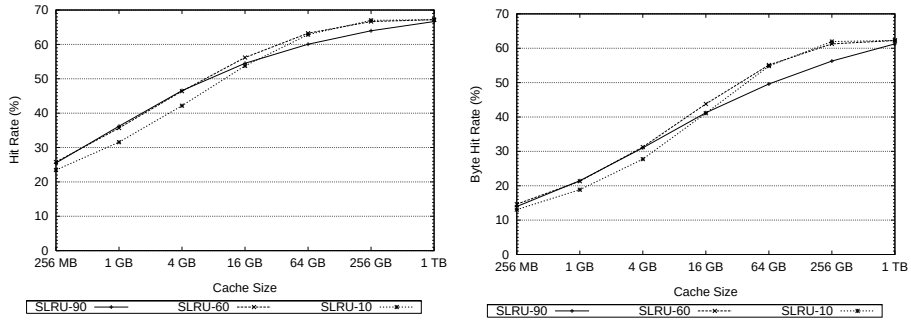
The LFU replacement policy requires two parameters, $\mathbf{A_{Max}}$ and $\mathbf{M_{Refs}}$, in order to age objects in the cache. We experimented with different settings for these parameters. We found that as long as the aging policy was periodically invoked (e.g., on a daily basis) the choice of values for these parameters did not have a significant impact on performance of the LFU-Aging policy. Figure 4 compares the performance of LFU without aging (LFU) to LFU with aging (LFU-Aging; $\mathbf{A_{Max}}$=10, $\mathbf{M_{Refs}}$=8,192). LFU-Aging clearly outperforms LFU. These results indicate that it is important for the replacement policy to be able to make changes to the active set of objects. The performance of LFU is similar to that of LFU-Aging in two situations. When cache sizes are large (e.g., 256 GB and up) few replacement decisions are needed and cache pollution is not a factor so the policies have similar performance. When cache sizes are very small (e.g., 256 MB), adding a single large object can result in the removal of a large number of smaller objects reducing the effects of cache pollution.



(a) Hit Rate (b) Byte Hit Rate

Figure 4. Analysis of LFU performance

The SLRU replacement policy uses a single parameter to set the size of the protected cache segment. We examined a wide range of values for this parameter. Figure 5 shows the results when either 10, 60 or 90 per cent of the available cache space is reserved for the protected segment. There is one curve on the graph for each parameter setting. For example, the SLRU-90 curves indicate the hit and byte hit rate achieved when the SLRU policy reserves 90% of the cache space for the protected segment. Figure 5 reveals that altering the size of the protected segment does affect the performance of the cache. If the protected segment is too small (e.g., 10% of total space) then the policy gives significantly more weight to the recency workload characteristic; as a result the policy behaves like LRU. When the protected segment is too large (e.g., 90% of total space) the policy considers the frequency characteristic to be significantly more important than recency. This causes the policy to retain a lot of extremely popular objects for extended periods of time, much like the LFU policy. The SLRU policy performs best when a balance is found that allows for popular objects to be retained for long periods of time without becoming susceptible to pollution. In our study we found the best results occurred for a protected segment size of 60 per cent. Since similar results were obtained in the original study using disk caches [16] we believe that this parameter setting is not specific to our data set.

