Mining Web Logs to Improve Web Caching and Prefetching

Qiang Yang, Henry Haining Zhang, Ian T.Y. Li, and Ye Lu

School of Computing Science
Simon Fraser University
Burnaby, BC, Canada V5A 1S6
(qyang, hzhangb, tlie, yel)@cs.sfu.ca

Abstract. Caching and prefetching are well known strategies for improving the performance of Internet systems. The heart of a caching system is its page replacement policy, which selects the pages to be replaced in a proxy cache when a request arrives. By the same token, the essence of a prefetching algorithm lies in its ability to accurately predict future request. In this paper, we present a method for caching variable-sized web objects using an n-gram based prediction of future web requests. Our method aims at mining a prediction model from the web logs for document access patterns and using the model to extend the well-known GDSF caching policy. In addition, we present a new method to integrate this caching algorithm with a prediction-based prefetching algorithm. We empirically show that the system performance is greatly improved using the integrated approach.

Keywords: Web log mining, web caching and prefetching

1 Introduction

As the World Wide Web rapidly grows, researchers and industry practitioners have designed various effective caching algorithms to contain improve the Web's quality of service. The idea behind web caching is to maintain a highly efficient but small set of retrieved results in a system cache, such that the system performance can be notably improved since common user queries can be directly answered by objects in the cache.

Lying in the heart of caching algorithms is the so-called ``page replacement policy", which specifies conditions under which a new page will replace an existing one. Many replacement policies have been proposed over the years, including the LRU-K algorithm [OOW93], which rejects the Least-Recently-Used objects in most recent *K* accesses, the GD-size policy [CI97] which considers access costs and varying page sizes, and an enhancement of the GD-size algorithm known as GDSF [ALCJ99] which incorporates the frequency information. The basic idea behind most of these caching algorithms is to rank objects based on their access trend by considering factors such as size, frequency and cost. The pages that are "young", relative to its last access, are ranked higher while pages that are "old" are ranked lower. Researchers have also looked at prefetching popular documents in order to improve system performance [CFKL95, MC98, CY97, Duc99].

N. Zhong et al. (Eds.): WI 2001, LNAI 2198, pp. 483–492, 2001.

In the Internet environment, the paging needs usually ranges over an extended period of time, and the fetching of a web page can be a long process. For example, a group of users querying a travel web site about potential travel plans will likely pose requests over a period of several days. Once the request is completed, the reference rate will gradually die out as the users' interest in this page is reduced. When this happens, it is important for a web server or a proxy server to notice the change in this trend, so as to better prepare for the caching of objects that are more likely to be accessed often in the near future.

The key observation we made is that it is possible to learn a predictive model using the vast amount of browsing log data and use it to enhance our caching policy. The past log data provides training for the predictive model, which can be used for predicting web objects that are likely to be accessed by users based on current access sequences. Those objects in the cache that are predicted to occur will be given a higher priority when computing a ranking of the cached objects. As a result, objects that are retained in the cache are more likely to be used again. In addition, although prediction can play a role of retaining objects in the cache when they are about to be replaced, thus having a significant impact in increasing caching performance, many predicted objects fall outside the cache. Therefore, to fully exploit the power of prediction we introduce an integrated caching and prefetching algorithm. In this approach, we use prediction for both object retention and prefetching. The contribution of our work is twofold. First, we introduce a predictive caching algorithm which utilizes a trained prediction model from past log data to further enhanced the well-known GDSF caching policy. Second, we propose a new method to integrate the predictive caching algorithm with an n-gram based prefetching algorithm to more efficiently exploit the predictive power of the n-gram based prediction approach.

The organization of the paper is as follows. In the next section, we review the work in caching and present our motivation. In Section 3 we introduce the formal prediction model for n-gram based prediction, and present the experimental results using the proposed model in predictive caching. In Section 4, we integrate prefetching into the caching model, and conclude in Section 5.

2 Background

Caching is a mature technique that has been widely applied in many computer science areas. Among those areas, Operating Systems and Databases are two most important ones. Currently, the World Wide Web is becoming another popular application area of caching. Below, we briefly review caching in these areas.

The widespread LRU (Least-Recently-Used) algorithm is a good approximation to the optimal page replacement algorithms by considering the age as well as the reference frequency of a page. It is based on the assumption that pages which have been heavily used in the past will probably be used again in the near future, and pages which have not been used recently will probably remain unused for a long time. Consequently, in the LRU algorithm, the page that has not been used for the longest period of time is replaced. This algorithm is chosen as the page replacement policy by almost all commercial systems.

