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

[1]J B Patil, [2]B.V. Pawar

ABSTRACT

Web caching is used to improve the performance of the Internet Web servers. Document caching is used to reduce the time it takes Web server to respond to client requests by keeping and reusing Web objects that are likely to be used in the near future in the main memory of the Web server, and by reducing the volume of data transfer between Web server and secondary storage. 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 Web 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.

Keywords: 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 Internet Web servers. Our study uses Web server traces from three different sites on the Internet.

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-Size-Frequency (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].

[1]Department of Computer Engineering, R. C. Patel Institute of Technology, Shirpur. (M.S.), India. E-mail: jbpatil@hotmail.com

[2]Department of Computer Science, North Maharashtra University, Jalgaon. (M.S.), India. E-mail: bvpawar@hotmail.com

In this paper, we propose a new algorithm, called Greedy-Dual-Size -Frequency # (GDSF #), which allows augmenting or weakening the impact of size or frequency or both on HR and BHR. We compare GDSF# with GDS family algorithms like GDS(1), GDS(P), GDSF(1), 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 » and ′ are rational numbers. If we set » or ′ above 1, it augments the role of the corresponding parameter. Conversely, if we set » or ′ 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 Web Servers, a very simple Web server is assumed with a single-level file cache. When a request is received by the Web server, it looks for the requested file in its file cache. A *cache hit* occurs if the copy of the requested document is found in the file cache. If the document is not found in the file cache (a *cache miss*), the document must be retrieved from the local disk or from the secondary storage. On getting the file, it stores the copy in its file cache so that further requests to the same document can be serviced from the cache. If the cache is already full when a file needs to be stored, it triggers a replacement policy.

Our model also assumes file-level caching. Only complete documents are cached; when a file is added to the cache, the whole file is added, and when a file is removed from the cache, the entire file is removed.

For simplicity, our simulation model completely ignores the issues of *cache consistency* (i.e., making sure that the cache has the most up-to-date version of the document, compared to the master copy version at the original Web server, which may change at any time).

Lastly, caching can only work with static files, dynamic files that have become more and more popular within the past few years, cannot be cached.

3.1 Workload Traces

In this study, logs from three different Web servers are used: a Web server from an academic institute, Symbiosis Institute of Management Studies, Pune; a Web server from a manufacturing company, Thermax, Pune, and a Web server for an E-Shopping site in UK, www.wonderfulbuys.co.uk.

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.

Cache Size

The first factor in this study is the size of the cache. For the Web server logs, we have used seven levels from 1 MB to 64 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.

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 Web servers.

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

FIFO is chosen as a representative of strategies that do not exploit any particular access pattern characteristics, and hence its performance can be used to gauge the benefits of exploiting any such characteristics.

5.1 Simulation Results for GDSF# Algorithm

In this section, we experiment with the various values of » and ´ 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#.

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 $\lambda = 2, 5, 10$ with $\delta = 1$ in the equation for H_i. Figure 1 shows a comparison of GDSF(1) and GDSF#(1) with $\lambda = 2, 5, 10$ with $\delta = 1$ for the three Web server traces. 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 $\lambda = 2$, we get the best results for BHR.

