Evaluating Content Management Techniques for Web Proxy Caches

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

1. 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. Current Web proxy caches utilize simple replacement policies to determine which files to retain in the cache. We utilize a trace of client requests to a busy Web proxy in an ISP environment to evaluate the performance of several existing replacement policies and of two new, parameterless replacement policies that we introduce in this paper. Finally, we introduce Virtual Caches, an approach for improving the performance of the cache for multiple metrics simultaneously.

2. INTRODUCTION

World-Wide Web requests have continued to grow at an exponential rate and are already the dominant workload component for Internet traffic [21]. The implications of this are clear: without good systems design network elements and servers will become bottlenecks. Since the majority of Web objects are static, caching them at HTTP proxies can reduce both network traffic and response time. HTTP proxies serve as intermediaries between the client browser and origin Web servers, which may be geographically distant. However, current Web proxy cache servers are achieving relatively low hit rates and contribute additional delays to servicing Web requests.

Our research has focused on developing a quantitative understanding of Web traffic and its implications on server and network design. These quantitative results yield insights into the behavior of current proxy caches and identify improvements for next generation caching.

In general the research described in this paper addresses the following questions:

- How effective are current proxy cache replacement policies for real workloads?
- What new replacement policies and approaches can improve cache effectiveness?
- How will higher speed access networks affect workloads?

The results in this paper are based on the most extensive Web proxy workload characterization yet reported in the literature. All subscriber client Web accesses from a single ISP were measured for a duration of five months. During this period 115 million requests for 1.3 terabytes of data were measured and analyzed. Furthermore, all clients accessed this proxy using high-speed cable modems. This study may therefore provide a glimpse of future Web traffic as access network speeds increase. The workload characterization itself is not described in detail in this paper but can be found in [3].

Through our workload study we have discovered several characteristics that affect proxy caches. Six of these are discussed in this paper. We use these characteristics in a simulator to evaluate current cache replacement policies with respect to two important performance metrics: hit rate and byte hit rate. These two metrics correlate to decreased client response time and decreased backbone network utilization respectively. Both reduce the origin server load.

This paper extends our previous results with the following. From insights gained from the workload we developed two new replacement policies that are optimized for a single metric and achieve the best performance for hit rate and byte hit rate. These new replacement policies utilize frequency to achieve higher performance but are neither susceptible to cache pollution nor require parameterization. Finally, we developed a new approach to Web cache management. This novel approach uses a number of virtual caches (i.e., cache partitions) to improve the global performance of the cache across multiple metrics simultaneously.

Our workload characterization study [3] builds on previous work in this area [10][12]. We use the identified characteristics to evaluate the strengths and weaknesses of different replacement policies. There have been a number of recent efforts to design more effective replacement policies for Web caches [6][17][23][24]. These studies have been limited to either short-term traces of busy proxies [6] or long-term traces of relatively inactive proxies [17][23][24]. Caching is more important in busy environments where the connection to the Internet is the performance bottleneck. Long-term traces are needed to evaluate how a replacement policy adapts to changes in the workload. Our study explains the performance of the examined replacement policies in terms of the underlying workload characteristics. There are other issues affecting Web proxy caches that are not covered in this paper. For example, a cache should provide users with objects that are consistent with those available from the origin server. Several more efficient solutions to this problem have been proposed [14][16]. Other researchers have investigated the performance of actual proxy cache implementations [1][18].

The remainder of this paper is organized as follows. Section 3 introduces the data set utilized for our workload characterization as well as our trace-driven simulations. Section 4 discusses the results of our workload characterization and the implications on proxy caches. Section 5 provides the experimental design of our simulations while Section 6 presents the results. Section 7 introduces Virtual Caches, a new method of managing Web caches. We conclude in Section 8 with a summary of the paper and a discussion of future work.

3. DATA COLLECTION AND REDUCTION

3.1 Data Collection Site

The site under study is an Internet Service Provider (ISP) that offers interactive data services to residential and business subscribers. These subscribers utilized cable modems to connect to the ISP's server complex. Several thousand subscribers utilized the system during the data collection period. All subscriber requests for Web objects (e.g., HTTP, FTP, and Gopher requests) were forwarded to a single server running a commercial proxy software package. This proxy server includes a file cache so some of the client requests were satisfied within the server complex. On a cache miss the proxy retrieved the object from an origin server on the Internet. All requests for the ISP's own web site issued by this group of subscribers went through the proxy. Since this site was quite popular with the subscribers the hit rate in the proxy cache is higher than traditionally seen in proxy caches.

The cable modems utilized at this site had peak bandwidths reaching several megabits per second. This is several orders of magnitude more than is achieved by traditional dialup modems. The increased access bandwidths made proxy caching important for reducing user latency as the ISP's connection to the Internet was the main bottleneck in the system.

