IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 1, January 2011
ISSN (Online): 1694-0814
www.IJCSI.org

Improving Performance on WWW using Intelligent Predictive Caching for Web Proxy Servers

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

[1] Department of Computer Engineering, R. C. Patel Institute of Technology
Shirpur, Maharashtra 425405, India

[2] Department of Computer Science, North Maharashtra University
Jalgaon, Maharashtra 425001, 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 and 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 proxy server logs, we use trace driven analysis to evaluate the effects of different replacement policies on the performance of a Web proxy server. We propose a modification of GDSF# policy, IPGDSF#. Our simulation results show that our proposed replacement policy IPGDSF# performs better than several policies proposed in the literature in terms of hit rate as well as byte hit rate.

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 Web proxy servers.

One might argue that the ever decreasing prices of RAM and disks renders the optimization or fine tuning of cache replacement policies a “moot point”. Such a conclusion is ill guided for several reasons. First, recent studies have shown that Web cache hit ratio (HR) and byte hit ratio (BHR) grow in a *log-like* fashion as a function of cache size [5, 26, 27, 28]. Thus, a better algorithm that increases hit ratios by several percentage points would be equivalent to a several-fold increase in cache size. Second, the growth rate of Web content is much higher than the rate with which memory sizes for Web caches are likely to grow. The only way to bridge this widening gap is through efficient cache management. Finally, the benefit of even a slight improvement in cache performance may have an appreciable effect on network traffic, especially when such gains are compounded through a hierarchy of caches [6].

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 is having a modest byte hit ratio (BHR). Conversely, GDF yields a best HR at the cost of worst BHR [12].

We have proposed 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 [13, 14, 15, 16, 17].

In this paper, we propose an extension to our algorithm GDSF#, called Intelligent Predictive Greedy-Dual-

Frequency-Size#, (IPGDSF#). We compare IPGDSF# with algorithms like LRU, GDSF, and GDSF#. Our simulation study shows that IPGDSF# outperforms all other algorithms under consideration in terms of hit rate (HR) as well as byte hit rate (BHR).

The remainder of this paper is organized as follows. Section 2 introduces IPGDSF#, 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. IPGDSF# Algorithm

We extract *future frequency* from the Web proxy server logs. Then it is used to extend our GDSF# policy. Our idea is similar to the work of Bonchi et al. [18, 19] and Yang et al. [20]. While the Web caching algorithm in [18, 19] was designed to extend the LRU policy, Yang et al. [20] extended GDSF policy. We will be extending our policy GDSF# [13, 14, 15, 16, 17].

As pointed out early in caching research [21], the power of caching is in accurately predicting the usage of objects in the near future. In earlier works, estimates for future accesses were mostly built on measures such as access frequency, object size and cost. Such measures cannot be used to accurately predict for objects that are likely to be popular but have not yet been popular at any given instant in time. For example, as Web users traverse Web space, there are documents that will become popular soon due to Web document topology, although these documents are not yet accessed often in the current time instant [20]. Our approach is based on predictive Web caching model described by Yang et al. [20]. However, there are many noteworthy differences. Firstly, we use simple statistical techniques to find future frequency while Yang et al. use sequential association rules to predict the future Web access behavior. Secondly, for simplicity we do not try to identify user sessions. We assume that a popular document, which is used by one user, is likely to be used by many other users, which normally is the case for popular documents. We demonstrate the applicability of the method empirically through increased hit rates and byte hit rates.

Similar to the approach by Bonchi et al. [18, 19], our algorithm is an *intelligent* one as it can adapt to changes in usage patterns as reflected by future frequency. This is because the parameter *future frequency*, which is used in assigning weight (key value) to the document while storing in the cache, can be computed periodically in order to keep track of the recent past. This characteristic of adapting to the flow of requests in the historical data makes our policy intelligent. We call this innovative caching algorithm as *Intelligent Predictive GDSF#, (IPGDSF#).*

In GDSF#, the key value of document *i* is computed as follows [13, 14, 15, 16, 17]:

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

where λ and δ are rational numbers, *L* is the inflation factor, c_i is the estimated cost of the document *i*, f_i is the access frequency of the document *i*, and s_i is the document size.

