Trace Driven Simulation of GDSF# and Existing Caching Algorithms for Web Proxy Servers

J B Patil
Department of Computer Engineering,
R. C. Patel Institute of Technology, Shirpur. (M.S.), INDIA

B. V. Pawar
Department of Computer Science
North Maharashtra University, Jalgaon. (M.S.), INDIA

Abstract: - Web proxy caching is used to improve the performance of the Web infrastructure. It aims to reduce network traffic, server load, and user perceived retrieval delays. The heart of a caching system is its page replacement policy, which needs to make good replacement decisions when its cache is full and a new document needs to be stored. The latest and most popular replacement policies like GDSF use the file size, access frequency, and age in the decision process. The effectiveness of any replacement policy can be evaluated using two metrics: hit ratio (HR) and byte hit ratio (BHR). There is always a trade-off between HR and BHR [1]. In this paper, using three different proxy server logs, we use trace driven analysis to evaluate the effects of different replacement policies on the performance of a Web server. We propose a modification of GDSF policy, GDSF#, which allows augmenting or weakening the impact of size or frequency or both on HR and BHR. Our simulation results show that our proposed replacement policy GDSF# gives close to perfect performance in both the important metrics: HR and BHR.

Key-words: - Web caching, Replacement policy, Hit ratio, Byte hit ratio, Trace-driven simulation

1 Introduction

The enormous popularity of the World Wide Web has caused a tremendous increase in network traffic due to http requests. This has given rise to problems like user-perceived latency, Web server overload, and backbone link congestion. Web caching is one of the ways to alleviate these problems [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. Web caches can be deployed throughout the Internet, from browser caches, through proxy caches and backbone caches, through reverse proxy caches, to the Web server caches.

In our work, we use trace-driven simulation for evaluating the performance of different caching policies for Web proxy servers.

Cao and Irani have surveyed ten different policies and proposed a new algorithm, Greedy-Dual-Size (GDS) in [5]. The GDS algorithm uses document size, cost, and age in the replacement decision, and shows better performance compared to previous caching algorithms. In [4] and [12], frequency was incorporated in GDS, resulting in Greedy-Dual-Frequency-Size (GDSF) and Greedy-Dual-Frequency (GDF). While GDSF is attributed to having best hit ratio (HR), it having a modest byte hit ratio (BHR). Conversely, GDF yields a best HR at the cost of worst BHR [12].

In this paper, we propose a new algorithm, called Greedy-Dual-Frequency-Size# (GDSF#), which allows augmenting or weakening the impact of size or frequency or both on HR and BHR. We compare GDSF# with common algorithms like GDS(1), GDS(P), GDSF(1), and GDSF(P). Our simulation study shows that GDSF# gives close to perfect performance in both the important metrics: HR and BHR.

The remainder of this paper is organized as follows. Section 2 introduces GDSF#, a new algorithm for Web cache replacement. Section 3 describes the simulation model for the experiment. Section 4 describes the experimental design of our simulation while Section 5 presents the simulation results. We present our conclusions in Section 6.

2 GDSF# Algorithm (Our Proposed Algorithm)

In GDSF, the key value of document *i* is computed as follows[4] [12]:

$$H_i = L + (f_i \times c_i)/s_i$$

The *Inflation Factor L* is updated for every evicted document i to the priority of this document *i*. In this way, *L* increases monotonically. However, the rate of increase is very slow. If a faster mechanism for increasing *L* is designed, it will lead to a replacement algorithm with features closure to LRU. We can apply similar reasoning to s_i and f_i. If we augment the frequency by using f_i^2, f_i^3,…, etc. instead of f_i then the impact of frequency is more pronounced than that of size. Similarly, if we use $s_i^{0.1}$, $s_i^{0.2}$,…, etc. or use $\log(s_i)$ instead of file size s_i , then the impact of size is less than that of frequency [4].
Extending this logic further, we propose an extension to the GDSF, called GDSF#, where the key value of document is computed as

$$H_i = L + (c_i \times f_i^{\lambda})/s_i^{\delta}$$

where λ and δ are rational numbers. If we set λ or δ above 1, it augments the role of the corresponding parameter. Conversely, if we set λ or δ below 1, it weakens the role of the corresponding parameter.