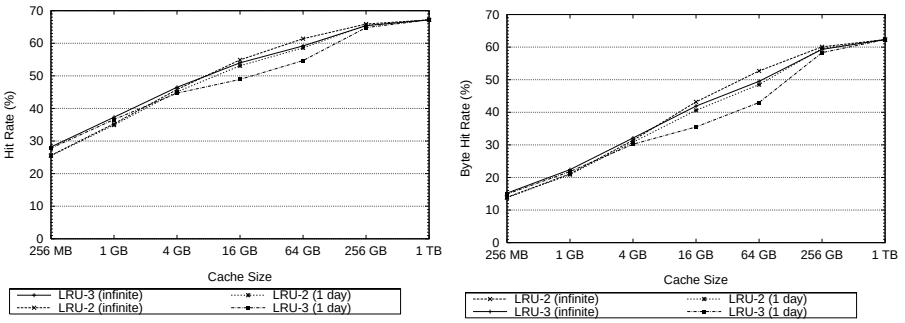


(a) Hit Rate (b) Byte Hit Rate

Figure 5. Analysis of SLRU performance

The final replacement policy under study that requires parameterization is LRU-K. LRU-K requires two parameters. The first parameter, **K**, is the number of reference times to retain for an object. The second parameter, **RP**, is the length of time to keep this information. We examine several different configurations: retaining either the last two (**K**=2) or the last three (**K**=3) reference times to each object, and retaining history information for either one day (**RP** = 1 day) or forever (**RP** = infinite). The results of these experiments are shown in Figure 6. With smaller cache sizes (e.g., 256 MB - 1 GB) retaining information on the last three references (for any length of time) provides higher hit rates and slightly higher byte hit rates. This can be attributed to the policy giving higher priority to the most popular objects. As the cache size gets larger it becomes necessary to retain information for a longer period of time in order to achieve better perfor-

mance. Requiring less information about each object (i.e., using only the last two reference times) also improves performance for the larger cache sizes.



(a) Hit Rate (b) Byte Hit Rate

Figure 6. Analysis of LRU-K performance

6.2. Comparison of Replacement Policies

In this section we compare the performance of all of the replacement policies. To make the comparison easier we include only the "best" curve for each of the policies that required parameterization (i.e., we use LFU-Aging ($\mathbf{A_{Max}}$=10, $\mathbf{M_{Refs}}$=8,192) for the LFU policy, SLRU-60 for the SLRU policy and LRU-2 with infinite history for the LRU-K policy). We have also sorted the legend in each graph by the performance of the policies. For example, in Figure 7(a), the first policy listed in the legend is GDS-Hits. The GDS-Hits policy achieved the highest hit rate. The last policy in the legend is LRU. LRU obtained the lowest hit rate of the policies that we examined.

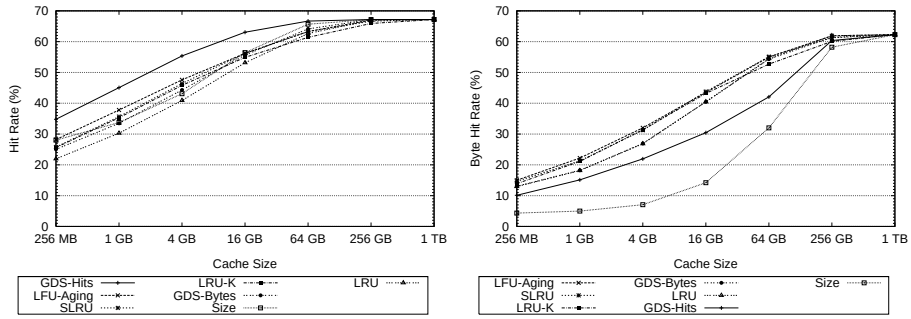
Figure 7(a) compares the hit rates achieved by each policy. The results indicate that the maximum achievable hit rate during the simulation period is 67% (obtained by all policies with a cache size of 1 TB). The remaining 33% of requests are for the initial requests for objects, for uncacheable objects (e.g., output from dynamic or cgi objects) or for the updates of objects which have been modified and cannot be served from the cache. Figure 7(a) shows that even small caches can perform quite well if the correct replacement policy is used. For example, a 256 MB cache using the GreedyDual-Size (Hits) policy achieved a hit rate of 35% which is 52% of the maximum achievable rate. This rate was achieved while allowing for only 0.06% of the entire object set size to be cached.

