An Efficient Web Proxy Algorithm That Reduces Network Latency Time

Pu Huang PHUANG@CS.CMU.EDU
CALD, School of Computer Science, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213

Abstract

Rapid growth of World Wide Web draws great attention to improve its performance. Web proxy caching has been considered as one of the most important techniques to reduce Web traffic, save bandwidth, and lower network latency. Both commercial corporations and academic community have shown a great deal of enthusiasm on proxy caching techniques. In recent years, studies in Web traffic measurement have provided convincing empirical evidence that Web traffic exhibits long-time scale self-similarity. However, what this pattern means to proxy caching remains unclear. In this paper, we propose a new proxy caching algorithm designed to minimize network latency time. This new algorithm, GD(H), is a hybrid extension of two of the best algorithms known today. A novel aspect of GD(H) is that it uses Hurst parameter, a measurement of network traffic self-similarity, to measure traffic load, and thus is traffic load aware. We compare GD(H) with GDSize on a new simulation platform that adopts a new traffic model and can generate close-to-reality traffic load. Experiments show that our algorithm outperforms GDSize in terms of reducing latency time in various traffic load conditions.

1. Introduction

A Web proxy (proxy, for short) is an application level software, usually running on a dedicated computer, providing Web access to end-users inside a local subnet. Every time an user sends out a HTTP request, the proxy first intercepts it and then forwards it to the destination Web server; after getting the response from the server, the proxy forwards it back to the user. Initially designed to provide Internet access to users inside firewalls, now proxies are widely deployed to provide caching service. Since all HTTP requests are forwarded by the proxy (if deployed), it is natural to extend its service by providing transparent caching service to all end users in an intranet.

With the continuous growth of the Web, its performance has drawn a lot of concerns. Users don't want to wait to load webpages into their browsers. Actually, proxy caching has been considered as one of the most effective techniques to reduce Web traffic, save bandwidth, and lower network latency time. An efficient caching algorithm is critical for proxy caching. Several new caching algorithms have been proposed in recent years [5][19][22], however, most of these algorithms are designed to maximize hit ratio or weighted hit ratio. Hit ratio measures the percentage of requests that produce cache hits to all requests; weighted hit ratio is the hit ratio weighted by the size of hit documents. These two measurements are suitable for main-memory caching performance. However, proxy caching is similar to, but not exactly the same as main-memory caching. The propose of proxy caching is to provide end-users a transparent, high-speed access to the Internet. Hence a natural performance measurement of proxy caching should be user experienced latency time. If a proxy is biased to cache small documents, it will produce a high hit ratio while users still have to wait a long time to download large webpages. Weighted hit ratio is a better measurement, however, simply weighting hit ratio with the size of documents does not take the consideration of Web traffic conditions: it may take a long time to download a certain webpage if the traffic load is heavy while a short time if it is light. Liu and Fox [12] illustrate this problem by showing that a single proxy cache almost always *increases* latency time in practice, while a majority of proxy academic studies have reported at least 30% cache hit ratio or weighted hit ratio.

In this paper, we propose a new proxy caching algorithm that is explicitly designed to minimize latency time. Our algorithm, GD(H), takes several factors affecting latency time into consideration, and at the same time keeps a simple form, which makes it attractive in terms of implementation. We identify that a widely used measurement of self-similarity, Hurst parameter, can be used to characterize traffic load as well. We further integrate it into our algorithm to make it traffic load aware. Although literature on proxy caching is voluminous, it seems that recent achievement in network traffic research is ignored, namely, the establishment of self-similar Web traffic model. Self-similar network traffic means that the measured network traffic exhibits nearly unchanged statistic characteristics in a large scale of time range. Studies in network traffic have shown empirically convincing evidence supporting this model. Theoretical work also gives this model a mathematically solid basis. The framework of our algorithm is motivated by GDSize [5] and Hybrid [22] algorithms. We extend them by taking traffic conditions into consideration and empirically show that GD(H) outperforms GD-Size algorithm, which have been shown is superior to Hybrid algorithm.