Therefore, we present the GDSF# algorithm as shown below:

```
begin
Initialize $L = 0$
Process each request document in turn:
let current requested document be $i$
if $i$ is already in cache
        $H_i = L + (c_i \times f_i^{\lambda})/s_i^{\delta}$
else
while there is not enough room in cache for $p$
begin
let $L = min(H_i)$, for all $i$ in cache
evict $i$ such that $H_i = L$
end
load $i$ into cache
        $H_i = L + (c_i \times f_i^{\lambda})/s_i^{\delta}$
end
```

3 Simulation Model for the Experiment

In case of proxy servers, all requests are assumed to be directed to the proxy server. When the proxy receives a request from a client, it checks its cache to see if it has a copy of the requested object. If there is a copy of the requested object in its cache, the object is returned to the client signifying a *cache hit*, otherwise the proxy records a *cache* miss. The original Web server is contacted and on getting the object, stores the copy in its cache for future use, and returns a copy to the requesting user. If the cache is already full when a document needs to be stored, then a replacement policy is invoked to decide which document (or documents) is to be removed.

3.1 Workload Traces

For Web proxy servers, we have used: Boston University Computer Science Department client traces collected in 1995; BU272 and BU-B19 [13] and one trace collected in 1998; BU98 [14] [15].

4. Experimental Design

This section describes the design of the performance study of cache replacement policies. The discussion begins with the factors and levels used for the simulation. Next, we present the performance metrics used to evaluate the performance of each replacement policy used in the study. Lastly, we discuss other design issues regarding the simulation study.

4.1 Factors and Levels

There are two main factors used in the in the trace-driven simulation experiments: cache size and cache replacement policy. This section describes each of these factors and the associated levels.

4.1.1 Cache Size

The first factor in this study is the size of the cache. For the proxy logs, we have used ten levels from 1 MB to 1024 MB except in case of BU-B19 trace, we have a upper bound of 4096 MB. The upper bounds of cache size are chosen to represent an infinite cache size for the respective traces. An *infinite cache* is one that is so large that no file in the given trace, once brought into the cache, need ever be evicted. It allows us to determine the maximum achievable cache hit ratio and byte hit ratio, and to determine the performance of a smaller cache size to be compared to that of an infinite cache.

4.1.2 Replacement Policy

In our research, we examine the following previously proposed replacement policies: GDS(1), GDS(P), GDSF(1), and GDSF(P). Our proposed policy GDSF# is also examined and evaluated against these policies.

Greedy-Dual-Size (GDS): GDS [5] maintains for each object a characteristic value H_i. A request for object *i* (new request or hit) requires a recalculation of H_i. H_i is calculated as

$$H_i = L + c_i / s_i$$

L is a running aging factor, which is initialized to zero, c_i is the cost to fetch object i from its origin server, and s_i is the size of object i. GDS chooses the object with the smallest Hi-value. The value of this object is assigned to L. if cost is set to 1, it becomes GDS(1), and when cost is set to p = 2 + size/536, it becomes GDS(P).

Greedy-Dual-Size-Frequency (GDSF): GDSF [4] [12] calculates H_i as

$$H_i = L + (f_i \times c_i)/s_i$$

It takes into account frequency of reference in addition to size. Similar to GDS, we have GDSF(1) and GDSF(P).

4.2 Performance Metrics

The performance metrics used to evaluate the various replacement policies used in this simulation are *Hit Rate* and *Byte Hit Rate*.

Hit Rate (HR) Hit rate (HR) is the ratio of the number of requests met in the cache to the total number of requests.

