Towards Efficient Web Proxy Caching Algorithms Under Self-Similar Traffic Load

Pu Huang
CALD, SCS
Carnegie Mellon University
Pittsburgh, PA 15213
phuang@cs.cmu.edu

March 10, 2001

Rapid growth of World Wide Web draws great attention to improve its performance. Web proxy caching has been considered 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 bybrid extension of two of the best algorithms known today and hence has a great chance to succeed. A novel aspect of GD(H) is that it uses Hurst parater, a measure of network traffic self-similarity, to measure traffic load, and thus GD(H) is traffic load aware. We will compare GD(H) with other algorithms on a new simulation platform that adopts a new traffic model and can generate close-to-reality traffic load.

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. Each 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 relays it back to the user. Initially designed to provide Internet access to users inside firewalls, now proxies are widely deployed because of their ability to cache previously requested documents. 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. Proxy caching is a technique to reduce Web traffic, save bandwidth, and lower network latency time. An efficient caching algorithm is critical for proxy caching. In this project, we propose a new proxy caching algorithm which is based on one of the best algorithm known today and we further extend them by taking traffic conditions into consideration.

Our algorithm, GD(H), is designed to reduce latency time. GD(H) takes several factors effecting 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 measure 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 finding that Web traffic is self-similar. 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 has shown empirically convincing evidence supporting this finding. Theoretical work also gives this finding a mathematically solid base.

Hit ratio, weighted hit ratio, and latency time are three measures used in evaluating caching algorithms. Hit ratio measures the percentage of requests that produce cache hits to all requests; weighted hit ratio is the hit ratio weighted with the size of hit documents; latency time is time users experienced when downloading webpages. We design our GD(H) algorithm to target latency time instead of hit ratio or weighted hit ratio because latency time is the direct and primary measure of user's satisfaction. Users just want fast Web response time; they don't care about hit ratio or weighted hit ratio, even though proxy caching studies use them as the performance measures. Liu and Fox's study illustrates this mismatch of academical studies and practical concerns. [12] shows a single proxy cache almost always *increases* latency time, while a majority of studies have reported at least 30% cache hit ratio.

We'll evaluate our algorithm by comparing it with GD-Size(1) and LRU via simulation.

The novel aspect of our simulation platform is that our traffic synthesizer is built on a *theoretically* sound and empirically good traffic model, which is developed by researchers in network traffic measurement community. Using this synthesizer, we can evaluate caching algorithms under close-to-reality traffic load. No study on evaluating caching algorithm under this model has been reported.

The outcomes of our project would be: (1) a new simulation platform for evaluating proxy caching algorithms, and with great chance, (2) a new and efficient caching algorithm.

2 Proposed Effort

Our project can be divided into three stages. In stage 1, we will focus on developing the traffic synthesizer. In stage 2, we will implement our GD(H) algorithm and other algorithms, then we will run simulation to get performance data. In stage 3, we will analyze the data to compare the effectiveness of various caching algorithms in terms of reducing latency time.

2.1 Traffic Synthesizer

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.

Based on the above analysis, a traffic synthesizer needs to consider the following factors:

1. the document size distribution;
2. popularity of documents, i.e. the distribution of the number of requests on documents;

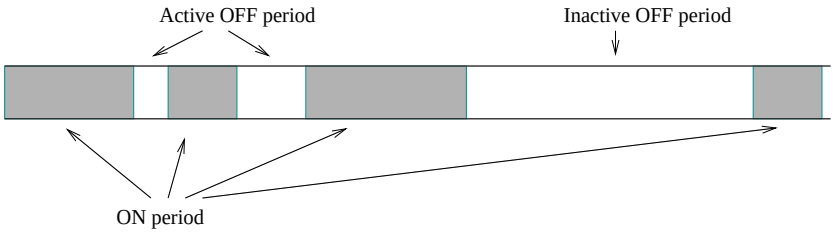

Figure 1: Traffic pattern generated by a single user.

3. the distribution of the number of embedded components in documents;
4. the distribution of the length of active OFF periods;
5. the distribution of the length of inactive OFF periods.

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 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(1) algorithm [5]. GD(H) considers not only download time but also 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

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

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 $1/2 < \alpha < 1$ is a 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 HTTP connection with the destination Web server and forwards the request to it; finally the requested document is transfered to the proxy and obtained by 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, α 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, α should approximate 1/2 since the first and the second terms in (1) contribute equally to 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.