3.2 Data Collection

The access logs of the proxy server described in Section 3.1 were collected for this study. These logs contain information on all client requests made between Jan 3rd, 1997 and May 31st, 1997. Each entry in an access log contained information on a single request received by the proxy. The recorded information included the client IP address, the time of the request, the client's request (method, URL, HTTP version), the response status from the proxy and the origin server, the amount of header and content data transferred, and the time required for the proxy to complete its response. In total 117,652,652 requests were handled by the proxy during this five month period.

Unfortunately the access logs did not contain all of the information of interest. For example, the logs contained no information that would allow us to correctly identify all aborted requests or file modifications. Thus to identify such occurrences we were forced to use heuristics based on observed changes in the size of the requested file. Furthermore, we have limited information on the HTTP request and response headers exchanged between the client and origin server. Because of this we cannot determine which requests were marked as uncacheable by clients or which responses were tagged as uncacheable or private (e.g., contained cookies) by the origin servers. Feldmann et. al. have shown that these types of requests and responses are quite common and thus greatly reduce the effectiveness of proxy caching [11]. In this paper we do not consider the impact of these headers on the performance of the cache. Instead our results should be seen as motivation for using cache control headers correctly, or for developing new applications or technologies to meet the needs of users, access providers, cache operators, and content providers.

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 our workload analyses and caching simulations could be completed in a reasonable amount of time. We performed these reductions in two ways: by storing data in a more efficient manner (e.g., map all distinct URLs to unique integers), and by removing information of little value (e.g., we kept only GET requests which accounted for 98% of all requests and 99.2% of the content data transferred). During the reduction process it became apparent that there were too many unique URLs to map all of them. Instead we mapped only the cacheable URLs. We considered a URL to be cacheable if it did not contain substrings such as 'cgi-bin' or '?', if it did not have a file extension such as '.cgi', and if the origin server response contained an appropriate status code (e.g., 200 Success). The overall statistics for the reduced log are given in Table 1. The reduced log is only 4.5 GB in size, 1.5 GB when compressed.

4. KEY WORKLOAD CHARACTERISTICS

This section introduces a number of Web proxy workload characteristics and discusses their potential impact on proxy caching and cache replacement decisions. These characteristics are described in detail in our workload characterization study [3].

Cacheable Objects: In order for caching to improve Web performance it is vital that most client requests be for cacheable objects. Table 1 indicates that 92% of all requests were for cacheable objects (9,020,632 requests, or 8%, were uncachable), while 96% of the data transferred was cacheable. As we mentioned in Section 3.2 this is an estimate of how many cacheable requests we could see if lower level issues such as consistency and privacy were handled efficiently and correctly.

Object Set Size: Table 1 indicates 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 avoid adding excessive latency to the response. The proxy must also be able to update its state efficiently on a cache hit, miss or replacement. Furthermore, Table 1 indicates that storing all requested objects in cache (one approach for improving the hit rate) would require an enormous amount of disk space; the 16 million unique cacheable objects have a cumulative size of 389 GB.

Object Sizes: One of the obstacles for Web caching is working effectively with variable-sized objects. The object size distribution in this data set is heavy-tailed with the tail weight estimated at α=1.5. Thus while most of the requested objects are small (the median object size in this data set was 4 KB) some extremely large objects were accessed. The largest object requested during the measurement period was a 148 MB video. We also observed requests for HTML objects over 10 MB in size and images over 90 MB in size. 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, cache a few large objects which could increase the byte hit rate, or some combination of the two.

Recency of Reference: 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 of re-references occurred within 24 hours of the previous request. These results suggest that recency is an important characteristic of Web proxy workloads.

Frequency of Reference: Several recent studies [4][8] 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. These findings suggest that popularity, or frequency of reference, is a characteristic that should be considered in a cache replacement decision.

In our study we found that many objects are extremely unpopular. In fact over 60% of the distinct objects (i.e., unique requests) seen in the proxy log were requested only a single time (we refer to these objects as "one-timers" [4]). Similar results have been reported by other researchers [5][15][17]. Obviously there is no benefit in caching one-timers. Thus, a replacement policy that can discriminate against one-timers should outperform a policy that cannot.

Table 1: Summary of Trace Characteristics

Access Log Duration	January 3rd - May 31st, 1997
Total Requests	115,310,904
Total Content Bytes	1,328 GB
Unique Cacheable Requests	16,255,621
Total Uncacheable Requests	9,020,632
Unique Cacheable Content Bytes	389 GB
Total Uncacheable Content Bytes	56 GB

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 in the active set.

5. EXPERIMENTAL DESIGN

The next step in our study utilized trace-driven simulation to evaluate the importance of the identified workload characteristics in making replacement decisions. This section provides an overview of the experimental design used for this simulation study.

5.1 Factors and Levels

In our simulation study we focus on two factors: the size of the cache and the cache replacement policy. We discuss each of these factors in turn.