The LRU-K algorithm is motivated by knowing that the popular LRU algorithm is not always appropriate for the database environment (for more details, see [CD85]). The key observation is that LRU keeps only the time of last reference to each page when making page replacement decision. Thus, the LRU algorithm cannot well distinguish between frequently referenced pages and infrequently referenced pages due to the limited information it is based on. The basic idea of LRU-K is to consider the time of the last *K* references to a page and uses such information to make page-replacement decision. To quote the original description in [OOW93] :

> The page to be dropped (i.e., selected as a replacement victim) is the one whose Backward K-distance, is the maximum of all pages in buffer.

The increasing usage of the World Wide Web has led to a great deal of traffic on the Internet, which in turn results in the degradation of network performance. Therefore, it is desirable that the traffic is reduced or smoothed by caching the popular web objects. With this goal, web caching, noted for variable-sized objects, has become an active research area and gained a lot of attention [AWY99,Mar96,Gla94].

Caching can be done at various levels in the network. It can lie on the side of a web server, a caching proxy, or a client. Caching proxies are a special kind of servers that are responsible for locating a cached copy of an object that is required. Web servers and caching proxies are higher-level caches, while client caches are at lower levels. Web caching is different from traditional caching in several ways. An important difference is that the size of web objects is not uniform, and the transfer time costs are not unique either. For the purpose of maximizing the hit ratio or byte-hit ratio, it is better for a caching replacement policy to take factors such as size and network cost into account.

One of the most successful algorithms is the GD-size algorithm introduced by Pei and Irani [CI97]. When a new object arrives, GD-size increases the ranking of a new object *i* by the cost of the removed object. Let S_i be the size of the new object *i*, C_i be the cost of object *i*, and K_i, the key value, be the rank of object *i*. Furthermore, let L be an inflation factor for a newly admitted object, where *L* is updated as the key value of the most recently removed object. Then, if *i* is in the cache, then the key value of object *i* is

$$K_i = L + C_i / S_i \tag{1}$$

Otherwise, if *i* is new and not yet in the cache, then

$$L = \min_j K_j \tag{2}$$

where *j* denotes objects in the cache. Then the object *l* with $K_l=L$ is ejected, and object *i* is inserted in the cache with K_i set according to Equation 1.

As an enhancement of the GD-size algorithm, Arlitt et. al [ALCJ99] introduced the frequency factor F_i which counts of number of references so far. With this new factor, the key value can be computed as

$$K_i = L + F_i * C_i / S_i \tag{3}$$

Both algorithms perform very well across a number of domains, as shown in a number of empirical tests [ALCJ99,CI97].

Another performance improvement strategy is to *prefetch* documents that are highly likely to occur. Prefetching can be done on either the server side, client side or the proxy-server side [CFKL95] discussed an integrated model of prefetching and caching in a file system. [MC98] discussed an approach called top-ten for prefetching, by prefetching the top-ten popular documents. In [CY97] Chinen and Yamaguchi prefetch the referenced pages from hyperlinks embedded in the current object. [Duc99] improved this idea by also considering the frequency of accesses of the hyperlinks. Later in the paper in Section 5, we discuss an integrated model by combining prediction, caching and prefetching in a unified framework, and demonstrate that the resulting system outperforms caching alone.

3 Learning N-gram Models from Web Server Logs

Given a web-server browsing log *L*, it is possible to train a path-based model for predicting future URL's based on a sequence of current URL accesses. This can be done on a per-user basis, or on a per-server basis. The former requires that the user-session be recognized and broken down nicely through a filtering system, and the latter takes the simplistic view that the accesses on a server is a single long thread. We now describe how to build this model using a single sequence *L* using the latter view. An example of the log file is shown in Figure 1.

Fig. 1. NASA log file example

We build an n-gram prediction model based on the object-occurrence frequency. Each sub-string of length n is an n-gram. These sub-strings serve as the indices of a hash table *T* that contains the model. During its operation, the algorithm scans through all sub-strings exactly once, recording occurrence frequencies of documents' requests of the next *m* clicks after the sub-string in all sessions. The maximum occurred request (conditional probability greater than *theta*, where *theta* is a threshold set at 0.6 in our experiments) is used as the prediction of next *m* steps for the sub-string. In this case we say that the n-gram prediction has a window-size *m*. The algorithm for building a path-based model on sub-strings is described below.

We observe that many of the objects are accessed only once or twice, and a few objects are accessed a great many times. Using this well-known fact (also known as the *Zipf* distribution), we can filter out a large portion of the raw log file and obtain a

compressed prediction model. In our filtering step, we removed all URL's that are accessed 10 times or less among all user requests.