Figure 7(a) shows that the GreedyDual-Size (Hits) policy is vastly superior to other policies when hit rate is used as the metric. For small cache sizes (256 MB to 16 GB) GDS-Hits outperforms all other policies by at least 6 percentage points. The success of the GDS-Hits policy can be attributed to two characteristics of the policy: it discriminates against large objects, allowing for more small objects to be cached; and it ages the object set to prevent cache pollution from occurring. During our experiments we monitored the number of objects kept in

the cache under the various replacement policies. With a 256 MB cache the GDS-Hits policy held 170,000 objects (average object size 1.5 KB) at the end of the simulation. The LFU-Aging policy, by comparison, held only 20,000 objects (an average object size of 13 KB). By inflating the number of objects kept in the cache GDS-Hits increases the probability that an object will be in the cache when it is requested. The other size-based policies (GDS-Bytes and Size) have much lower hit rates. GDS-Bytes attempts to improve the byte hit rate by favoring larger objects (it kept 26,000 objects in the 256 MB cache). Thus, the lower hit rate of GDS-Bytes is not unexpected. The Size policy discriminates even more harshly against large objects. In the 256 MB cache the Size policy collected over 900,000 objects (average object size 300 bytes). However, the Size policy failed to age the object set. The poor performance of the Size policy can therefore be attributed to cache pollution.

The frequency-based replacement policies (LFU-Aging, SLRU and LRU-K) achieve similar hit rates. Since these policies do not discriminate against large objects (they do not consider object size at all) they require about four times as much cache space to achieve hit rates similar to the GDS-Hits policy. However, the frequency-based policies are able to discriminate against one-timers, retain popular objects for longer time periods and age the object set to prevent cache pollution. These characteristics allow frequency-based policies to outperform recency-based policies.

The only recency-based policy that we examine is LRU. LRU achieves the lowest hit rate since it does not consider enough information when making replacement decisions and therefore tends to make poorer choices. Because of this the LRU policy requires almost eight times as much cache space as the GDS-Hits policy to achieve similar hit rates to the GDS-Hits policy. One positive feature of LRU is that it ages the object set which prevents cache pollution.



(a) Hit Rate (b) Byte Hit Rate

Figure 7. Comparison of all Replacement Policies

Figure 7(b) shows the achieved byte hit rates for the replacement policies under study. Figure 7(b) reveals a maximum byte hit rate during the simulation period

of 62% for the data set under study. The remaining 38% of the data needed to be transferred across the external network link. The results also indicate that it is more difficult to achieve high byte hit rates than high hit rates. For example, a 256 MB cache can achieve a byte hit rate of 15% which is only one quarter of the maximum achievable byte hit rate.

According to the results in Figure 7(b) the frequency-based policies (LFU-Aging, SLRU and LRU-K) are the best choice for reducing network traffic. The three frequency-based policies achieve similar byte hit rates, approximately 2-4 percentage points higher than LRU. The frequency-based policies work well because they do not discriminate against the large objects which are responsible for a significant amount of the data traffic. Frequency-based policies also retain popular objects (both small and large) longer than recency-based policies, another reason that frequency-based policies achieve higher byte hit rates.

The LRU and GDS-Bytes policies have almost identical performance in terms of byte hit rates. LRU does not discriminate against large objects which allows it to outperform size-based policies which do. Although GDS-Bytes is a size-based policy it has been designed to treat large objects more favorably in an attempt to improve the byte hit rate. Both LRU and GDS-Bytes require about twice the cache space to achieve byte hit rates comparable to the frequency-based policies.

Since size-based policies (generally) discriminate against large objects it is not surprising that these policies have the worst byte hit rate performance. The GDS-Hits policy requires four times more cache space to achieve the same byte hit rate as a frequency-based policy. The byte hit rate of the Size policy is even worse than GDS-Hits because of more unfavorable treatment of large objects and cache pollution.

7. Contributions and future work

This paper has presented our performance study of a Web proxy cache. This study is the first to include the effects of high-speed cable modems by clients and also has the largest data set of any proxy workload. Trace-driven simulations were used to evaluate the performance of different cache replacement policies. Our results indicate that size-based policies achieve higher hit rates than other policies while frequency-based policies are more effective at reducing external network traffic. The results show that a properly chosen replacement policy can reduce the purchase cost of Web proxy caches by making better use of available resources. The results also indicate that it is important to examine the performance of replacement policies over extended time durations to test for side effects such as cache pollution.