Cache Size: The cache size indicates the amount of space available for storing Web objects. We examine a range of sizes: 256 MB, 1 GB, 4 GB, 16 GB, 64 GB, 256 GB and 1 TB. The smallest size (256 MB) represents the performance of a memory cache. The intermediate cache sizes (e.g., 1 GB to 16 GB) indicate likely cache sizes for Web proxies. The largest cache size (1 TB) can store the entire object set and thus approximates the performance of an infinite-sized cache.

Cache Replacement Policy: A replacement policy is an algorithm for determining which objects to evict from the cache when space is needed for a newly requested object. In this paper we restrict ourselves to examining four different, previously proposed replacement policies and two new policies that utilize frequency:

Least-Recently-Used (LRU) - replaces the object requested least recently.

This traditional policy is the most often used in practice. Previous studies have found that LRU does not work as well as other policies for Web proxy caches since it considers only a single workload characteristic [6][23].

SIZE [6]- replaces the largest object.

This strategy tries to minimize the miss ratio by replacing one large object rather than many smaller ones. However, some of the small objects brought into the cache may never be accessed again. The SIZE strategy does not provide any mechanism to evict such objects, which leads to pollution of the cache.

GreedyDual-Size (GD-Size) [6]- replaces the object with the lowest utility.

This strategy replaces the object with the smallest key value for a certain utility (cost) function. When an object i is requested it is given a priority key K_i computed as follows:

$$K_i = C_i / S_i + L$$

where

- C_i is the cost associated with bringing object i into the cache.
- S_i is the object size.
- L is a running age factor that starts at 0 and is updated for each replaced (evicted) object f to the priority key of this object in the priority queue: i.e., $L = K_f$.

Cao and Irani identified several variations of the GreedyDual-Size policy. To get the best hit rate with GD-Size the Cost function for each object is set to 1. In this way, larger objects have a smaller priority key than smaller ones, and are more likely to be replaced if they are not referenced again in the near future. To maximize the hit rate it is more "profitable" to replace one large object than many small objects. This strategy is denoted GD-Size(1). To get the best byte hit rate with GD-Size the Cost function is set to $2 + S_i/536$. This function estimates the number of network packets sent and received to satisfy a cache miss for a requested object, and therefore tries to minimize network traffic. This strategy is denoted GD-Size(Packets).

LFU - replaces the least frequently used object.

The LFU policy maintains a frequency count for each object in the cache. We also examine the LFU-Aging policy [20], which avoids cache pollution by reducing the frequency count of each cached object by a factor of 2 whenever the average frequency count exceeds a threshold parameter. Since some objects may be extremely popular, they might stay in the cache for longer than desired or have significant influence on the current frequency count. To prevent this situation a second parameter limits the maximum frequency count an object can accumulate. To achieve good performance these parameters should be set based upon an analysis of the workload offered to the cache.

We chose these policies because each one considers at least one of the proxy workload characteristics when making a replacement decision. We also examined two policies designed for other computer systems (S-LRU [13] and LRU-K [19]). We found both of these policies had similar performance to LFU-Aging [2]. Due to space constraints we do not include the S-LRU and LRU-K results in this paper.

During our examination of these existing policies we made several observations that led to the development of two new parameterless (i.e., no manual tuning required to achieve good performance) replacement policies. These two new policies are:

GreedyDual-Size with Frequency (GDSF)

The GD-Size policies perform well, but do have one significant shortcoming: they do not take into account how many times the object was accessed in the past. For example, consider how GD-Size(1) handles two different objects of the same size. If they are requested at about the same time, they are inserted into a priority queue with about the same key value. The object f_1, which was accessed n times in the past will get the same K_i value as the object f_2 accessed for the first time. In the worst case scenario, f_1 will be replaced instead of f_2.

The GD-Size algorithm can be improved to reflect object access patterns by incorporating a frequency count F_i in the computation of K_i: An analysis of this policy can be found in [7].

$$K_i = F\mathrm{i} * C\mathrm{i} / S_i + L$$

This policy achieves the best hit rate when C_i=1. We denote this strategy as GDSF-Hits.

Least Frequently Used with Dynamic Aging (LFU-DA)

The results from [2] (summarized in Section 6) show that frequency-based policies (e.g., LFU-Aging) achieve the highest byte hit rates. These policies accomplish this by retaining the most popular objects, regardless of object size. Unfortunately, all of the frequency-based policies that we examined in [2] require parameterization to perform well. Parameterless policies are preferable as they are generally less complex and easier to manage. Thus, we replaced the tuneable aging mechanism of the LFU-Aging policy with a dynamic mechanism (i.e., the inflation factor, L, used by the GD-Size policy). We call this new policy Least Frequently Used with Dynamic Aging. LFU-DA calculates the key value K_i for object i using the following equation:

$$K_i = C_\mathrm{i} * F_i + L$$

With C_i set to 1 this equation uses only the frequency count and the inflation factor (for aging objects) to determine the key value of an object. (We did not examine other values for C_i.) The LFU-DA policy may prove useful in other caching environments where frequency is an important characteristic but where LFU has not been utilized due to cache pollution concerns.