We now consider how to find future frequency, ff_i for document *i* from the Web logs. We mine the preprocessed Web log files. We extract the unique documents from the logs. Then we arrange these documents in the temporal order. Now for each unique document, we extract the number of future occurrences of that document. We call this parameter as *future frequency, ff.*

With this parameter, we can now extend GDSF# by calculating H_i, the key value of document *i* as follows:

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

Here we add f_i and ff_i together, which implies that the key value of a document *i* is determined not only by its past occurrence frequency f_i, but also by its future frequency ff_i. By considering both the past occurrence frequency and future frequency, we can enhance the priority i.e. the key value of those objects that may not have been accessed frequently enough in the past, but will be in the near future according to the future frequency. The more likely it occurs in the future, the greater the key value will be. This will promote objects that are potentially popular objects in the near future even though they are not yet popular in the past. Thus, we look ahead in time in the request stream and adjust the replacement policy.

Finally, we make the policy intelligent by periodically updating future frequency when some condition becomes false, e.g. at fixed time intervals or when there is a degradation in the cache performance.

Now we present the IPGDSF# algorithm as shown in Fig.1:

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

Fig. 1 IPGDSF# algorithm.

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.

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

For Web proxy servers, we have used: Boston University Computer Science Department client traces collected in 1995; BU272 and BU-B19 [26] and one trace collected in 1998; BU98 [30] [31].

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.

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 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. Similar cache sizes are used by many researchers [9, 22, 23, 24]. The upper bounds represent the *Total Unique Mbytes* in the trace, which is essentially equivalent to having an infinite size cache [29]. 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 [23, 25]. 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

We show the simulation results of LRU, GDSF, GDSF#, and IPGDSF# for the Web proxy traces for hit rate, and byte hit rate. For the last three algorithms, we consider the cost function as one. In GDSF# and IPGDSF#, we use the best combination of $\lambda = 2$ and $\delta = 0.9$ in the equation for H_i. Since we have already demonstrated that GDSF# is the champion of all the algorithms in terms of both hit rate and byte hit rate [13, 14, 15, 16, 17], we have not chosen other algorithms for the comparison. LRU is chosen as a baseline algorithm.

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.

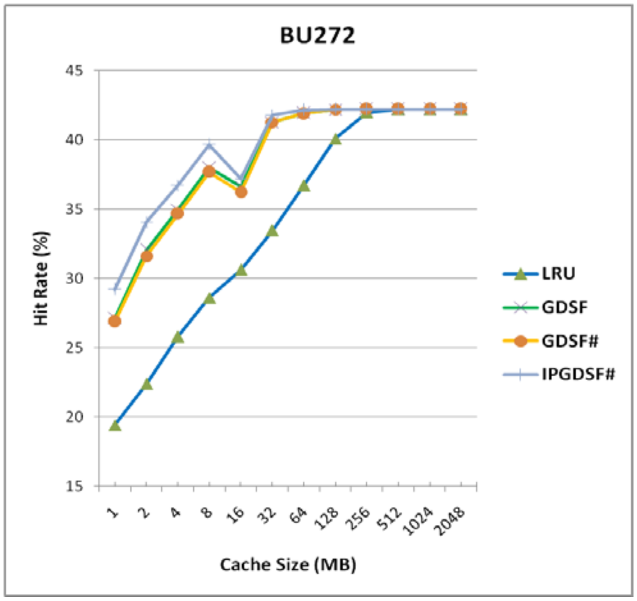
Byte Hit Rate (BHR) Byte hit rate (BHR) is concerned with how many bytes are saved. This is the ratio of the number of bytes satisfied from the cache to the total bytes requested.

5. Simulation Results

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

5.1 Simulation Results for BU272

Fig. 2 gives the comparison of IPGDSF# with other algorithms.