2. Proxy Simulation Platform and Caching Algorithm

2.1 Web Traffic Model

Figure 1 illustrates the typical traffic pattern generated by a single end user. In this figure, ON periods are the time the documents requested are transfered in the Web, i.e. there is some traffic presented; OFF periods are the time when there is no traffic. Two types of OFF periods are shown in the figure: the inactive OFF periods are the user's thinking time (human-determined time); the active OFF periods are the time when the documents are processed by the computer (computer-determined time). A webpage usually has several embedded components (images, frames, video and audio, etc.). After the arrival of the requested webpage, the user's machine first parses it, then sends out more requests to fetch the embedded components. This is the reason why there are active OFF periods between a sequence of ON periods. Typically, an active OFF period lasts less than 1 second while an inactive OFF period lasts much longer. Empirical work has shown that the distributions of all these three periods are heavy-tailed [6][7]. When a large group of users generate this traffic pattern simultaneously, the supercomposed traffic load is self-similar, as observed in real Web traffic data.

Note that we do not list the distribution of ON periods here because the length of ON periods is the latency time we are going to examine. Typical function forms for the distributions listed above and corresponding

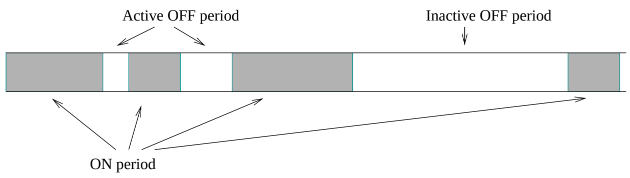

Figure 1. Traffic pattern generated by a single user.

parameters can be found in [1].

2.2 Proxy Caching Algorithm

Our caching algorithm, GD(H), is motivated by Cao and Irani's GD-Size algorithm [5]. GD(H) considers not only downloading time but also the number of references to a document, document size and traffic conditions. In GD(H), each cached document is associated with a value V. When a replacement needs to be made, the document with the lowest V value, min_V, is replaced, and then all pages reduce their V values by min_V. We set V value for each document to be

$$V = H\frac{fs}{B/n} + (1-H)fc, \qquad (1)$$

where f is the number of references; B is the available bandwidth; s is the document size; n is the number of users; c is the time to establish HTTP connection; and $1/2 < H < 1$ is the Hurst parameter representing traffic conditions.

In the case when there is no caching, when a user sends out a request, the request is first queued by the proxy; then the proxy establishes a HTTP connection with the destination Web server and forwards the request to it; finally the requested document is transfered to the proxy and forwarded back to the user. Therefore, the latency time to fetch a document comprises of the waiting time in proxy queue, the HTTP connection time, and the document transfer time. If a document has been requested for f times, the term $\frac{fs}{B/n}$ in equation (1) estimates the sum of waiting time and transfer time[1], and the term fc estimates the total HTTP connection time. If the traffic load is heavy, H should be close to 1 since most of the latency time is spent in waiting in the proxy queue and transferring document. Otherwise, if the traffic load is light, H should

[1]In fact, the document transfer time should be fs/B. However, the waiting time is hard to estimate. So, we just view as if the bandwidth B is shared by n users and estimate the total waiting time and transfer time as fsn/B.

approximate 1/2 since the first and the second terms in (1) contribute equally to the latency time. So, the basic idea of this algorithm is to estimate the total latency time for each document if we do not cache it, and cache the documents with large total latency time.

Hurst parameter measures self-similarity of Web traffic. Willigner et al. [20] show that the superposition of many users who generate the ON/OFF traffic pattern illustrated in above subsection is a fractional Gaussian noise, which is a self-similar stochastic process. Hurst parameter is the only parameter needed to characterize this process. As shown in [20], as the number of users increases, the degree of self-similarity of the superposited traffic increases as well . On the other hand, large number of users means heavy proxy traffic load. Therefore, Hurst parameter can also be use to measure traffic load. Hurst parameter always falls into interval $[1/2, 1]$, 1 means perfect self-similarity (heavy load) and 1/2 means no self-similarity. As we can see, it is perfectly matched with H in formula (1). If the ON periods and the OFF periods have heavy-tailed distributions with degree β_1 and β_2 respectively, then the Hurst parameter of the superposited traffic is

$$H = \frac{3 - min\{\beta_1\ \beta_2\}}{2}. \qquad (2)$$

The main idea behind our algorithm is intuitively simple: directly measure the latency time to download each document. However, rapidly changing Web traffic conditions prevent us from obtaining accurate latency time estimation. We adopt Hurst parameter, a measurement of Web traffic self-similarity, to characterize the traffic load and weight the estimate latency time with this parameter.