5.2 Performance Metrics

The two most common metrics used to evaluate proxy cache performance are:

- **Hit rate** - the number of requests satisfied from the proxy cache as a percent of total requests.
- **Byte hit rate** - the number of bytes that the proxy cache served directly as a percent of the total number of bytes for all requests.

5.3 Other Design Issues

When monitoring a system only the steady-state behaviour is of interest. During the initial or transient state of the simulation many of the cache misses occur simply because the cache is empty (i.e., cold misses). After monitoring the influence of cold misses on the overall miss rate on a day-to-day basis we chose to use the initial three weeks of the data set to warm-up the simulator. We use this same warm-up period for all simulations.

In this study all requests except for aborted transfers (about 10% of all requests) were used to drive the simulations. All uncacheable requests are counted as cache misses. All cacheable requests, including those with status 304 responses are used to update the state information maintained by the cache for the requested object. In the case of 304 responses only the hit rate is affected as no content data is transferred. We believe that using this information helps the replacement policy to better determine the active set of objects.

In this study we do not investigate the performance implications of the proxy having to perform validations. All file modifications that we determined from the logs result in consistency misses in the simulations. As we discussed in Section 3.2 we do not have any information on cache control headers. Since our simulator does not provide all of the functionality of a real proxy cache our results (e.g., hit rates and byte hit rates) will be optimistic. However, we believe our results indicate the level of performance that could be achieved if other issues, such as the proper use of cache control headers, can be resolved.

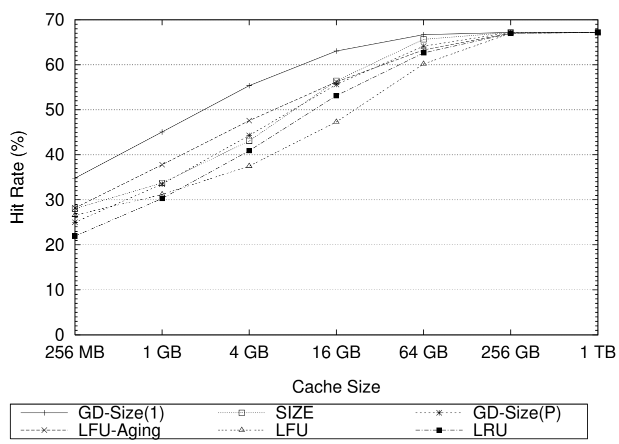
6. SIMULATION RESULTS

6.1 Performance of Existing Policies

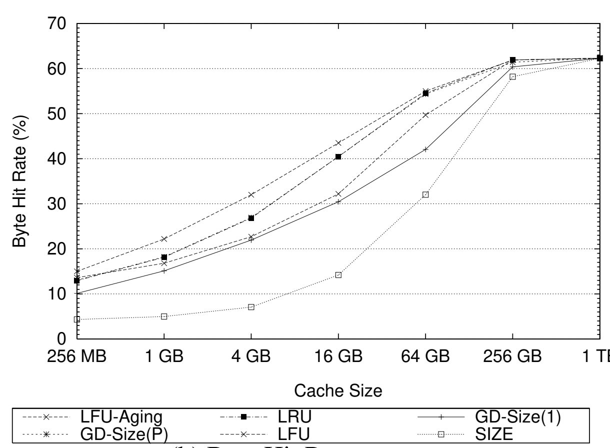
This section provides the simulation results of the proxy cache replacement policy study. Figure 1 consists of two graphs, with the graph on the left indicating the achieved hit rate for a cache of a particular size while the graph on the right shows the achieved byte hit rate for a similarly configured cache. We have also sorted the legend in each graph by the performance of the policies with a cache size of 256 MB. For example, in Figure 1(a), the GD-Size(1) policy achieved the highest hit rate. LRU obtained the lowest hit rate of the policies that we examined.

The results indicate that the hit rate achieved with an infinite-sized cache is 67% (obtained by all policies with a cache size of 1 TB). The remaining 33% of requests are for uncacheable objects (e.g., output from dynamic or cgi objects), the initial requests for objects or the updates for objects which have been modified and cannot be served from the cache. Figure 1(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 GD-Size(1) policy achieved a hit rate of 35% which is 52% of the hit rate that is obtained with an infinite-sized cache. This rate was achieved while allowing for only 0.06% of the entire object set size to be cached.