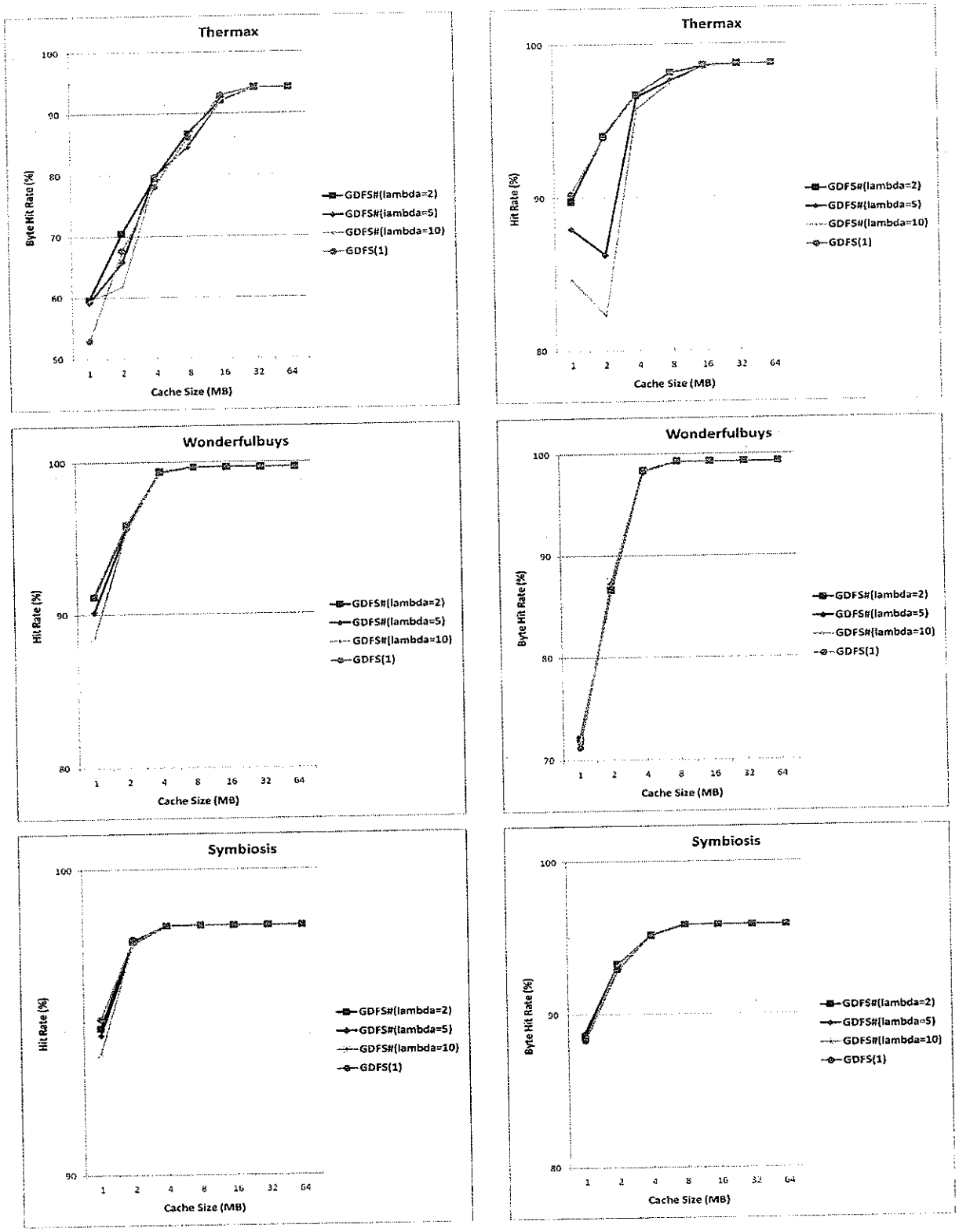


Figure 1: HR and BHR for GDSF# algorithm using Web server traces (λ=2, 5, 10)

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. Figure 2 shows a comparison of GDSF(1) and GDSF#(1) with δ = 0.3, 0.6, 0.9 with δ = 1 for the three Web server traces. The results are on expected lines. The effect of decreased impact of file size improves BHR across all the six 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.