As an example, consider a sequence of URL's in the server log: {A,B,C,A, B,C,A,F} Then a two-gram model will learn on the two-grams shown in Table 1, along with the predicted URL's and their conditional probabilities.

Table 1. A learned example of n-gram model

2-Gram	Prediction
A, B	{ < C, 100% > }
B, C	{ < A, 100% > }
C, A	{ < B, 50% >, < F, 50% > }

Applying n-gram prediction models has a long tradition in network systems research. Su et. al. [SYLZ00] compared *n*-gram prediction models under different sized *n*, and presented a cascading algorithm for making the prediction. Silverstein et al [SHMM98] provided a detailed statistical analysis of web log data, pointing out the distribution of access patterns on web pages. [KL96,JP99,SKS98] studied path-based prediction models for networked systems. In our work, we need to adapt the prediction

Normally, there simultaneously exist a number of sessions on a web server. Based on their access sequences, our prediction model can predict future requests for each particular session. Different sessions will give different predictions to future objects. Since our prediction of an object comes with a probability of its arrival, we can combine these predictions to calculate the future occurrence frequency of an object. Let O_i denote a web object on the server, S_j be a session on a web server, $P_{i,j}$ be the probability predicted by a session S_j for object O_i. If $P_{i,j}=0$, it indicates that object O_i is not predicted by session S_j. Let W_i be the future frequency of requests to object O_i. If we assume all the sessions on a web server are independent to each other, we can obtain Equations 4 and 5.

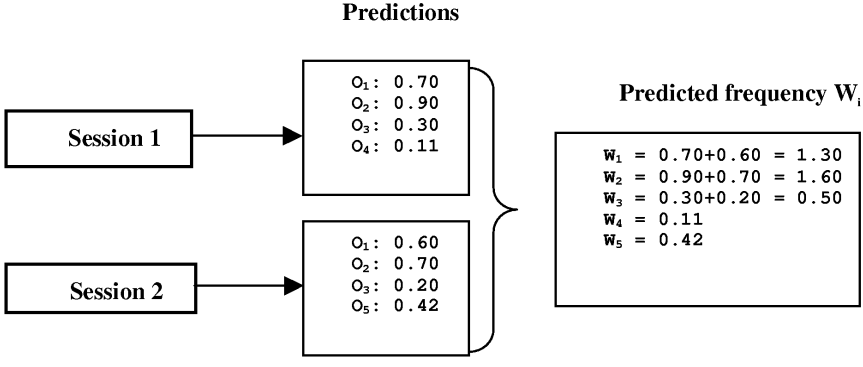

Fig. 2. Prediction to frequency weight calculation

To illustrate Equation 4, we map three sessions in Figure 2. Each of these sessions yields a set of predictions to web objects. Since sessions are assumed independent to each other, we use Equation 4 to compute their weights W_i. For example, object O_1 is predicted by two sessions with a probability of 0.70 and 0.60, respectively. From

$$W_i = \sum_j P_{i,j} \tag{4}$$

Equation 4, W_1 = 1.3. This means that, probabilistically, object O_1 1.3 times in the near future.

Once the access frequency W(p) of a page p is predicted, we incorporate the W(p):

$$K(p)=L+(W(p) + F(p))*C(p)/S(p) \tag{5}$$

In this formula, we add W(p) and F(p) together in Equation 5 as a combined frequency term. This implies that the key value of a page p is determined not only by its past occurrence frequency, but also affected by its future frequency. The more likely it occurs in the future, the greater the key value will be. The rationale behind our extension is that we look ahead some time in the request stream and adjust the replacement policy.

We have conducted a series of experimental comparisons with two data logs that we are able to obtain. In the experiments, the EPA (United States Environmental Protection Agency) data contains a day's worth of all HTTP requests to the EPA WWW server located at Research Triangle Park, NC. The NASA data is from NASA Kennedy Space Center WWW server in Florida containing 17 days' worth of requests. Before experiments, we removed un-cacheable URLs from the access logs. A URL is considered un-cacheable when it contains dynamically generated content such as CGI scripts. We also filtered out requests with unsuccessful HTTP response code.