The results in Figure 1 show the performance of five of the policies examined by Cao and Irani [6]: GD-Size(1), GD-Size(P), SIZE, LRU and LFU. We have also included the LFU-Aging policy to show how the turnover characteristic impacts cache performance. Our results, in terms of the relative ordering of the policies is consistent with those of Cao and Irani. Figure 1(a) shows that the GD-Size(1) policy is superior to all of the other policies tested when considering only the hit rate metric, followed by LFU-Aging, GD-Size(P), SIZE, LRU and finally LFU. Neither LFU-Aging nor LRU consider the size of an object when making replacement decisions which is why these policies require significantly more cache space to achieve hit rates



(a) Hit Rate



(b) Byte Hit Rate

Figure 1. Comparison of existing Replacement Policies

similar to GD-Size(1). However, both of these policies are able to prevent cache pollution from occurring. Since LFU-Aging retains popular objects for longer periods of time and discriminates against one-timers it outperforms LRU. GD-Size(P) achieves lower hit rates than GD-Size(1) as GD-Size(P) retains larger files in order to improve the byte hit rate. Both the SIZE and LFU policies restrict the turnover of the cached object set, so the performance of these policies suffers due to cache pollution. To understand the impact of cache pollution we can compare the performance of LFU with LFU-Aging. Figure 1(a) clearly shows that LFU-Aging, which uses an aging mechanism, consistently outperforms LFU, which does not. The performance of LFU is similar to LFU-Aging in only two situations. When cache sizes are very small (e.g., 256 MB), adding a single large file to the cache can result in the removal of many smaller objects from the cache, reducing the effects of cache pollution. When cache sizes are large (e.g., 256 GB), few replacement decisions are needed and thus cache pollution has less impact on performance.

Figure 1(b) shows the achieved byte hit rates for the replacement policies under study. Figure 1(b) reveals an achieved byte hit rate of 62% for an infinite-sized cache. 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 achieves a byte hit rate of 15% which is only one quarter of the byte hit rate obtained with an infinite-sized cache.

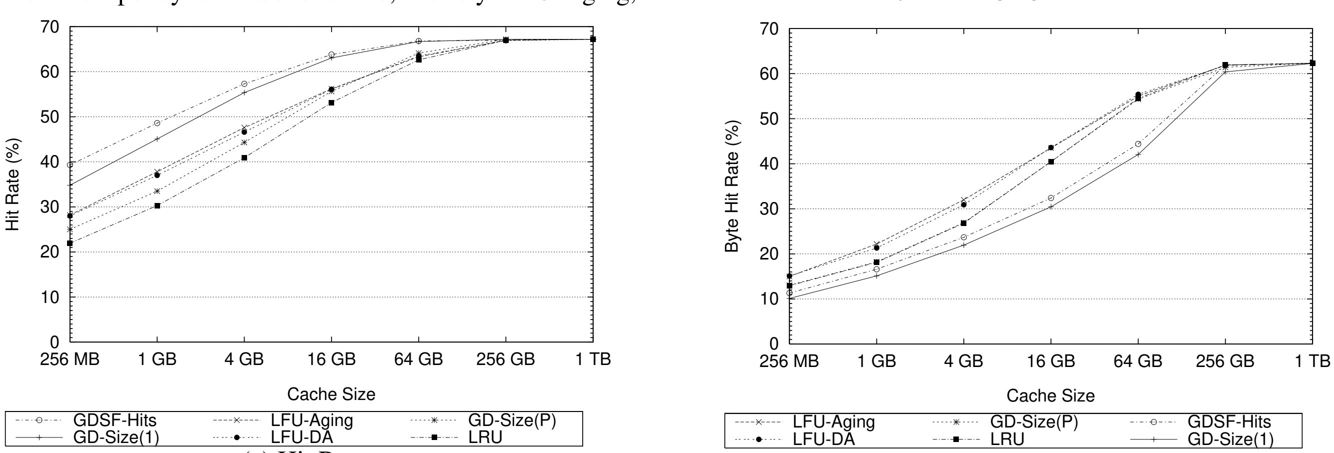
The relative ordering of the policies in Figure 1(b) is consistent with the results of Cao and Irani [6] for the five common policies. GD-Size(P) and LRU, which have almost identical performance, achieved the highest byte hit rates of the common policies, followed by LFU, GD-Size(1), and finally SIZE. LFU performs poorly due to cache pollution. GD-Size(1) achieves lower byte hit rates than GD-Size(P) because GD-Size(1) discriminates against large objects. The SIZE policy performance is extremely poor because of discrimination against large objects and cache pollution. The sixth policy that we examine, namely LFU-Aging, obtains the highest byte hit rates. This policy works well for this metric as it does not discriminate against the large objects which are responsible for a significant amount of the network traffic. LFU-Aging also retains popular objects (both small and large) longer than recency-based policies, while the aging mechanism prevents cache pollution, the reason why LFU-Aging achieves higher byte hit rates than LFU.

6.2 Performance of New Policies

In Figure 2 we compare the performance of the GDSF-Hits and LFU-DA policies with the GD-Size(1), GD-Size(P), LFU-Aging and LRU policies. We removed the SIZE and LFU policies in order to keep the graphs comprehensible. We chose to remove these two policies since they both suffered from cache pollution, and were not the best policy for either metric.