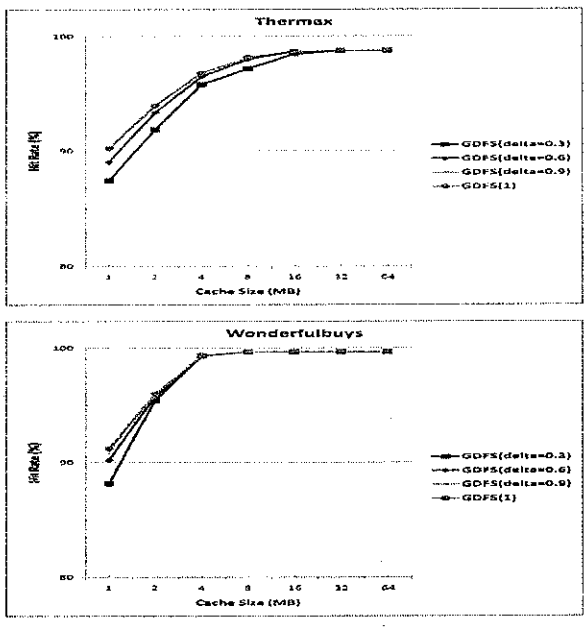
Effect of Augmenting Frequency & De-Augmenting Size in GDSF#

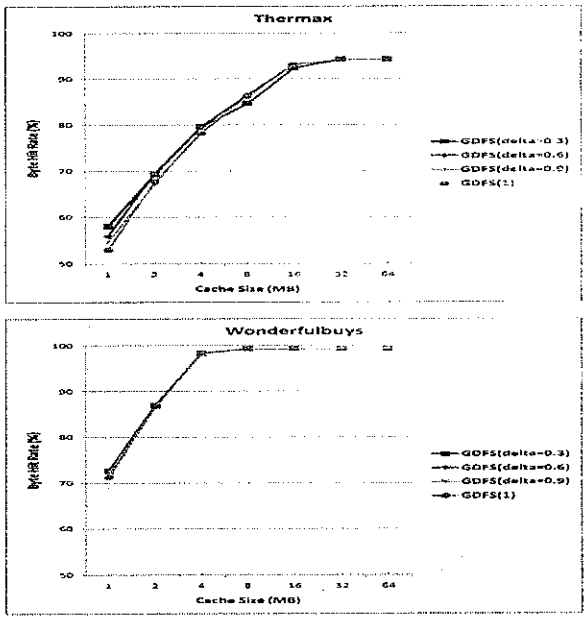
We have seen that emphasizing frequency and de-emphasizing size in GDSF# results in improved BHR, at the cost of slight reduction in HR. Now the question is then whether we can achieve still better results by combination of both augmenting frequency and de-augmenting size. For this, we try different 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#.





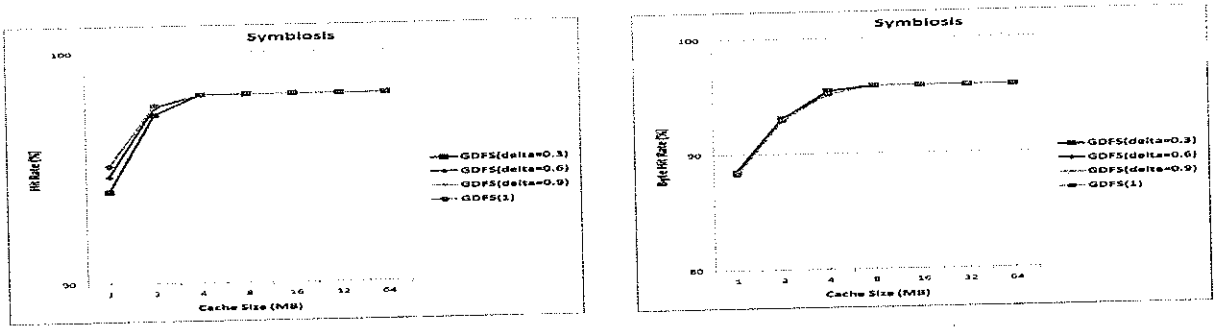


Figure 2: HR and BHR for GDSF# algorithm using Web server traces (δ=0.3, 0.6, 0.9)

5.2 Simulation Results for Web Servers

In this section, we present and discuss simulation results for Thermax, Wonderfulbuys, and Symbiosis Web servers.