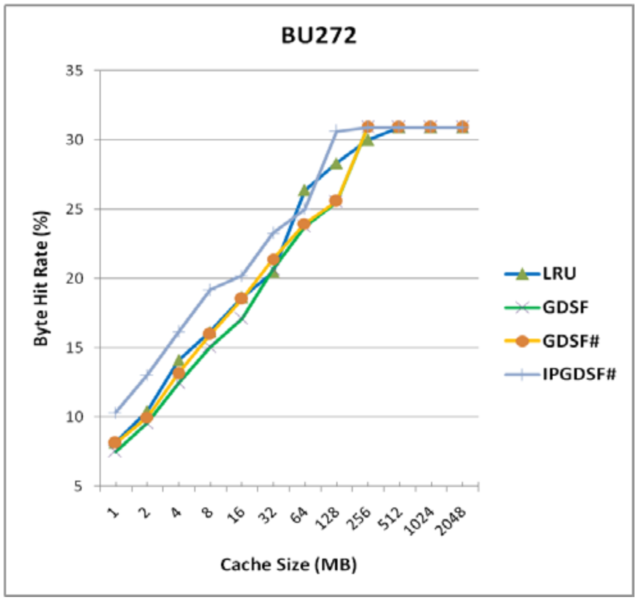


Fig. 2 Comparison of IPGDSF# with other algorithms using BU272 trace

From Figure 2, it can be seen that IPGDSF# outperforms all other algorithms in terms of hit rate as well as byte hit rate for the BU272 data. In case of hit rate, for a cache size of 16MB, there is a performance gain of 6.59% (from 30.62% to 37.21%) over LRU, 0.58% (from 36.63% to 37.21%) over GDSF and 0.99% (from 36.22% to 37.21%) over GDSF#.

In case of byte hit rate, for a cache size of 16MB, there is a performance gain of 4.62% (from 18.64% to 23.26%) over LRU, 6.16% (from 17.10% to 23.26%) over GDSF and 4.73% (from 18.53% to 23.26%) over GDSF#. The graphs, as expected, converge as the cache size grows.

5.2 Simulation Results for BU-B19

Figure 3 gives the comparison of IPGDSF# with other algorithms.

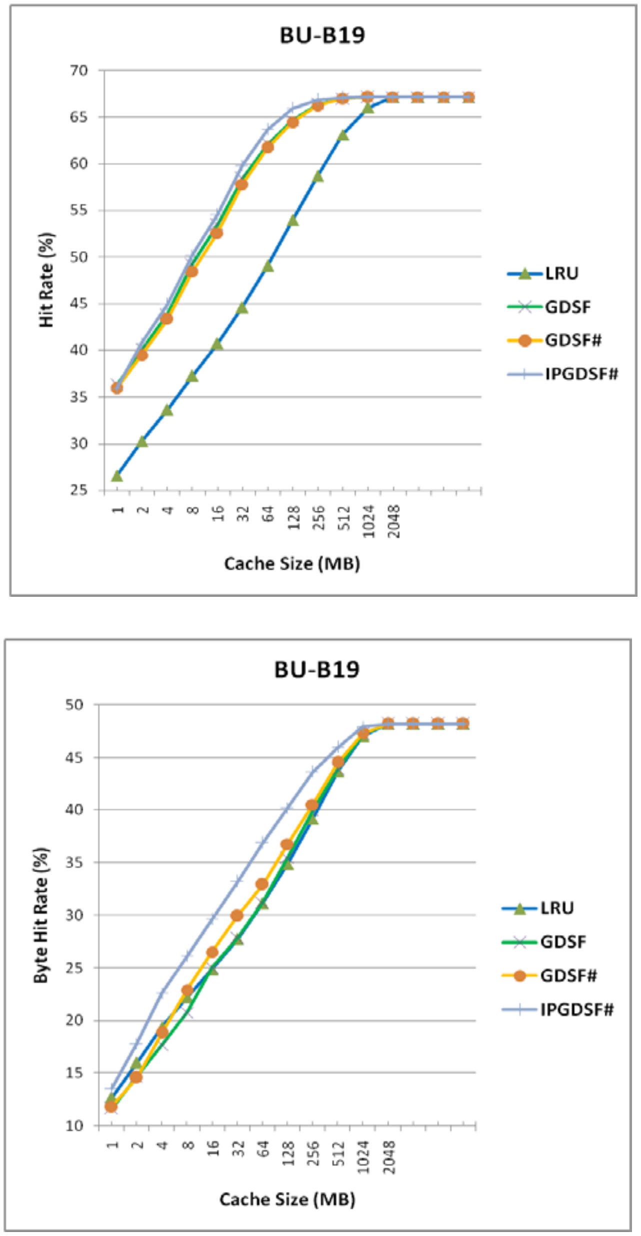


Fig. 3 Comparison of IPGDSF# with other algorithms using BU-B19 trace

Similarly, from Figure 3, it can be seen that IPGDSF# outperforms all other algorithms in terms of hit rate as well as byte hit rate for the BU-B19 data. In case of hit rate, for a cache size of 64MB, there is a performance gain of 14.6% (from 49.06% to 63.66%) over LRU, 1.6% (from 62.06% to 63.66%) over GDSF, and 1.95% (from 61.71% to 63.66%) over GDSF#.