The intent of this paper was not to promote the use of a single replacement policy for Web proxies. Instead, our goal was to explain the performance of different

policies by examining the workload characteristics that each policy used or did not use when making replacement decisions. This information can be applied in the design of a new replacement policy that achieves both high hit rates and byte hit rates.

There are many open issues regarding Web proxy performance and Web caching. Future work in this area could include implementing new replacement policies and other cache management techniques [3], for example, in actual caching products. Other issues that require additional investigation include examining the relationship between hit rates and latency reduction for end users, implementing a more efficient consistency mechanism [17][18] and adding more functionality to Web proxy caches (e.g., accounting and security). Finally, much effort will be required to ensure that the majority of Web objects remain cacheable as the Web evolves.

Acknowledgments

The authors would like to thank Mike Rodriquez of HP Labs and all the people in the Telecommunication Platforms Division (TPD) who supplied us with access logs; John Dilley, Gita Gopal and Jim Salehi of HP Labs and the anonymous reviewers for their constructive comments on the paper; and Greg Oster of the University of Saskatchewan for his assistance with the development of the simulator.

References

[1] J. Almeida and P. Cao, ''Measuring Proxy Performance with the Wisconsin Proxy Benchmark'', Technical Report, University of Wisconsin Department of Computer Science, April 1998.

[2] M. Arlitt, R. Friedrich, and T. Jin, ''Workload Characterization of a Web Proxy Cache in a Cable Modem Environment'', to appear in *ACM SIGMETRICS Performance Evaluation Review*, August 1999.

[3] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich and T. Jin, "Evaluating Content Management Techniques for Web Proxy Caches", *Proceedings of the Second Workshop on Internet Server Performance,* Atlanta, GA, May 1999.

[4] M. Arlitt and C. Williamson, ''Internet Web Servers: Workload Characterization and Performance Implications'', *IEEE/ACM Transactions on Networking,* Vol. 5, No. 5, pp. 631-645, October 1997.

[5] M. Arlitt and C. Williamson, ''Trace-Driven Simulation of Document Caching Strategies for Internet Web Servers'', *The Society for Computer Simulation SIMULATION Journal*, Vol. 68, No. 1, pp. 23-33, January 1997.

[6] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm, ''Enhancing the Web's Infrastructure: From Caching to Replication'', IEEE Internet Computing, Vol. 1, No. 2, pp. 18-27, March-April 1997.

[7] P. Barford, A. Bestavros, A. Bradley and M. Crovella, "Changes in Web Client Access Patterns: Characteristics and Caching Implications", to appear in *World Wide Web,* Special Issue on Characterization and Performance Evaluation, 1999.

[8] T. Berners-Lee, R. Cailliau, A. Luotenen, H. Nielsen, and A. Secret, ‘‘The World-Wide Web’’, Communications of the ACM, 37(8), pp. 76-82, August 1993.

[9] P. Cao and S. Irani, ‘‘Cost-Aware WWW Proxy Caching Algorithms’’, *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS)*, Monterey, CA, pp. 193-206, December 1997.

[10] C. Cunha, A. Bestavros, and M. Crovella, ‘‘Characteristics of WWW Client-based Traces’’, Technical Report TR-95-010, Boston University Department of Computer Science, April 1995.

[11] B. Duska, D. Marwood, and M. Feeley, ‘‘The Measured Access Characteristics of World-Wide Web Client Proxy Caches’’, *Proceedings of USENIX Symposium of Internet Technologies and Systems (USITS)*, Monterey, CA, pp. 23-35, December 1997.

[12] A. Feldmann, R. Cáceres, F. Douglis, G. Glass and M. Rabinovich, “Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments”, Proceedings of IEEE Infocom ‘99, New York, NY, pp. 107-116, March 1999.

[13] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, ‘‘RFC 2068 - Hypertext Transfer Protocol - - HTTP/1.1’’, January 1997.