All variables on the r.h.s. of equation (1) are easy to obtain except α. We propose to use Hurst parameter as the traffic condition parameter α. Hurst parameter is a measure of self-similarity. 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 stochastical 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). As we can see, it is perfectly matched with α. 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

$$\alpha = \frac{3 - min\{\beta_1\ \beta_2\}}{2}. \tag{2}$$

The distribution of OFF periods is given in the traffic synthesizer, therefore β_2 is easy to get. For β_1, we need to run simulation to get the distribution of ON periods and draw a log-log plot to obtain its value (see Appendix).

We are going to compare GD(H) with:

1. Least-Recently -Used (LRU) algorithm. LRU evicts the document which was requested the least recently.
2. GD-Size(1), which is a algorithm proposed by Cao and Irani [5]. GD-Size(1) is the same as GD(H) except that the V value for each document is $1/s$, where s is the size of the document.

[5] investigates a dozen of proxy caching algorithms (including LRU) and shows that GD-Size(1) is the best among all of them. Therefore we just compare our algorithm with GD-

[1] In fact, the document tranfer 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 tranfer time as fsn/B.

Size(1) here. We include LRU here because it is the simplest yet most widely used algorithm in real systems and we can use it as a benchmark.

Cao and Irani shows that GD-Size(1), which does not take latency time into account, outperforms other GD-Size-like algorithms which are latency aware. Their explanation of this unexpected result is that "the high variance among loading latencies for the same document reduces the effectiveness of latency-conscious algorithms". However, they did not consider traffic conditions. By taking traffic conditions into consideration, we expect to get a more efficient algorithm.

2.3 Evaluation Method

We will conduct 5-10 simulation runs to compare above algorithms. In each simulation run, the same sequence of requested documents will be fed into each algorithm. (We can do this by setting random number generators the the same seed for each algorithm.) We will check the distribution of ON periods to evaluate the effectiveness of each algorithm in terms of reducing latency time. We expect the length of ON periods follows a heavy-tailed distribution since empirical studies have shown that this is the case in real traffic data and our traffic model is derived from this data. For a heavy-tailed distribution, if we draw a log-log plot for its tail distribution, i.e. $log(1 - F(x))$ on $log(x)$, we will get a line with a constant slope $-\beta$ (see Appendix for details). Large β means short latency time. Therefore, for each simulation run, if we first arrange the data as pairs $\{log(x),\ log(1 - F(x))\}$, where x is the length of a ON period, and $F(x)$ is the proportion of ON periods with length less than x to all ON periods, then we can use linear-regression to evaluate the slope β . Linear regression will also give us estimated confidential interval $Con(\beta)$ for β in each simulation run. After several simulation runs, we can use all the β's and $Con(\beta)$'s to compare the overall effectiveness of each algorithm.

3 Related Work

Pioneer work of 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]. Willigner 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 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. Cao and Irani omit this updating step and show that GD-Size(1) outperforms the hybrid algorithm in terms of reducing latency even though GD-Size(1) 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(1) and is not as heavily parameterized as the hybrid algorithm, hence it is easy to implement.

4 Work Plan

1. Feb. 28, (2 weeks) Develop traffic synthesizer.
2. Mar. 15, (2 weeks) Implement proxy caching algorithms and run simulation.
3. Apr. 1,(2 weeks) Evaluate results.
4. Apr. 15, (2 weeks) Project write-up.

Appendix

In this appendix, we provide some background on heavy-tailed distribution and self-similar network traffic.

Heavy-Tailed Distribution

A probability distribution function $F(x) = P\{X < x\}$ is said to be heavy-tailed if its tail approximates a power function,

$$P\{X > x\} \sim x^{-\beta}, \qquad as\ x \to \infty, \tag{3}$$

where $\beta > 0$ is a constant. The main difference between heavy-tailed distribution and other widely used distributions is that the tail of the former converges to 0 much slower than the later does. For example, the tail of an exponential distribution with mean $1/\lambda$ is

$$P\{X > x\} = e^{-\lambda x},$$

which means this tail converges to 0 at an exponential rate $e^{-\lambda x}$. Compared to power converging rate $x^{-\beta}$, an exponential rate is much faster. Give an example, when $\beta = 2$ and $\lambda = 1$, the probability of random variable X to be greater than 10 is about 0.01 in (3); while it is about about 4.54×10^{-5} in the exponential distribution, which is about 220 times lesser than that in (3). With numbers much greater than 10, the difference increases much fast, in fact, exponentially faster. That is the reason why this family of distributions are named as heavy-tailed: they have long tails and thus the probability of some usual events, such as a big X shows up, is much higher than that in other widely used distributions, such as the exponential family.