Byte Hit Rate (BHR) Byte hit rate (BHR) is concerned with how many bytes are saved. This is the ratio of the number of bytes satisfied from the cache to the total bytes requested.

5 Simulation Results

This section presents the simulation results for comparison of different file caching strategies.

Section 5.1 gives the simulation results for the GDSF# algorithm. Section 5.2 shows the results for proxy servers.

We show the simulation results of GDS(1), GDS(P), GDSF(1), GDSF(P), GDSF#(1), and GDSF#(P) for the proxy server traces for hit rate and byte hit rate. The graph for Infinite indicates the performance for the Infinite cache size.

5.1 Simulation Results for GDSF# Algorithm

In this section, we experiment with the various values of λ and δ in the equation for computing key value,

$$H_i = L + (c_i \times f_i^{\lambda})/s_i^{\delta}$$

to augment or weaken the impact of frequency and size in GDSF#.

5.1.1 Effect of Augmenting Frequency in GDSF#

If we add frequency in the GDS to make it GDSF, it improves BHR considerably and HR slightly. To check whether we can further improve the performance, we set λ = 2, 5, 10 with δ = 1 in the equation for H_i. Figures 1a and 1b show a comparison of GDSF(1) and GDSF#(1) with λ = 2, 5, 10 with δ = 1 for the BU272 Web proxy trace.

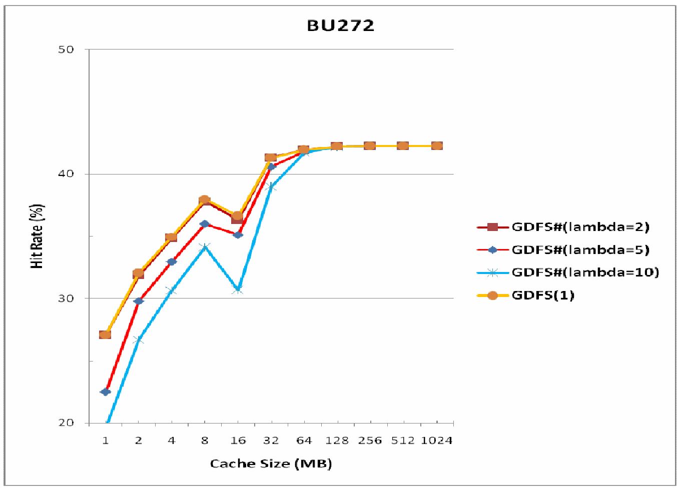

Fig. 1a: HR for GDSF# algorithm for BU272 trace (λ=2, 5, 10)

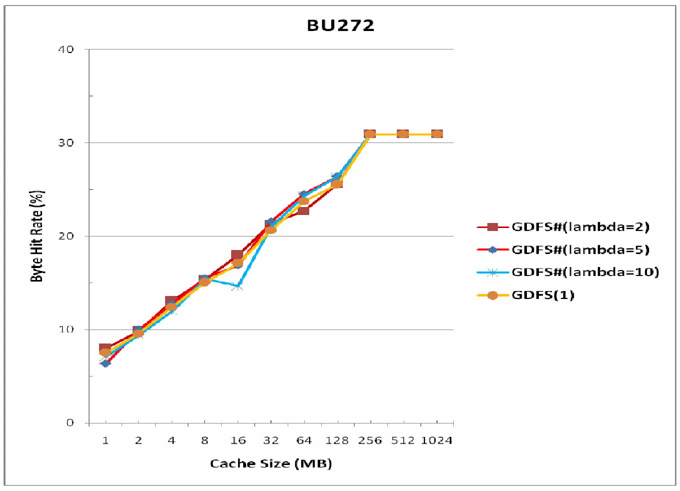

Fig. 1b: BHR for GDSF# algorithm for BU272 trace (λ=2, 5, 10)

The results indicate that augmenting frequency in GDSF# improves BHR in all the three traces but the improvement comes at the cost of HR. Again, we find that with λ = 2, we get the best results for BHR. We get the similar results for the remaining two traces.