2.3 Implementation Issues

Our simulator consists of two parts, a traffic synthesizer and a proxy cache. Following the work of [1], We implement the traffic synthesizer by modelling

1. the document size as a mix of a lognormal distribution $lognormal(9.357, 1.318)$ and a Pareto distribution $pareto(133000, 1.1)$, where the lognormal is the body and the Pareto is the tail;
2. the documents popularity as a Zipf distribution with degree 0.8.
3. the number of embedded components as a Pareto distribution $pareto(1, 2.43)$ with up-bound 138;
4. the length of active OFF periods as a Weibull distribution $weibull(0.382, 1.46)$ with up-bound 1 second.
5. and the length of inactive OFF periods as a Pareto distribution $pareto(1, 1.5)$ with up-bound 1800 seconds

All these distributions and parameters are estimated via real traffic log data, see [1] for details.

Performance is the main issue of implementing the proxy cache. Every time the proxy receives a HTTP request, it first needs to check the cache to see whether the requested webpage is already in the cache. If it is not, after fetching it from the remote Web server, the proxy needs to decide which cached webpage should be replaced by the newly fetched one. If the cache size is huge, membership checking and searching for webpage with minimal V value may take a long time. We use two well-known data structures to maintain the cache: a hashtable, indexed on ID of webpage, is used to conduct the membership check; a B-tree, indexed on V value, is used to facilitate inserting new webpage into the cache and searching for the minimal V value webpage. Therefore, membership checking takes a constant time; inserting and searching take $log(n)$ time,where n is the cache size, the total computation time is $log(n)$.

A FIFO queue connects the traffic synthesizer and the proxy cache. Every HTTP request generated by the synthesizer is first queued and then processed by the proxy according to its entering time.

3. Experiment Results

We compare our GD(H) algorithm with the GDSize algorithm in two scenarios, in the first scenario, there are 64 users and the outbound bandwidth is 1Megabytes per second. This corresponds to a situation where there is a small group of users sharing a limited bandwidth. In the second scenario there are 512 users and bandwidth is 10Megabytes per second. For each scenario, we compare these two algorithms on 8-hour and 4-hour synthetic data. For the GD(H) algorithm, we need to estimate the Hurst parameter. Although formula (2) gives an analytic expression to compute Hurst parameter, it holds in the asymptotic sense, i.e., when the Web traffic converges to a perfect self-similar stochastic process. To better estimate the Hurst parameter, we use the algorithm provided in [18]. The Hurst parameters of these two scenarios are estimated as 0.64 and 0.73, respectively.

Figure 2 and 3 shows the experiment results for scenario one; Figure 4 and 5 shows the experiment results for scenario two. As we can see, GD(H) outperforms GDSize in terms of reducing network latency time.

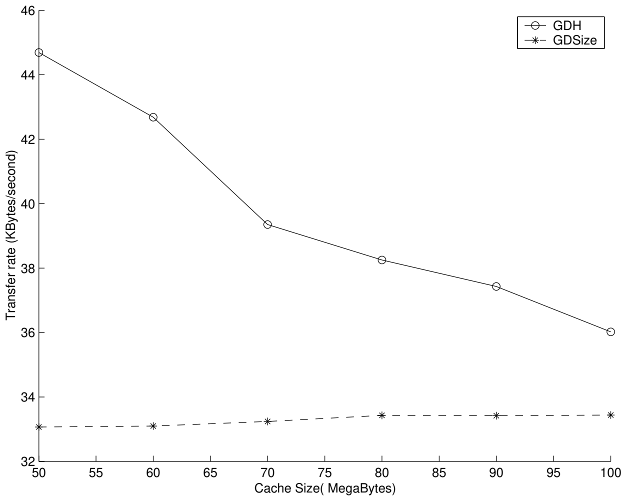

Figure 2. 64 users, 1M bytes per second bandwidth, GDH and GDSize on 4-hour synthetic data.