One classic example of heavy-tailed distribution is Pareto distribution. By definition, a Pareto distribution has the form

$$F(x) = P\{X < x\} = 1 - (a/x)^{\beta},$$

Where $\beta > 0$ and $x > a$. As we can see, its tail has the form

$$a^{\beta} x^{-\beta},$$

which matches (3). The probability density function of Pareto distribution is

$$f(x) = \beta a^{\beta} x^{-\beta-1}.$$

Self-Similarity and Long-Range Dependence

Consider a stochastic process $\{X_t\}$ with constant mean $\mu = E[X_t]$ and finite variance $\sigma^2 = E[(X_t - \mu)^2]$. We assume the autocorrelation function of this process is *stationary,* i.e.,

$$r(k) = \frac{E[(X_t - \mu)(X_{t+k} - \mu)}{\sigma^2},$$

where $k = 1, 2, 3, ...$, is only a function of lag parameter k. If as k goes to infinity, the autocorrelation function follows a power law, i.e. has the form

$$r(k) \sim k^{-\beta}, \qquad as\ k \to \infty, \tag{4}$$

where $0 < \beta < 1$, then we say process $\{X_t\}$ is long-range dependent. Intuitively, power law $k^{-\beta}$ means that autocorrelation between X_t and X_{t+k} decreases slowly as k increase, i.e., events in a

long time future are dependent on the current event and this dependence decreases by following a power law.

Now, consider an aggregated process $\{X_t^{(m)}\}$ constructed from $\{X_t\}$ by averaging the series $\{X_t\}$ over non-overlapping blocks of size m, i.e., using the following formula

$$X_t^{(m)} = \frac{X_{tm-m+1} + ... + X_{tm}}{m},$$

where $t = 1, 2, 3.....$ Since autocorrelation function $r(k)$ of original process $\{X_t\}$ is stationary, the autocorrelation function $r^{(m)}(k)$ of the derivative process $\{X_t^{(m)}\}$ is also stationary. By self-similarity, we mean that

$$r^{(m)}(\cdot) = r(\cdot),$$

for all $m = 1, 2, ...$ If this is the case, then in any time scale order m, the process $\{X_t\}$ just looks the same as its m order average $\{X_t^{(m)}\}$.

Fractional Gaussian noise $\{X_t\}$ is a widely used self-similar process. It has a constant mean μ and a finite variance σ^2. Its autocorrelation function is

$$r(k) = \frac{(k+1)^{2H} - k^{2H} + (k-1)^{2H}}{2},$$

where $k = 1, 2, 3.....$ and H is the $Hurst$ parameter. It can be shown that as k goes to infinity,

$$r(k) \sim H(2H-1)k^{2H-2}.$$

Compared with (4), we can see that

$$H = 1 - \frac{\beta}{2}.$$

For $0 < \beta < 1$, we have $\frac{1}{2} < H < 1$. Also, it can be shown that the aggregate process $\{X_t^{(m)}\}$ has an exactly same autocorrelation function as $\{X_t\}$. As we can see, Hurst parameter H is the only parameter in the autocorrelation function of Fractional Gaussian noise and self-similarity is defined in terms of autocorrelation function $r(\cdot)$, thus, H is also a measure metric of the degree of self-similarity. As $H \to 1$, both the degree of self-similarity and the long-range dependence of Fractional Gaussian noise increase. As $H \to \frac{1}{2}$, Fractional Gaussian noise degenerates to ordinary Brownian Motion.

Fractional Gaussian noise is an appealing model for network traffic mainly because of two reasons: first, it is simple and easy to characterize, only one parameter H needs state; second, it matches with empirical data reasonably well.

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] C. Wills and M. Mikhailov, Examining the Cacheability of User-Requested Web Resources ", Proceedings of the 4th International Web Caching Workshop, San Diego, CA, March/April 1999.

[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.