5.1.2 Effect of De-Augmenting Size in GDSF#

We have seen that emphasizing frequency in GDSF# results in improved BHR. Now let us check the effect of de-augmenting or weakening the size. For this, we set δ = 0.3, 0.6, 0.9 with λ = 1 in the equation for H_i. Figures 2a and 2b show a comparison of GDSF(1) and GDSF#(1) with δ = 0.3, 0.6, 0.9 with λ = 1 for the BU272 Web proxy trace.

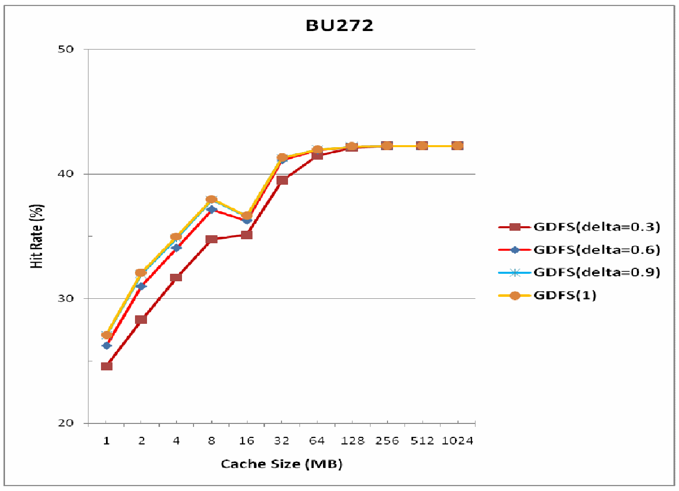

Fig. 2a: HR for GDSF# algorithm for BU272 trace (δ=0.3, 0.6, 0.9)

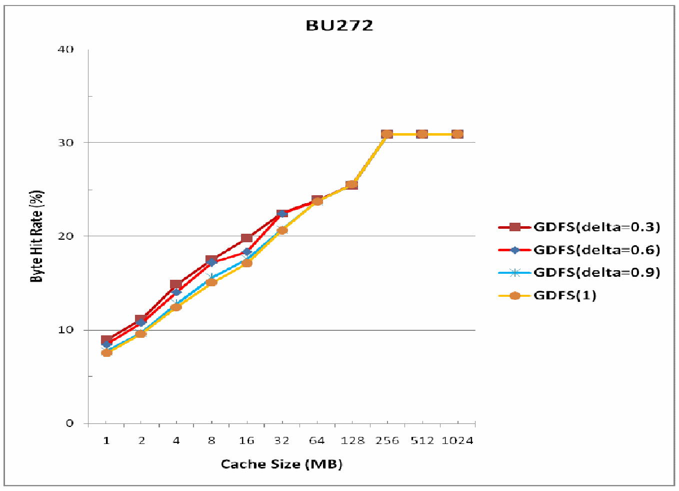

Fig. 2b: BHR for GDSF# algorithm for BU272 trace (δ=0.3, 0.6, 0.9)

The results are on expected lines. The effect of decreased impact of file size improves BHR across all the three traces. Again, it is at the cost of HR. Specifically, we get best BHR at δ = 0.3, and best HR at δ = 0.9.

5.1.3 Effect of Augmenting Frequency & De-Augmenting Size in GDSF#

We have seen that emphasizing frequency and de-emphasizing size in GDSF# results in improved BHR, at the cost of slight reduction in HR. Now the question is then whether we can achieve still better results by combination of both augmenting frequency and de-augmenting size. For this, we try different combinations of λ and δ. We find that we get best results for both HR and BHR for the combination λ = 2 and δ = 0.9 for all the six Web traces. This combination shows close to perfect performance for both the important metrics: HR and BHR.