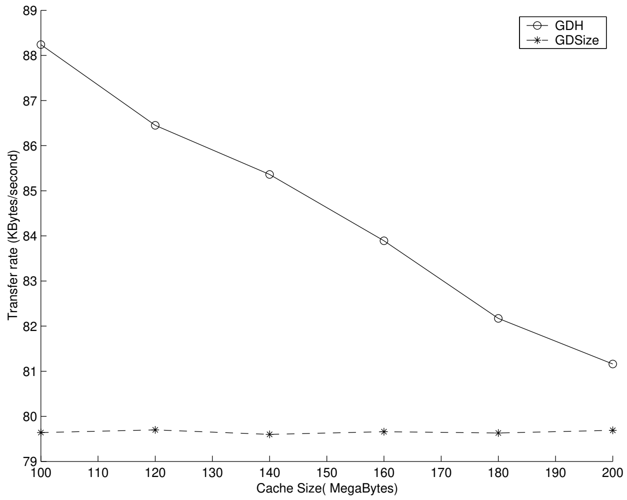

Figure 3. 64users 1M bytes per second bandwidth, GD(H) and GDSize on 8-hour synthetic data.

Two observations about how to interpret the experiment results:

1. If we compare Figure 2 with Figure 3, we can see that the transfer rate in the 8-hour experiment is much higher than that in the 4-hour experiment, even when the cache sizes are the same, say, 100M bytes. The same thing happens for Figure 4 and Figure 5. This happens because all the experiments start with empty cache, it takes certain time to fill up the cache; once the cache has been established, the average latency time will decrease,and the transfer rate will increase.

2. Compared with GDSize algorithm, GD(H) is more sensitive to the cache size. A counter intuitive observation is that the transfer rate of GD(H) algorithm decreases as the cache size increases. This is due to the ON/OFF model. In ON/OFF model, the length of a ON period is the latency time to finish a single HTTP request. Since GD(H) is efficient in reducing this latency time, users tend to send out more requests, which in turn decrease the transfer rate. Another way to show the experiment results is to directly show the total latency time experienced by all users, weighted by the number of requests. However, since file sizes of different requests vary widely (remember file size follows a heavy-tailed distribution), it makes more sense to show the total latency time weighted by the total bytes transfered. In above figures, we show the transfer rate, which is the inverse of this quantity.

4. Related Work

Pioneer work on self-similar network traffic begins with Leland et al. [11], where they show that observed Ethernet LAN traffic exhibits long-range dependence and large time-scale self-similarity. Subsequent work provides convincing empirical evidence that self-similar traffic patterns are prevalent in today's network infrastructure [2][8][10][15]. Willinger et al. [20] provide more insights into the reason why this is the case by showing that the superposited traffic of many ON/OFF processes is self-similar, given that the ON periods and the OFF periods in each process follow heavy-tailed distributions. A bibliographical guide to self-similar traffic research can be found in Willigner et al. [21], where they provide hundreds of references.

With the explosive growth of World Wide Web, Web traffic draws researchers' attention in recent years. Crovella and Bestavros [6] show that Web traffic does exhibit self-similar pattern when the workload is high. Using the ON/OFF model developed by Willigner et al., they also empirically show that the self-similar Web traffic can be attributed to two factors: heavy-tailed Web document distribution and heavy-tailed user thinking time distribution (also see [7] [9] [14]). Mahanti et al. examine Web traffic in a proxy hierarchy and find similar results[13].