Figure 2(a) shows that the GDSF-Hits policy achieves higher hit rates than the GD-Size(1) policy which had the highest hit rate of all the policies in Figure 1(a). For small cache sizes GDSF-Hits requires only half of the cache space to achieve the same hit rates as the GD-Size(1) policy, and almost 16 times less space than the LRU policy. With a 256 MB cache (0.06% of the object set size) the GDSF-Hits policy achieves a hit rate of more than 39% which is 58% of the hit rate obtained with an infinite-sized cache. This result is significant as it means that a proxy cache could potentially satisfy many cache hits directly from its memory cache without having to read from its disk cache. This could have a significant, positive impact on the performance of the proxy cache. Figure 2(b) indicates that the GDSF-Hits policy also achieves a higher byte hit rate than does GD-Size(1). However, the byte hit rate of GDSF-Hits is still significantly lower than several of the other policies.

The results in Figure 2 indicate that the LFU-DA policy achieves hit rates and byte hit rates that are quite close to the LFU-Aging policy, even though the LFU-Aging policy was parameterized for the workload used to drive the simulations while the LFU-DA policy was not. Furthermore, the dynamic aging mechanism of the LFU-DA



(a) Hit Rate

(b) Byte Hit Rate

Figure 2. Comparison of Proposed Policies to Existing Replacement Policies

policy is more efficient computationally than that of LFU-Aging. With the exception of LFU-Aging, LFU-DA achieved higher byte hit rates than all of the other tested policies.

7. VIRTUAL CACHES

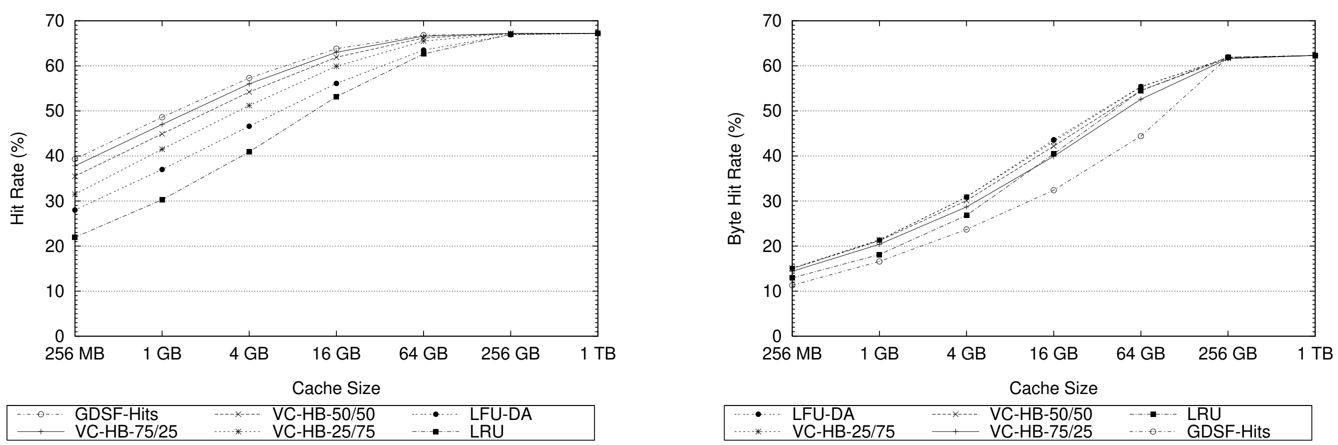
In Section 5.1 we introduced two new replacement policies, neither of which requires parameterization. In Section 6.2 we showed how GDSF-Hits achieved high hit rates while LFU-DA achieved hit byte hit rates. A drawback of having two policies is that a decision must be made on which metric is most important and therefore which replacement policy should be used. In some situations the choice may be obvious; in others both high hit rates and high byte hit rates may be important.

We developed an approach that can focus on both of these metrics simultaneously. This approach logically partitions the cache into N virtual caches. Each virtual cache (VC) is then managed with its own replacement policy. Initially all objects are added to VC_0. Replacements from VC_i are moved to VC_{i+1}. Replacements from VC_{n-1} are evicted from the cache. All objects that are reaccessed while in the cache (i.e., cache hits) are reinserted in VC_0. This allows in-demand objects to stay in the cache for a longer period of time. For example, to achieve high hit rates and high byte hit rates simultaneously, two VCs are used. One VC focuses on obtaining high hit rates using a replacement policy such as GDSF-Hits. The other VC aims to achieve high byte hit rates by utilizing a replacement policy such as LFU-DA.

We simulated this cache management policy to evaluate its effectiveness when utilizing two VCs. Our first experiment used the GDSF-Hits policy to manage VC_0 (we refer to this as the Hits VC) while VC_1 (the Bytes VC) employed the LFU-DA policy. We examined several different allocations of the cache space: 75% for the Hits VC, 25% for the Bytes VC (denoted VC-HB-75/25); 50% for each VC (VC-HB-50/50); and 25% for the Hits VC, 75% for the Bytes VC (VC-HB-25/75). Figure 3 shows the performance of the VC management policy for the different configurations we have just described. Figure 3 also includes the results for the GDSF-Hits, LFU-DA and LRU policies when each manages 100% of the total cache space.