This is important result because as noted earlier, there is always a trade-off between HR and BHR [2]. Replacement policies that try to improve HR do so at the cost of BHR, and vice versa [5]. Often, a high HR is preferable because it allows a greater number of requests to be serviced out of cache and thereby minimizing the average request latency as perceived by the user. However, it is also desirable to maximize BHR to minimize disk accesses or outward network traffic.

In the next sections, we use the best combination of λ = 2 and δ = 0.9 in the equation for H_i for GDSF# to compare the performance of GDSF# with GDS(1), GDS(P), GDSF(1), and GDSF(P). So, instead of denoting it as GDSF#(λ=2, δ=0.9), we will denote it as simply GDSF#. For the cost = 1, it will be denoted as GDSF#(1), and for packet cost (p = 2 + size/536), it will be denoted as GDSF#(P).

5.2 Simulation Results for Web Proxy Servers

In this section, we present and discuss simulation results for BU272, BU-B19, and BU98 Web proxy servers.

5.2.1 Simulation Results for BU272

Figures 3a and 3b give the comparison of GDSF# with other algorithms.

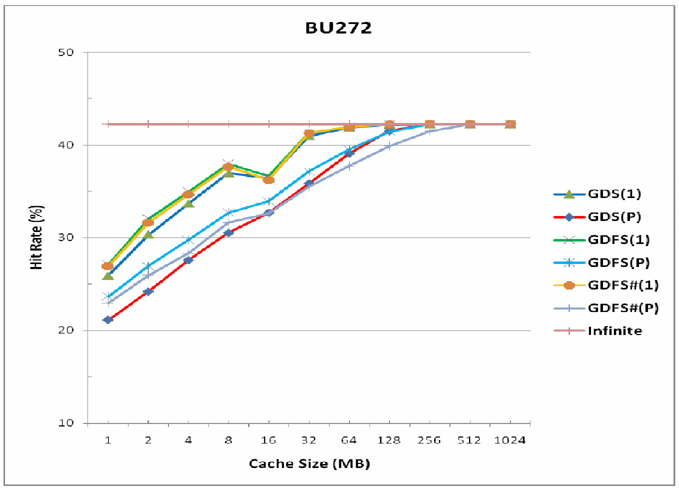

Fig. 3a: Hit rate of BU272 trace

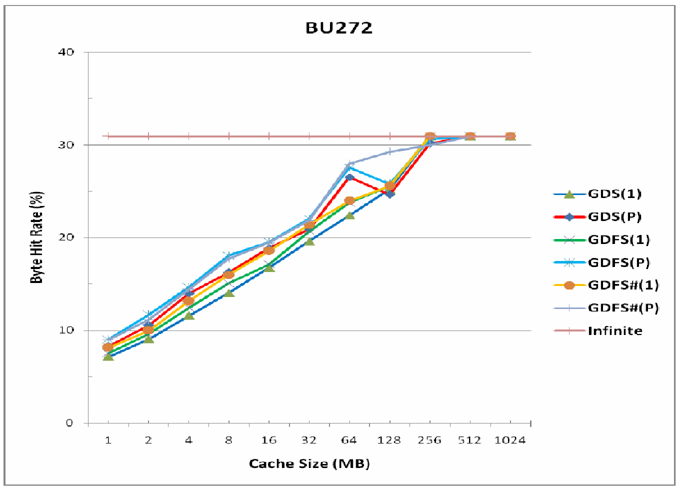

Fig. 3b: Byte Hit rate of BU272 trace

The results indicate that the HR achieved with an infinite sized cache is 42.23% while the BHR is 30.92% for the BU272 trace. Of the algorithms shown in Figure 3, GDSF(1) had the highest HR followed by GDSF#(1). GDS(1) and GDS(P) had a significantly lower HRs.
In case of BHRs, GDSF(P), and GDSF#(P) had the highest BHRs. However, GDSF# is optimized for both HR and BHR in case of BU272 trace.