Wooster and Abrams's hybrid caching algorithm is designed to reduce latency[22]. Our GD(H) is also motivated by this algorithm. The V value of hybrid algo-

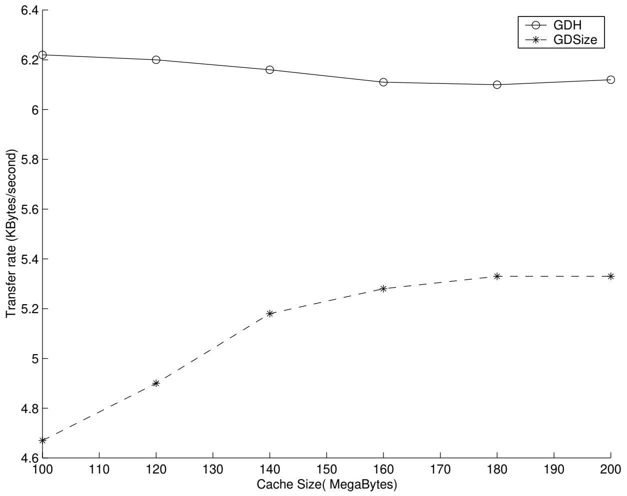

Figure 4. 512 users 10M bytes per second bandwidth, GD(H) and GDSize on 4-hour synthetic data.

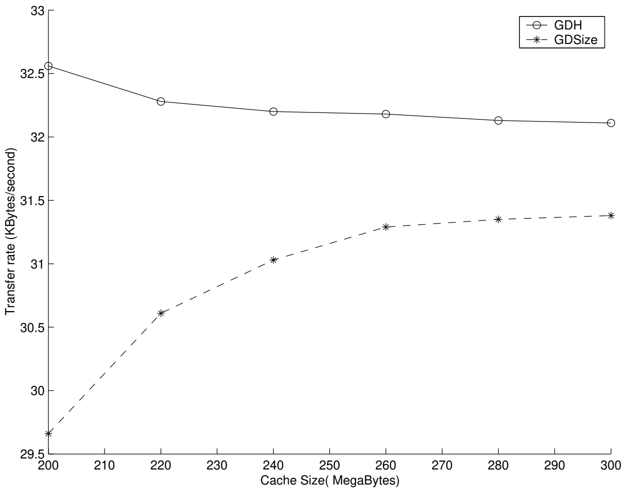

Figure 5. 512 users, 10M bytes per second bandwidth, GDH and GDSize on 8-hour synthetic data.

rithm is computed by

$$V = \frac{(c + w/B)f^y}{s}, \tag{3}$$

where f is the number of references; B is the available bandwidth; s is the document size; c is the time to establish HTTP connection; and w, y are parameters to estimate. Hybrid algorithm is an on-line algorithm which has several parameters to estimate. A full implementation of this algorithm requires updating V value of many pages whenever the estimates of the parameters related to these pages change, which is very time-consuming.

In GDSize algorithm, the V value of each document is simply the inverse of document size. Cao and Irani omit the updating step of Hybrid algorithm and show that GD-Size outperforms the Hybrid in terms of reducing latency even though GD-Size does not consider latency time. Our GD(H) algorithm takes all the factors concerned by the Hybrid algorithm into consideration, plus the number of users and the traffic conditions. Furthermore, GD(H) adopts the framework of GD-Size and is not as heavily parameterized as the Hybrid algorithm, hence it is easy to implement.

5. Conclusion and Discussions

We have empirically shown that GD(H) algorithm, which applies Hurst parameter to estimate the Web traffic load and explicitly estimates the network latency time, is in deed an efficient proxy caching algorithm to reduce latency time. The reason we can use Hurst parameter here is that previous studies have shown that the degree of self-similarity is positive correlated with traffic load. By using Hurst parameter, we actually build a simple traffic model in the proxy, making it traffic load aware. That is the essential reason why it can reduce latency time. If we embed a more precise traffic model into the proxy, we'll expect more efficient caching algorithms, however, more precise models will make the structure of caching algorithm more complicated and hard to implement, therefore, we should make a trade-off between efficiency and complication.

References

[1] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In Proceedings of the ACM SIGMETRICS Conference, Madison, WI, July 1998.