In case of byte hit rate, for a cache size of 64MB, there is a performance gain of 5.74% (from 31.15% to 36.89%) over LRU, 5.64% (from 31.25% to 36.89%) over GDSF, and 3.97% (from 32.92% to 36.89%) over GDSF#. The graphs, as expected, converge as the cache size grows.

5.3 Simulation Results for BU98

Figure 4 gives the comparison of IPGDSF# with other algorithms.

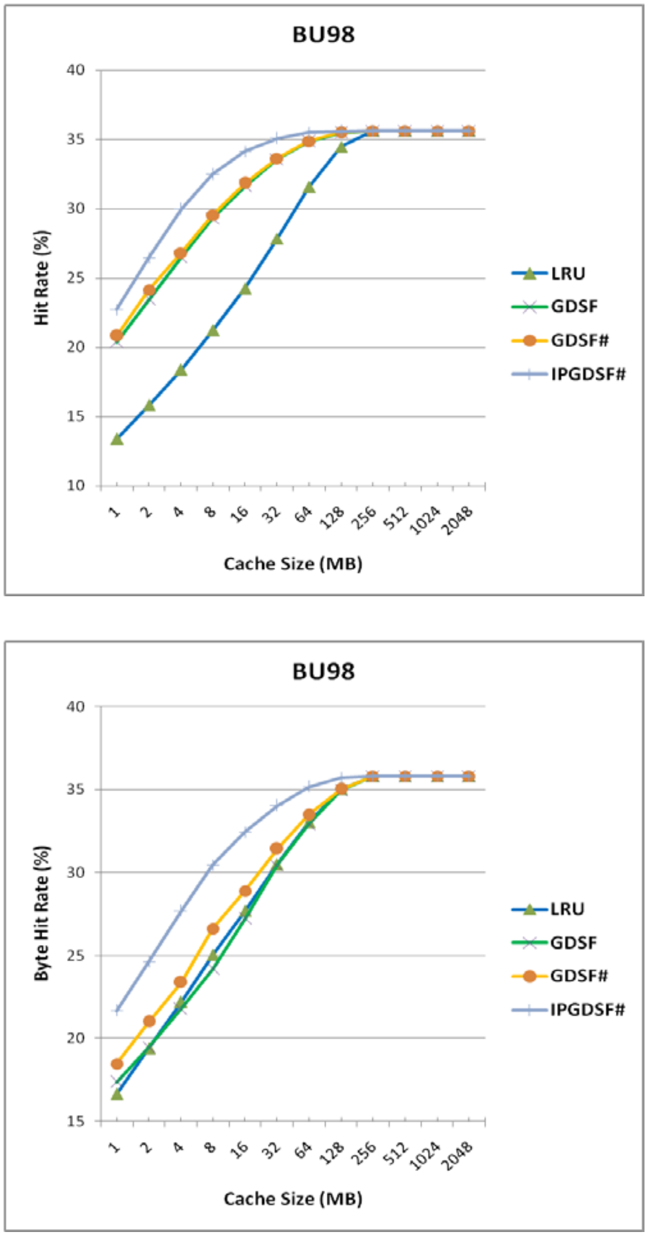


Fig. 4 Comparison of IPGDSF# with other algorithms using BU98 trace

Similarly, from Figure 4, it can be seen that IPGDSF# outperforms all other algorithms in terms of hit rate as well as byte hit rate for the BU98 data. In case of hit rate, for a cache size of 32MB, there is a performance gain of 7.86% (from 27.84% to 35.7%) over LRU, 2.13% (from 33.57%

to 35.7%) over GDSF, and 2.06% (from 33.64% to 35.7%) over GDSF#.

In case of byte hit rate, for a cache size of 32MB, there is a performance gain of 3.53% (from 30.49% to 34.02%) over LRU, 3.57% (from 30.45% to 34.02%) over GDSF, and 2.6% (from 31.42% to 34.02%) over GDSF#. The graphs, as expected, converge as the cache size grows.