5.2.2 Simulation Results for BU-B19

Figures 4a and 4b show the performance graphically.

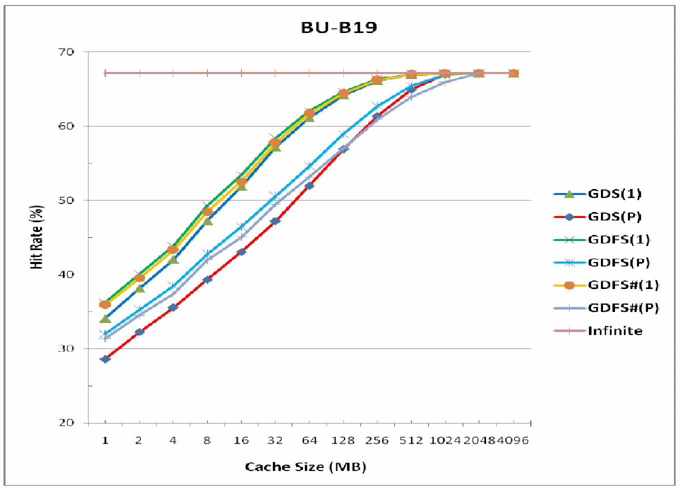

Fig. 4a: Hit rate of BU-B19 trace

The results indicate that the HR achieved with an infinite sized cache is 67.14% while the BHR is 48.21% for the BU-B19 trace. Of the algorithms shown in Figure 4, GDSF(1) had the highest HR followed by GDSF#(1). GDS(1) and GDS(P) had a lower HRs.

In case of BHRs, GDSF#(P) had the highest BHR followed by GDSF(P). GDSF# is optimized for both HR and BHR in case of BU-B19 trace.

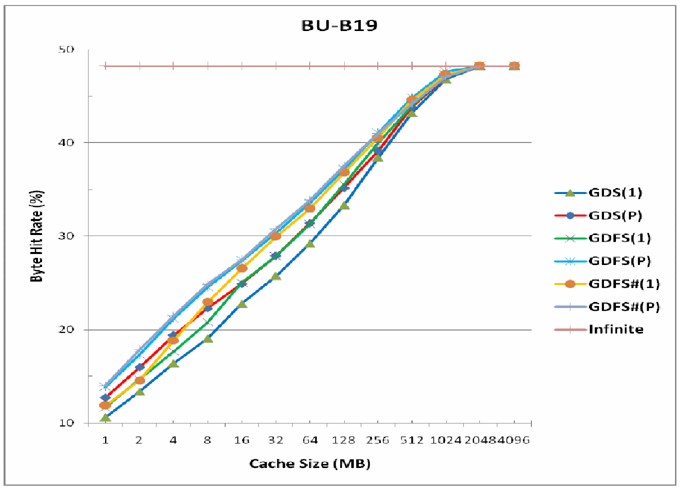

Fig. 4b: Byte Hit rate of BU-B19 trace

5.2.3 Simulation Results for BU98

Figures 5a and 5b show the performance graphically.