[14] S. Gribble and E. Brewer, ‘‘System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace’’, *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS),* Monterey, CA, pp. 207-218, December 1997.

[15] V. Jacobson, ‘‘How To Kill the Internet’’, SIGCOMM ‘95 Middleware Workshop, Cambridge, MA, August 1995.

[16] R. Karedla, J. Love and B. Wherry, ‘‘Caching Strategies to Improve Disk System Performance’’, IEEE Computer, Vol. 27, No. 3, pp. 38-46, March 1994.

[17] B. Krishnamurthy and C. Wills, ‘‘Study of Piggyback Cache Validation for Proxy Caches in the World-Wide Web’’, Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS), Monterey, CA, pp. 1-12, December 1997.

[18] C. Liu and P. Cao, ‘‘Maintaining Strong Cache Consistency in the World-Wide Web’’, Proceedings of the 17th IEEE International Conference on Distributed Computing Systems, May 1997.

[19] P. Lorenzetti and L. Rizzo, ‘‘Replacement Policies for a Proxy Cache’’, Technical Report, Universita di Pisa, December 1996.

[20] A. Luotonen, *Web Proxy Servers*, Prentice Hall, Upper Saddle River, NJ, 1998.

[21] C. Maltzahn, K. Richardson and D. Grunwald, “Reducing the Disk I/O of Web Proxy Server Caches”, to appear in the *USENIX Annual Technical Conference*, Monterey, CA, June 1999.

[22] C. Maltzahn and K. Richardson, ‘‘Performance Issues of Enterprise Level Web Proxies’’, Proceedings of the 1997 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems, Seattle, WA, pp. 13-23, June 1997.

[23] E. O’Neil, P. O’Neil and G. Weikum, ‘‘The LRU-K Page Replacement Algorithm for Database Disk Buffering’’, *Proceedings of SIGMOD ‘93*, Washington, DC, May 1993.

[24] J. Robinson and M. Devarakonda, ''Data Cache Management Using Frequency-Based Replacement'', *Proceedings of the 1990 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, Boulder, CO, pp. 134-142, May 1990.

[25] S. Williams, M. Abrams, C. Standridge, G. Abdulla, and E. Fox, ''Removal Policies in Network Caches for World-Wide Web Documents'', *Proceedings on ACM SIGCOMM '96*, Stanford, CA, pp. 293-305, August 1996.

[26] R. Wooster and M. Abrams, ''Proxy Caching that Estimates Page Load Delays'', Proceedings of the 6th International World-Wide Web Conference, Santa Clara, CA, April 1997.

[27] World-Wide Web Consortium, ''Replication and Caching Position Statement'', August 1997. Available at: `http://www.w3.org/Propogation/activity.html`

Vitae

Martin Arlitt is a research engineer at Hewlett-Packard Laboratories in Palo Alto, California, USA. His general research interests are computer networks and computer systems performance analysis. His specific interests include performance issues for the World-Wide Web. He graduated from the University of Saskatchewan in 1996.

Rich Friedrich is a Senior Researcher at Hewlett-Packard Laboratories in Palo Alto, California, USA. He has held several research and product development positions within Hewlett-Packard including leading the system performance engineering team that developed and optimized the first commercial RISC based systems in the mid-1980's and the design of a distributed measurement system for the OSF DCE in the early 1990's. His current interests are in QoS control mechanisms for Internet services, distributed systems and the visualization of large data sets. He was the program co-chair for the IEEE Sixth International Workshop on Quality of Service. He attended Northwestern University and Stanford University.

Tai Jin is a research engineer at Hewlett-Packard Laboratories in Palo Alto, California, USA. He was a key contributor to the HP-UX networking projects in the late 1980's and was involved in the creation of the HP intranet. During that time he developed a tool which revolutionized network software distribution within the company. He has also created several useful services accessible through the World-Wide Web. His interests include networked systems, exploiting the World-Wide Web, performance tuning, creating useful tools, and the stock market. He graduated from Columbia University in 1984.