Simulation Results for Thermax

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

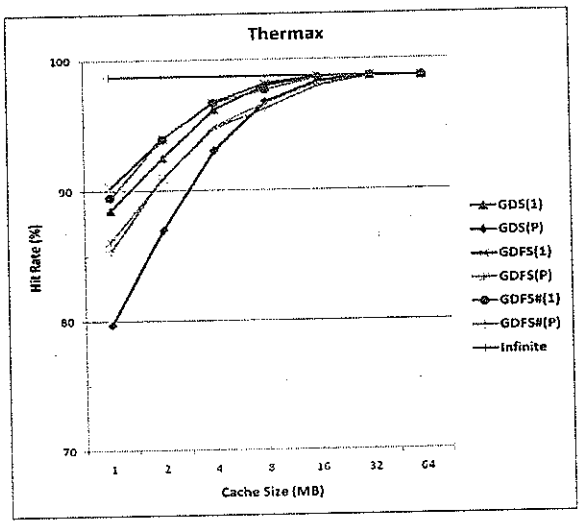


Figure 3a: Hit rate of Thermax trace

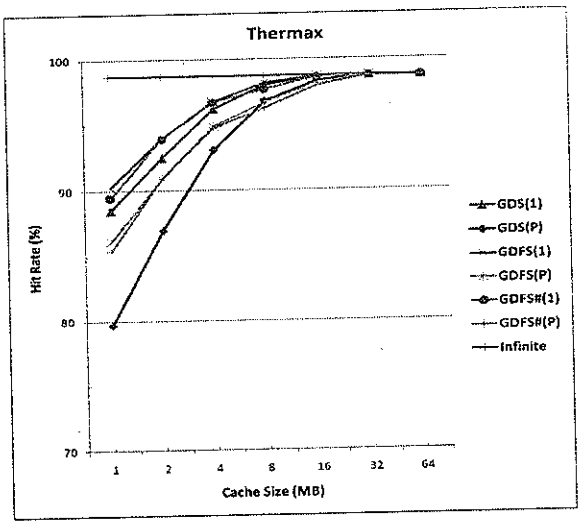


Figure 3b: Byte Hit rate of Thermax trace

The results indicate that the HR achieved with an infinite sized cache is 98.71% while the BHR is 94.19% for the Thermax trace. Of the algorithms shown in Figure 3, GDSF(1) and GDSF#(1) had the highest and almost similar HRs.

In case of BHRs, GDSF(P), GDSF#(1), and GDSF#(P) had the highest BHRs. However, GDSF(P) had a lower HR. GDSF# is thus optimized for both HR and BHR in case of Thermax trace.

Simulation Results for Wonderfulbuys

Figures 4a and 4b show the performance graphically.