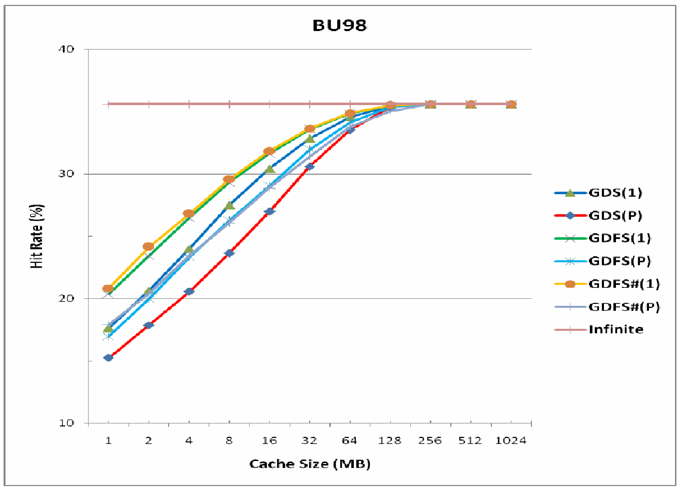

Fig. 5a: Hit rate of BU98 trace

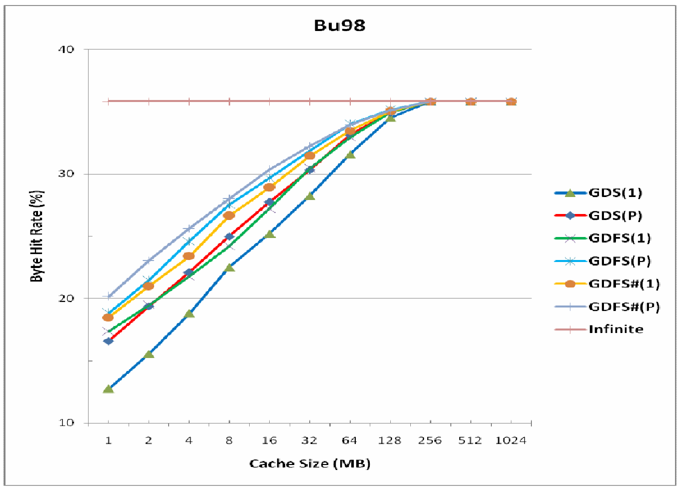

Fig. 5b: Byte Hit rate of BU98 trace

The results indicate that the HR achieved with an infinite sized cache is 35.61% while the BHR is 35.81% for the BU98 trace. Of the algorithms shown in Figure 5, GDSF#(1) had the highest HR followed by GDSF(1). GDS(1) and GDS(P) had a considerably lower HRs.
In case of BHRs, GDSF#(P) had the highest BHR followed by GDSF(P). However, like we noted earlier, GDSF# scores over all other algorithms in its optimized HR and BHR in case of BU-98 trace.

6. Conclusions

In this paper, we proposed a Web cache algorithm called GDSF#, which tries to maximize both HR and BHR. It incorporates the most important parameters of the Web traces: size, frequency of access, and age (using inflation value, *L*) in a simple way.

We have compared GDSF# with some popular cache replacement policies for Web proxy servers using a trace-driven simulation approach. We conducted several experiments using three proxy server traces. The replacement policies examined were GDS(1), GDS(P), GDSF(1), GDSF(P), GDSF#(1), and GDSF#(P). We used metrics like Hit Ratio (HR) and Byte Hit Ratio (BHR) to measure and compare performance of these algorithms. Our experimental results show that:

1. The HRs for Web proxy servers range from 35 to 68%, while BHRs range from 30 to 48% for the infinite caches. The values are consistent with the values reported in the literature [4] [5] [12] [16] [17] [18].

2. The results also indicate that it is more difficult to achieve high BHRs than high HRs. For example, in all the three traces, the maximum BHR is always less than maximum HR.

3. The results are consistent across all the three traces. GDSF# and GDSF show the best HR and BHR significantly outperforming other algorithms for these metrics.

4. Replacement policies emphasizing the document size yield better HR, but typically show poor BHR. The explanation is that in size-based policies, large files are always the potential candidates for the eviction, and the inflation factor is advanced very slowly, so that even if a large file is accessed on a regular basis, it is likely to be evicted repeatedly. GDSF and GDSF# use frequency as a parameter in its decision-making, so popular large files have better chance of staying in a cache. In addition, the inflation or ageing factor, *L* is now advanced faster. GDSF and GDSF# shows substantially improved BHR across all traces.