The results in Figure 3 indicate that the use of the VC management approach does in fact improve the performance of the cache across multiple metrics. Both the hit rate and byte hit rate of the VC management approach appear to be bounded by those of the policies used to manage each of the individual VCs (we do not have a theoretical proof that this is always true). As the percentage of space dedicated to the Hits VC decreases the hit rate also decreases although it remains higher than the policy used to manage the Bytes partition. At the same time the byte hit rate increases, nearing that achieved by the policy used to manage the Bytes partition. These improvements can be explained from the characteristics of the proxy workload. Due to the non-uniform popularity distribution a cache is able to achieve relatively high hit rates with a very small amount of storage space. However, as the cache gets larger, the rate of improvement in hit rate declines as it becomes increasingly difficult to identify objects that will continue to improve the hit rate (the diminishing returns effect). A similar argument can be made regarding the objects that affect the byte hit rate. By partitioning the cache we are retaining most of the benefit of a cache that is dedicated to a single metric while making more effective use of the remaining space relative to a second metric.

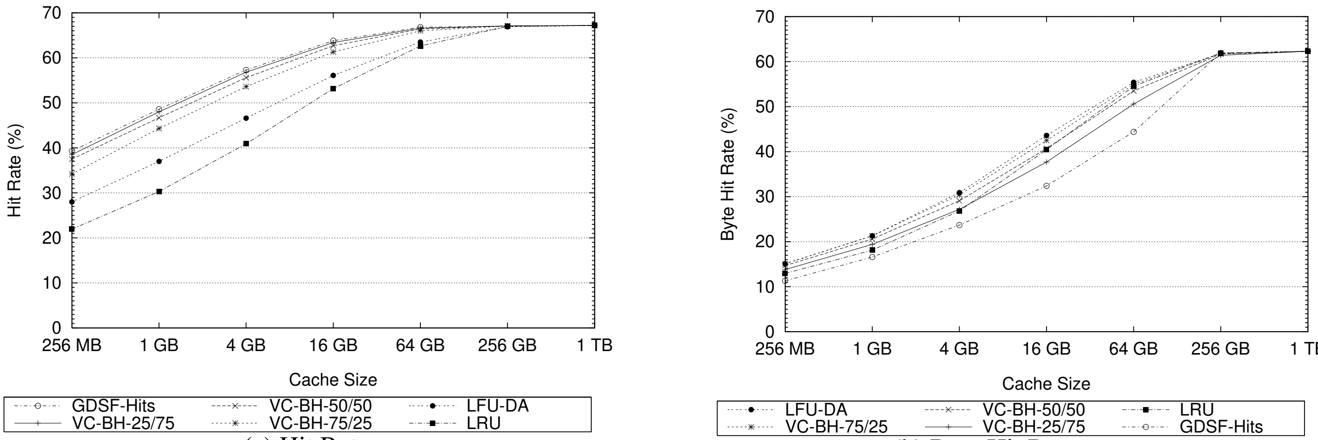
To see how the VC management policy performs relative to the other replacement policies not included in Figure 3 we rely on comparisons with the GDSF-Hits and LFU-DA policies. Since the hit rates for the tested VC management policies are bounded below by the LFU-DA policy, the VC approach achieves higher hit rates than all of the other tested replacement policies except for GD-Size(1). However, the byte hit rate of the tested VC policies are bounded below by the performance of the GDSF-Hits policy, which achieves a higher byte hit rate than does GD-Size(1). Thus GD-Size(1) does not achieve higher performance for both hit rate and byte hit rate compared to any of the VC policies that we examined. Similar analogies can be made for the other



(a) Hit Rate

(b) Byte Hit Rate

Figure 3. Analysis of Virtual Cache Performance; VC_0 using GDSF-Hits, VC_1 using LFU-DA



(a) Hit Rate (b) Byte Hit Rate

Figure 4. Analysis of Virtual Cache Management; VC_0 using LFU-DA, VC_1 using GDSF-Hits

policies.