The results illustrating both hit rates and byte-hit rates are shown in Figure 3. The algorithms under comparison are n-gram, GDSF, GD-Size, LFUDA (least-frequently-used with dynamic aging), and the LRU method [ALCJ99]. Overall, the n-gram algorithm outperforms the other algorithms using all of the selected cache sizes. It is obvious from the figures that the performance gain is substantially larger when the n-gram algorithm is applied on the NASA dataset. This observation can be explained by considering the difference between the two datasets. The EPA dataset is the web log data collected over a period of 24 hours. We have used the first 12 hours of data for training and the remaining data for evaluation. The users' access pattern may vary dramatically between the two time periods and thus decreasing the prediction accuracy. By comparison, 6 days of the NASA log data are used for training while the remaining 7 days of data are used for evaluation. The users' access patterns are much more stable over this extended period of time, making the training data much more representative of the actual access patterns. This no doubt aids tremendously in prediction accuracy.

4 Integrated Caching and Prefetching

So far we have shown that predictive caching improves system performance in terms of hit rate and byte hit rate. These two metrics implicitly reflect reduction of network latency. In this section, we investigate an integrated caching and prefetching model to further reduce the network latency perceived by users. The motivation lies in two aspects. Firstly, from Figure 3, we can see both the hit rate and byte hit rate are growing in a log-like fashion as a function of the cache size. Our results consist with those of other researchers [CI97, RV98, AFJ98, ACDFJ99]. This suggests that hit rate or byte-hit rate does not increase as much as the cache size does, especially when cache size is large. This fact naturally leads to our thought to separate part of the cache memory (e.g. 10% of its size) for prefetching. By this means, we can trade the minor hit rate loss in caching with the greater reduction of network latency in prefetching. Secondly, almost all prefetching methods require a prediction model. Since we have already embodied an n-gram model into predictive caching, this model can also serve prefetching. Therefore, a uniform prediction model is the heart of our integrated approach.

In our approach, the original cache memory is partitioned into two parts: cache-buffer and prefetch-buffer. A prefetching agent keeps pre-loading the prefetch-buffer with documents predicted to have the highest weight W. The prefetching stops when the prefetch-buffer is full. The original caching system behaves as before on the reduced cache-buffer except it also checks a hit in the prefetch-buffer. If a hit occurs in the prefetch-buffer, the requested object will be moved into the cache-buffer according to original replacement algorithm. Of course, one potential drawback of prefetching is that the network load may be increased. Therefore, there is a need to balance the decrease in network latency and the increase in network traffic. We next describe two experiments that show that our integrated predictive-caching and prefetching model does not suffer much from the drawback.

In our prefetching experiments, we again used the EPA and NASA web logs to study the prefetching impact on caching. For fair comparison, the cache memory in cache-alone system equals the total size of cache-buffer and prefetch-buffer in the integrated system. We assume that the pre-buffer has a size of 20% of the cache memory. Two metrics are used to gauge the network latency and increased network traffic:

Fractional Latency: The ratio between the observed latency with a caching system and the observed latency without a caching system.

Fractional Network Traffic: The ratio between the number of bytes that are transmitted from web servers to the proxy and the total number of bytes requested.

As can be seen from Figure 4 and 5, prefetching does reduce network latency in all cache sizes. On EPA data, when cache size is 1% (approximately 3.1MB), fractional latency has been reduced from 25.6% to 19.7%. On NASA data, when cache size is 0.001%(~ 240KB), fractional latency has been reduced from 56.4% to 50.9. However, as can be seen from Figure 4 and 5, we pay a price for the network traffic, whereby the prefetching algorithm incurs an increase in network load. For example, in NASA dataset, the fractional network traffic increases 6% when cache size is 0.01%. It is therefore important to strike for a balance the improvement in hit rates and the

network traffic. From our result, prefetching strategy better performs in a larger cache size while relatively less additional network traffic is incurred.

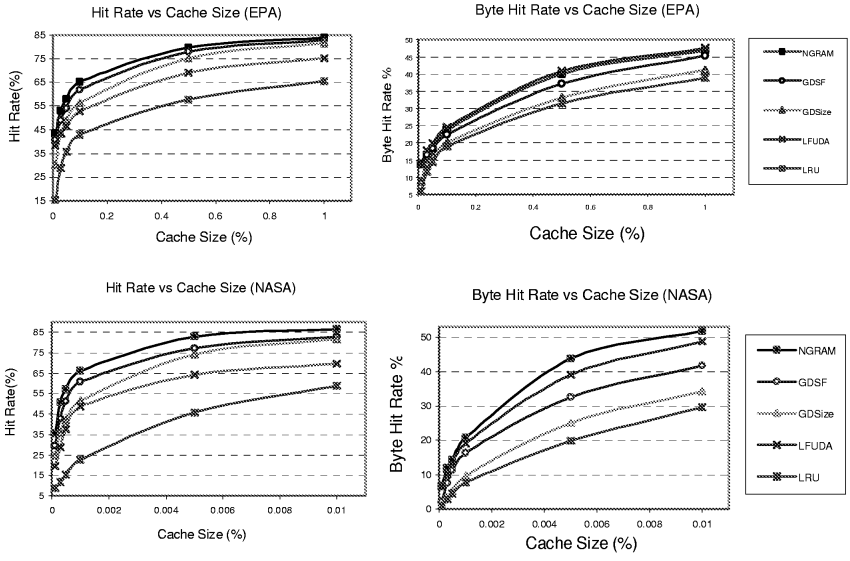

Fig. 3. Hit and byte hit rates results using EPA and NASA logs.