5. Similarly, replacement policies giving importance to frequency yield better BHR because they do not discriminate against the large files. These policies also retain popular objects (both small and large) longer than recency-based policies like LRU. However, normally these policies show poor HR because these policies do not take into account the file size which results in a higher file miss penalty.

6. We analyzed the performance of GDSF# policy, which allows augmenting or weakening the impact of size or frequency or both on HR and BHR. Our results show that our proposed replacement policy gives close to perfect performance in both the important metrics: HR and BHR.

References:

[1] M. Arlitt, R. Friedrich, and T. Jin, "Workload Characterization of Web Proxy Cache Replacement Policies", *ACM SIGMETRICS Performance Evaluation Review,* August 1999.
[2] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox, "Caching Proxies: Limitations and Potentials", *Proceedings of the 4th International World Wide Web Conference*, Boston, MA, December 1995, pp. 119-133.
[3] M. Arlitt and C. Williamson, "Trace Driven Simulation of Document Caching Strategies for Internet Web Servers", *Simulation Journal,* Vol. 68, No. 1, January 1977, pp. 23-33.
[4] L. Cherkasova, "Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency Caching Policy", *HP Technical Report HPL-98-69(R.1)*, November 1998.
[5] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms", *Proceedings of the USENIX Symposium on Internet Technology and Systems*, December 1997, pp. 193-206.
[6] S. Jin and A. Bestavros, "GreedyDual*: Web Caching Algorithms Exploiting the Two Sources of Temporal Locality in Web Request Streams", *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, 2000.
[7] S. Podlipnig and L. Boszormenyi, "A Survey of Web Cache Replacement Strategies", *ACM Computing Surveys*, Vol. 35, No. 4, December 2003, pp. 374-398.
[8] L. Rizzo, and L. Vicisano, "Replacement Policies for a Proxy Cache", *IEEE/ACM Transactions on Networking,* Vol. 8, No. 2, April 2000, pp. 158-170.

[9] A. Vakali, "LRU-based Algorithms for Web Cache Replacement", *International Conference on Electronic Commerce and Web Technologies*, Lecture Notes in Computer Science, Vol. 1875, Springer-Verlag, Berlin, Germany, 2000, pp. 409-418.
[10] R. P. 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, pp. 325-334.
[11] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox, "Removal Policies in Network Caches for World-Wide-Web Documents", *Proceedings of ACM SIGCOMM*, Stanford, CA, 1996, Revised March 1997, pp. 293-305.
[12] M. F., Arlitt, L. Cherkasova, J. Dilley, R. J. Friedrich, and T. Y Jin, "Evaluating Content Management Techniques for Web Proxy Caches", *ACM SIGMETRICS Performance Evaluation Review*, Vol. 27, No. 4, March 2000, pp. 3-11.
[13] C. R. Kunha, A. Bestavros, & M. E. Crovella, "Characteristics of WWW Client-based Traces", *Technical Report, BU-CS-95-010,* Computer Science Department, Boston University, 1995.
[14] A. Bradley, "BU Computer Science 1998 Proxy Trace", *Technical Report,* Computer Science Department, Boston University, 1999.
[15] P. Barford, A. Bestavros, A. Bradley, & M. Crovella, "Changes in Web Client Access Patterns Characteristics and Caching Implications", *World Wide Web*, Vol. 2, No. 1-2, 1999.
[16] M. Arlitt, R. Friedrich, & T. Jin, "Performance Evaluation of Web Proxy Cache in a Cable Modem Environment", *HP Technical Report*, *HPL-98-97(R.1),* Palo Alto, 1998.
[17] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, & E. A. Fox, "Caching Proxies: Limitations and Potentials", *Proceedings of the 4th International World Wide Web Conf.*, Boston, MA, 1995, pp. 119-133.
[18] H. Bahn, K. Koh, S. H. Noh, & S. L. Min, "Efficient Replacement of Nonuniform Objects in Web Caches", *IEEE Computer*, Vol. 35, No. 6, 2002, pp. 65-73.