6. Conclusions

In this paper, we have proposed an Intelligent Predictive Web caching algorithm, IPGDSF#, capable of adapting its behavior based on access statistics. This algorithm is based on the GDSF# algorithm, which we proposed in [13, 14, 15, 16, 17]. IPGDSF# considers future frequency in calculating the key value of the document, i.e. we look ahead in time in the request stream and adjust the replacement policy. The future frequency is mined from Web server logs using the simple statistical techniques. We make the policy intelligent by periodically updating future frequency when some condition becomes false.

We compare IPGDSF# with cache replacement policies like LRU, GDSF, and GDSF# for Web proxies, using a trace-driven simulation approach. We conduct several experiments using three Web proxy traces. We use metrics like Hit Ratio (HR) and Byte Hit Ratio (BHR) to measure and compare performance of these algorithms.

Our study shows that IPGDSF# outperforms all other algorithms in terms of hit rate as well as byte hit rate. GDSF# has improved performance in case of both HR and BHR. Now IPGDSF# has further improved both the metrics. Thus, we find that our approach gives much better performance than the other algorithms, in the quantitative measures such as hit ratios and byte hit ratios of accessed documents. We believe that use of future frequency coupled with the adaptiveness is indeed the reason that makes our approach preferable to any other caching algorithm.

References

[1] M. Arlitt, R. Friedrich, & 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, & 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 & C. Williamson, “Trace Driven Simulation of Document Caching Strategies for Internet Web Servers”, Simulation Journal, Volume 68, Number 1, Pages 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 & S. Irani, “Cost-Aware WWW Proxy Caching Algorithms”, In Proceedings of the USENIX Symposium on Internet Technology and Systems, Pages 193-206, December 1997.

[6] S. Jin & A. Bestavros, “GreedyDual*: Web Caching Algorithms Exploiting the Two Sources of Temporal Locality in Web Request Streams”, In Proceedings of the Fifth International Web Caching and Content Delivery Workshop, 2000.

[7] S. Podlipnig & L. Boszormenyi, “A Survey of Web Cache Replacement Strategies”, ACM Computing Surveys, Volume 35, Number 4, Pages 374-398, December 2003.

[8] L. Rizzo, & L. Vicisano, “Replacement Policies for a Proxy Cache”, IEEE/ACM Transactions on Networking, Volume 8, Number 2, Pages 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, Volume 1875, Pages 409-418, Springer-Verlag, Berlin, Germany, 2000.

[10] R. P. Wooster & M. Abrams., “Proxy Caching that Estimates Page Load Delays”, In Proceedings of the Sixth International World Wide Web Conference, Pages 325-334, Santa Clara, CA, April 1997.

[11] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, & E. A. Fox, “Removal Policies in Network Caches for World-Wide-Web Documents”, In Proceedings of ACM SIGCOMM, Pages 293-305, Stanford, CA, 1996, Revised March 1997.

[12] M. F., Arlitt, L. Cherkasova, J. Dilley, R. J. Friedrich, & T. Y Jin, “Evaluating Content Management Techniques for Web Proxy Caches”, ACM SIGMETRICS Performance Evaluation Review, Volume 27, Number 4, Pages 3-11, March 2000.

[13] J. B. Patil and B. V. Pawar, “GDSF#, A Better Algorithm that Optimizes Both Hit Rate and Byte Hit Rate in Internet Web Servers”, International Journal of Computer Science and Applications, ISSN: 0972-9038, Volume 5, Number 4, Pages 1-10, 2008.

[14] J. B. Patil and B. V. Pawar, “Trace Driven Simulation of GDSF# and Existing Caching Algorithms for Internet Web Servers”, Journal of Computer Science, Volume 2, Issue 3, Page 573, March-April 2008.

[15] J. B. Patil and B. V. Pawar, “GDSF#, A Better Algorithm that Optimizes Both Hit Rate and Byte Hit Rate in Internet Web Servers”, BRI’S Journal of Advances in Science and Technology, ISSN: 0971-9563, Volume 10, No. (I&II), Pages 66-77, June, December 2007.

[16] J. B. Patil and B. V. Pawar, “GDSF#, A Better Web Caching Algorithm”, In Proceedings of International Conference on Advances in Computer Vision and Information Technology (ACVIT-2007), Co-sponsored by IEEE Bombay Section, Pages 1593-1600, Aurangabad, India, November 28-30, 2007.