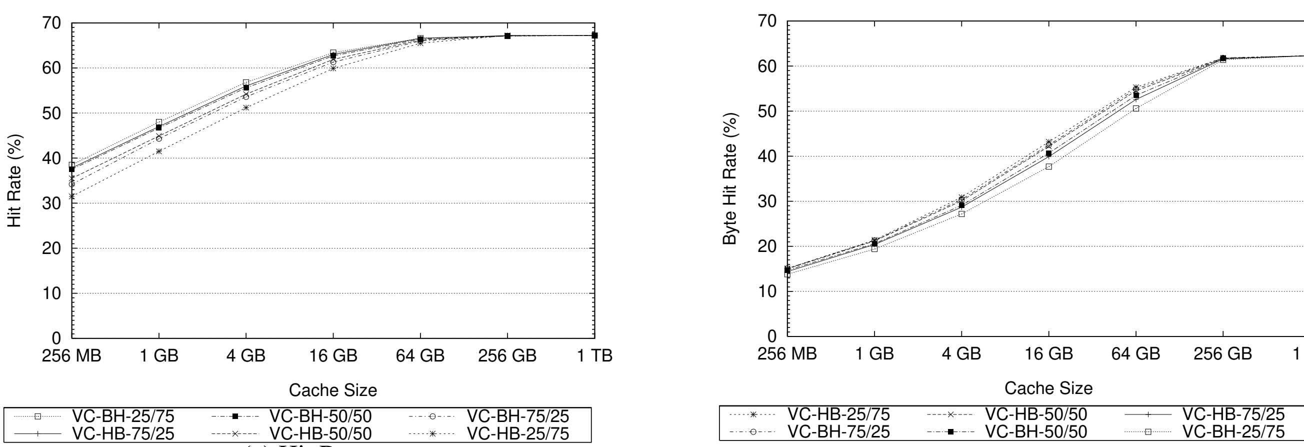
Our next experiment analyzed the importance of the ordering of the VCs. Figure 4 shows the results when the Bytes VC is the first partition (VC_0) and the Hits VC is the second (VC_1). The results in Figure 4 show that once again the performance of the VC management approach is bounded by the performance of the replacement policies selected to manage each individual VC.

Figure 5 compares the performance between the Hits-Bytes ordering of VCs (Figure 3) with the performance of the Bytes-Hits ordering of VCs (Figure 4). The results in Figure 5 indicate that the ordering of the VCs does have some effect on the overall performance of the cache. More specifically, VC_1 appears to have more impact on the overall performance than does VC_0. This occurs because the objects in VC_1 are moved back into VC_0, in order that these objects can remain in the cache for a longer period of time. This action influences the replacement decisions of VC_0, as it must remove some of its preferred objects in order to add the objects passed from VC_1.

8. Contributions and Future Work

This paper has presented a performance study of a Web proxy cache based on the most significant proxy workload characterization to date. Trace-driven simulations were used to evaluate the performance of different cache replacement policies for proxy caches. We then used the workload results to develop two new replacement policies that built upon the strengths of the existing policies while avoiding the weaknesses. Our results indicate that size-based policies achieve higher hit rates than other policies. Further improvements in hit rate result by giving preference to frequency of reference. We also found that frequency-based policies are more effective at improving the byte hit rate of a proxy cache. With either metric it is important to consider turnover in the active set of objects. Furthermore, we have developed virtual caches as an approach to provide optimal cache performance for multiple metrics simultaneously.

There are many open issues regarding Web proxy performance and caching. Future work in this area may include designing new replacement policies, particularly ones that can achieve higher byte hit rates (through the use



(a) Hit Rate (b) Byte Hit Rate

Figure 5. Analysis of Virtual Cache Management; Effects of VC Order on Performance

of future knowledge we have determined that significantly higher byte hit rates than those achieved by the LFU-DA policy are possible). We have implemented the new replacement policies in a proxy cache and are examining the effects that a replacement policy has on system performance [9]. More investigation into the VC approach is needed, including a theoretical description of its operation. We intend to continue our investigation of Web caching and its role in providing more predictable end-to-end performance. Other issues include examining the relationship between hit rates and latency reduction for end users, implementing a more efficient consistency mechanism [14][16], 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 while providing users and content providers with the functionality they desire.

9. ACKNOWLEDGEMENTS

The authors would like to thank Mike Rodriquez of Hewlett-Packard Laboratories and all the people in HP's Telecommunication Platforms Division (TPD) who supplied us with the access logs used in this research. The authors are grateful to Greg Oster of the University of Saskatchewan for his assistance with the development of the simulator, and to Godfrey Tan of UC-Berkeley for his work on validating the proposed replacement policies. We would also like to thank the anonymous reviewers for their many constructive comments.

10. 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, "Performance Evaluation of Web Proxy Cache Replacement Policies", to appear in *Performance Evaluation Journal*, 1999.

[3] M. Arlitt, R. Friedrich, and T. Jin, "Workload Characterization of a Web Proxy in a Cable Modem Environment", to appear in *ACM SIGMETRICS Performance Evaluation Review,* August 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. 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.

[6] 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.

[7] L. Cherkasova, "Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency Caching Policy", Technical Report HPL-98-69R1, Hewlett-Packard Laboratories, November 1998.

[8] 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.

[9] J. Dilley, M. Arlitt and S. Perret, "Enhancement and Validation of the Squid Cache Replacement Policy", submitted for publication, January 1999.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] M. Kurcewicz, W. Sylwestrzak and A. Wierzbicki, "A Filtering Algorithm for Proxy Caches", *3rd International Web Cache Workshop*, http://wwwcache.ja.net/events/workshop/.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] K. Thompson, G. Miller and R. Wilder, "Wide-Area Internet Patterns and Characteristics", *IEEE Network*, Vol. 11, No. 6, November/December 1997.

[22] D. Wessels, "Squid Internet Object Cache" http://squid.nlanr.net.

[23] 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.

[24] 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.