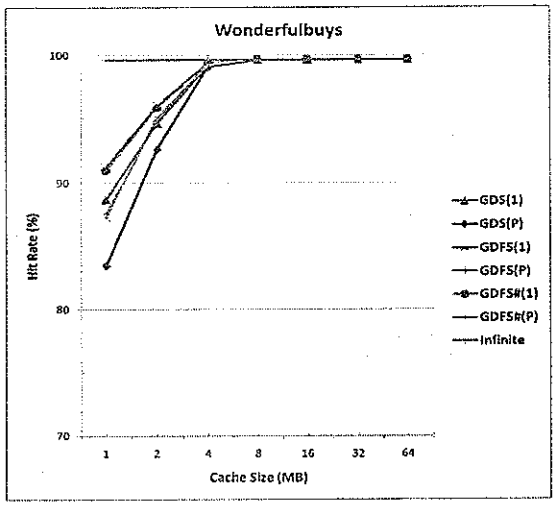


Figure 4a: Hit rate of Wonderfulbuys trace

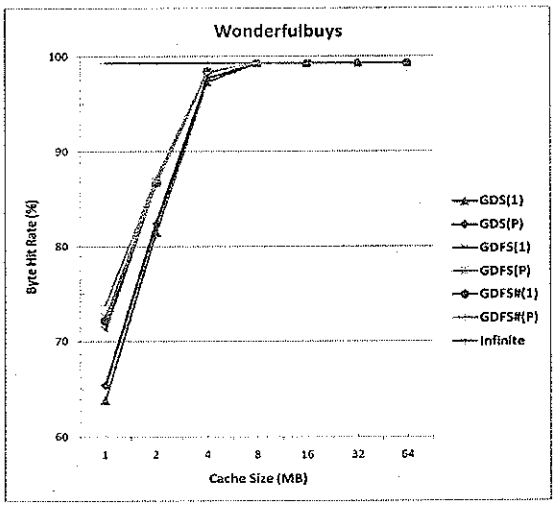


Figure 4b: Byte Hit rate of Wonderfulbuys trace

The results indicate that the HR achieved with an infinite sized cache is 99.66% while the BHR is 99.27% for the Wonderfulbuys trace. Of the algorithms shown in Figure 4, GDSF(1) had the highest HR followed by GDSF#(1). GDS(1) and GDS(P) also had a lower HRs.

In case of BHRs, GDSF#(P) had the highest BHR followed by GDSF(P), GDSF#(1), and GDSF(1). However, GDSF# scores over the others in better HR in case of Wonderfulbuys trace.

Simulation Results for Symbiosis

Figures 5a and 5b show the performance graphically.

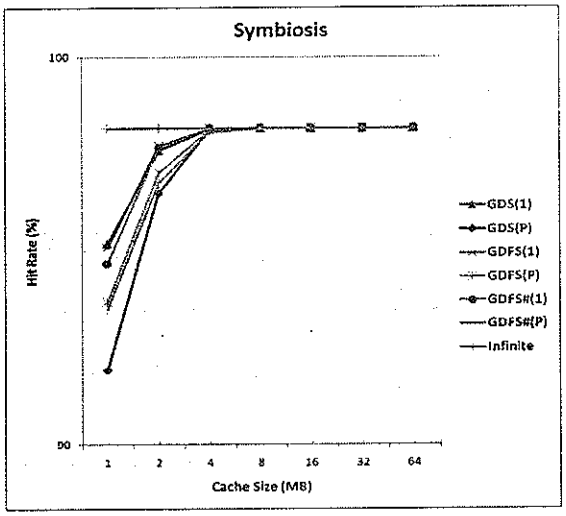


Figure 5a: Hit rate of Symbiosis trace

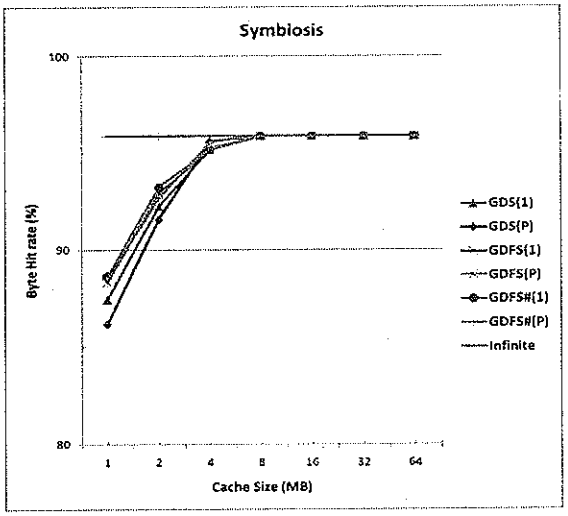


Figure 5b: Byte Hit rate of Symbiosis trace

The results indicate that the HR achieved with an infinite sized cache is 98.16% while the BHR is 95.87% for the Symbiosis trace. Of the algorithms shown in Figure 5, GDS(1) and GDSF(1) had the highest HR followed by GDSF#(1). GDS(1) had a lower HR.

In case of BHRs, GDSF#(1), and GDSF#(P) had the highest BHRs followed by GDSF(1) and GDSF(P). However,

because of comparatively better HR, GDSF# scores over most of the other algorithms in case of Symbiosis trace.