[17] J. B. Patil and B. V. Pawar, “Trace Driven Simulation of GDSF# and Existing Caching Algorithms for Web Proxy Servers”, In Proceedings of The 6th WSEAS International Conference on DATA NETWORKS, COMMUNICATIONS and COMPUTERS (DNCOCO 2007), Trinidad and Tobago, November 5-7, 2007, Pages 378-384, ISBN: 978-960-6766-11-4, ISSN: 1790-5117.

[18] F. Bonchi, F. Giannotti, C. Gozzi, G. Manco, M. Nanni, D. Pedreschi, C. Renso, and S. Ruggieri, “Web Log Data Warehousing and Mining for Intelligent Web Caching,” Data and Knowledge Engineering, Volume 39, Number 2, Pages 165-189, 2001.

[19] F. Bonchi, F. Giannotti, G. Manco, M. Nanni, D. Pedreschi, C. Renso, and S. Ruggieri, “Web Log Data Warehousing and Mining for Intelligent Web Caching,” In Proceedings of International Conference on Information Technology: Coding and Computing (ITCC’01 Pages 0599- , 2001.

[20] Q. Yang, and H.H. Zhang, “Web-Log Mining for Predictive Web Caching”, IEEE Transactions on Knowledge and Data Engineering, Volume 15, Number 4, Pages 1050-1053, July/August 2003.

[21] L.A. Belady, “A Study of Replacement Algorithms for Virtual Storage Computers,” IBM Systems Journal, Volume 5, Number 2, Pages 78-101, 1966.

[22] 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.

[23] M. Busari, “Simulation Evaluation of Web Caching Hierarchies”, MS Thesis, Dept of Computer Science, Uni of Saskatchewan, Canada, 2000.

[24] R. Ayani, Y. M. Teo, & P. Chen, “Cost-based Proxy Caching”, In Proceedings of International Symposium on Distributed Computing & Applications to Business, Engineering & Science, Wuxi, China, December 2002.

[25] M. Busari & C. Williamson, “On the Sensitivity of Web Proxy Cache Performance to Workload Characteristics”, In Proceedings of IEEE Infocom, Anchorage, Alaska, April 2001, 1225-1234.

[26] C. R. Cunha, A. Bestavros, & M. E. Crovella, “Characteristics of WWW Client-based Traces”, Technical Report, BU-CS-95-010, Computer Science Department, Boston University, 1995.

[27] V. Almeida, A. Bestavros, M. Crovella, & A., de Oliveria, “Characterizing Reference Locality in the WWW”, In Proceedings of PDIS’96: The IEEE Conference on Parallel and Distributed Information Systems, Miami, 1996.

[28] L. Breslau, P. Cao, L. Fan, G. Philips, & S. Shenker, “Web Caching and Zipf-like Distributions: Evidence and Implications”, In Proceedings of Conference on Computer Communications (IEEE Infocom), New York, 1999, 126-134.

[29] H. Bahn, S. H. Noh, S. L. Min, & K Koh, “Using Full Reference History for Efficient Document Replacement in Web Caches”, In Proceedings of Second USENIX Symposium on Internet Technologies and Systems, Boulder, Colorado, USA, October 1999.

[30] A. Bradley, “BU Computer Science 1998 Proxy Trace”, Technical Report, Computer Science Department, Boston University, 1999.

[31] P. Barford, A. Bestavros, A. Bradley, & M. Crovella, “Changes in Web Client Access Patterns Characteristics and Caching Implications”, World Wide Web, 2(1-2), 1999.

J. B. Patil did his M. Tech. in Computer Science and Data Processing from Indian Institute of Technology, Kharagpur in 1993 and Ph. D. in Computer Engineering from North Maharashtra University, Jalgaon in 2008. He is currently working as a Principal and Professor in Computer Engineering at R. C. Patel Institute of Technology, Shirpur, India. He is a Member of Member of Institute of Engineers, India and also Life Member of Indian Society for Technical Education and Computer Society of India.

B. V. Pawar did his B. E. in Production Engineering from VJTI, Mumbai in 1986, his M. Sc. In Computer Science from University of Mumbai in 1988, and his Ph. D. in Computer Science from North Maharashtra University, Jalgaon in 2000. He is currently working as Professor and Head of Department of Computer Science, North Maharashtra University, Jalgaon. His current research interests include Natural Language Processing, Web Technologies, Information Retrieval, Web Mining, etc.