[2] J. Beran, R. Sherman, M.S. Taqqu and W. Willinger, "Long-Range Dependence in Variable-Bit-Rate Video Traffic", IEEE Trans. on Communications, Vol. 43, pp. 1566-1579, 1995.

[3] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. Technical Report 1371, Computer Sciences Dept, Univ. of Wisconsin-Madison, April, 1998.

[4] Ramon Caceres, Fred Douglis, Anja Feldmann, Gideon Glass, and Michael Rabinovich. Web proxy caching: the devil is in the details. In Workshop on Internet Server Performance, Madison, Wisconsin USA, June 1998.

[5] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in Proc. USENIX Symp. on Internet Technologies and Systems, pp. 193–206, December 1997.

[6] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic evidence and possible causes. In Proceedings of the ACM SIGMETRICS 96, pages 160–169, Philadelphia, PA, May 1996.

[7] S. Deng. Empirical Model of WWW Document Arrivals at Access Link. In Proc. International Communications Conference – ICC'96, Dallas, Texas, USA, 23-17 June 1996.

[8] D.E. Duffy, A.A. McIntosh, M. Rosenstein and W. Willinger, "Statistical Analysis of CCSN/SS7 Traffic Data from Working Subnetworks", IEEE J. Select. Areas in Communications, Vol. 12, No. 3, pp. 544-551, 1994.

[9] S. D. Gribble, G. S. Manku, D. Roselli, E. A. Brewer, T. J. Gibson, & E. L. Miller. Self-Similarity in File Systems, in 'Proceedings of ACM SIGMETRICS '98', 1998.

[10] S. Klivansky, A. Mukherjee and C. Song, "Factors Contributing to Self-Similarity over NSFNet", preprint, Georgia Institute of Technology, 1994.

[11] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the Self-Similar Nature of Ethernet Traffic (Extended Version). IEEE/ACM Transactions on Networking, Vol. 2, pp. 1–15, 1994.

[12] Binzhang Liu and Edward Fox. Web traffic latency: Characteristics and implications. In WebNet98, Orlando, FL, November 1998.

[13] A. Mahanti, C. Williamson, and D. Eager, "Traffic Analysis of a Web Proxy Caching Hierarchy", IEEE Network Magazine, May– June 2000

[14] K. Park, G. Kim, and M. Crovella. On the effect of traffic self-similarity on network performance. In Proc. SPIE International Conference on Performance and Control of Network Systems, November 1997.

[15] V. Paxson and S. Floyd, "Wide-area Traffic: The Failure of Poisson Modeling," IEEE/ACM Transactions on Networking, pp.226-244, June 1995.

[16] Proxy Caching. http://www.web-caching.com/proxy-caches.html. 2001.

[17] M. S. Taqqu and J. Levy. Using Renewal Processes to Generate Long-Range Dependence and High Variability. In E. Eberlein and M. S. Taqqu, editors, Dependence in Probability and Statistics, pp. 73–89, Boston, 1986. Birkhauser.

[18] M. Wang and C. Faloutsos, "Datamining meets performance evaluation: Fast algorithms for modeling bursty traffic," *Manuscript*, School of Computer Science, Carnegie Mellon University. February 2001.

[19] 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'96 Conference, pages 293–305, August 1996.

[20] W. Willinger, M. S. Taqqu, W. E. Leland, R. Sherman and D. V. Wilson. Self-Similarity in High-Speed Packet Traffic: Analysis and Modeling of Ethernet Traffic Measurements. IEEE/ACM Transactions on Networking, pp.71-86, Vol. 5, No. 1, 1997.

[21] W. Willinger, M. S. Taqqu and A. Erramilli, A bibliographical guide to self-similar traffic and performance modeling for modern high-speed networks, in Stochastic Networks, F. P. Kelly, S. Zachary and I. Ziedins (eds.), Oxford University Press, Oxford, 1996, 339–366.

[22] R. Wooster and M. Abrams. Proxy Caching the Estimates Page Load Delays. In the 6th International World Wide Web Conference, April, 1997. Santa Clara, CA.