6. Conclusion

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 servers using a trace-driven simulation approach. We conducted several experiments using three Web 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:

- As pointed out by Williams et al. in [11], the observed HRs can range from 20% to as high as 98%, with majority ranging around 50%. The workload with a hit rate of 98% comes from a Web server cache, rather than proxy cache. Our results are consistent with this finding.
- 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.
- The results are consistent across all the three traces. GDSF# and GDSF show the best HR and BHR significantly outperforming the baseline algorithms like LRU, LFU for these metrics.
- 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# uses 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.
- 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.
- 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.

7. References

1. M. Arlitt, R. Friedrich, and T. Jin, *"Workload Characterization of Web Proxy Cache Replacement Policies"*, In *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"*, In Proceedings of the Fourth International World Wide Web Conference, Pages 119-133, Boston, MA, December 1995.

3. M. Arlitt and C. Williamson, *"Trace Driven Simulation of Document Caching Strategies for Internet Web Servers"*, Simulation Journal, Vol. 68, No. 1, PP 23-33, January 1977.
4. L. Cherkasova, *"Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency Caching Policy"*, In HP Technical Report *HPL-98-69(R.1)*, November 1998.
5. P. Cao and S. Irani, *"Cost-Aware WWW Proxy Caching Algorithms"*, In Proceedings of the USENIX Symposium on Internet Technology and Systems, PP 193-206, December 1997.
6. S. Jin and A. Bestavros, *"GreedyDual*: Web Caching Algorithms Exploiting the Two Sources of Temporal Locality in Web Request Streams"*, In Proceedings of the 5th International Web Caching and Content Delivery Workshop, 2000.
7. S. Podlipnig and L. Boszormenyi, *"A Survey of Web Cache Replacement Strategies"*, ACM Computing Surveys, Vol. 35, No.4, PP 374-398, December 2003.
8. L. Rizzo, and L. Vicisano, *"Replacement Policies for a Proxy Cache"*, IEEE/ACM Transactions on Networking, Vol. 8, No. 2, PP 158-170, April 2000.
9. A. Vakali, *"LRU-based Algorithms for Web Cache Replacement"*, In International Conference on Electronic Commerce and Web Technologies, Lecture Notes in Computer Science, Vol.1875, PP 409-418, Springer-Verlag, Berlin, Germany, 2000.
10. R. P. Wooster and M. Abrams., *"Proxy Caching that Estimates Page Load Delays"*, In Proceedings of the Sixth International World Wide Web Conference, PP 325-334, Santa Clara, CA, April 1997.
11. S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox, *"Removal Policies in Network Caches for World-Wide-Web Documents"*, In Proceedings of ACM SIGCOMM, PP 293-305, Stanford, CA, 1996, Revised March 1997.
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, PP 3-11, March 2000.

Author's Biography

Prof. J. B. Patil received his BE (Electronics) degree from the SGGS College of Engineering & Technology, Nanded in 1986; M.Tech.(Computer Science & Data Processing) from Indian Institute of Technology, Kharagpur in 1993; and has submitted his Ph.D. thesis to the North Maharashtra University, Jalgaon in Computer Science in January 2008. He is presently working as Principal and Professor in the Department of Computer Engineering, R. C. Patel Institute of Technology, Shirpur since 2001. His current research interest is in the area of Internet Web caching and prefetching. He has authored and co-authored over 20 papers in referred academic journals and national/international conference proceedings. He is a life member of Computer Society of India (CSI), Institute of Engineers (India), and Indian Society for Technical Education (ISTE).

Prof. B. V. Pawar received his B.E. degree from the V.J.T.I., Mumbai, in Production Engineering in 1986; M.Sc. degree from the Mumbai University, Mumbai, in Computer Science in 1988; Ph.D. degree from the North Maharashtra University, Jalgaon in Computer Science in 2000. He is presently Professor in the Department of Computer Science, North Maharashtra University,

Jalgaon where he has been involved in teaching and research since 1991. His current research interests are in the areas of natural language processing and information retrieval. He has authored and co-authored over 60 papers in referred academic journals and national/international conference proceedings. He is a life member of Computer Society of India (CSI), India and Linguistic Society of India (LSI).