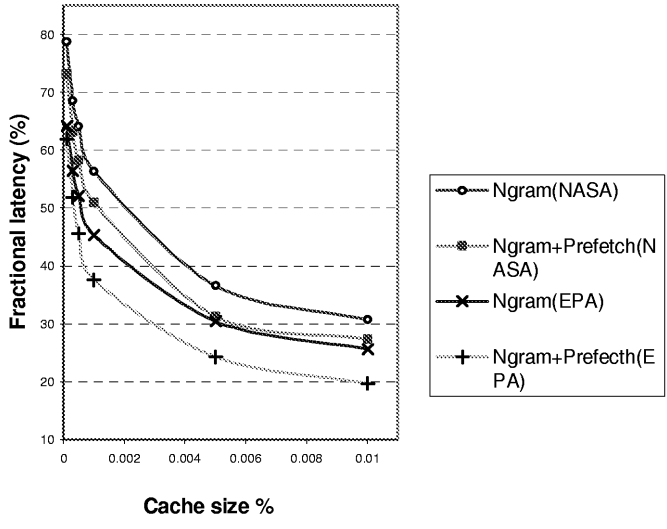

Fig. 4. Fractional Latency Comparison

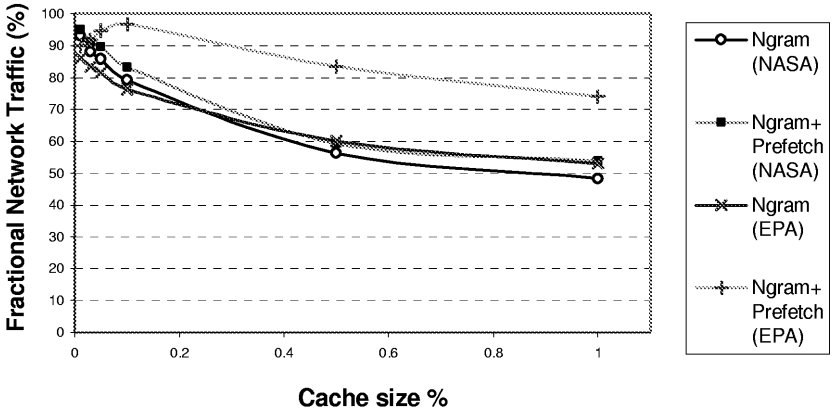

Fig. 5. Fractional Network Traffic

5 Conclusions and Future Work

In this paper, we applied association rules minded from web logs to improve the well-known GDSF algorithm. By integrating path-based prediction caching and prefetching, it is possible to dramatically improve both the hit rate and byte hit rate while reducing the network latency. In the future, we would like to extend our approach by taking into account other statistical features such as the data transmission rates that can be observed over the Internet.

Acknowledgement. We thank Michael Zhang for interesting discussions. This work is supported by Canadian NSERC and IRIS research grants. We also thank Zhong Su and HJ Zhang at Microsoft Research China for early collaboration on n-gram based prediction.

References

[ALCJ99] M. Arlitt, R. Friedrich L. Cherkasova, J. Dilley, and T. Jin. Evaluating content management techniques for web proxy caches. In *HP Technical report*, Palo Alto, Apr. 1999.

[AWY99] C. Aggarwal, J. L. Wolf, and P. S. Yu. Caching on the World Wide Web. In *IEEE Transactions on Knowledge and Data Engineering*, volume 11, pages 94-107, 1999.

[CFKL95] Pei Cao, Edward W. Felten, Anna R. Karlin and Kai Li. A Study of integrated Prefetching and Caching Strategies. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1995.

[CD85] H. T. Chou and D. J. DeWitt. An evaluation of buffer management strategies for relational database systems. In *Proceedings of the Eleventh International Conference on Very Large Databases*, pages 127-141, August 1985.

[CI97] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, Dec. 1997.

[CY97] E. Markatos and C. Chironaki. A Top Ten Approach for Prefetching the Web. In *Proceedings of the INET'98 Internet Global Summit*. July 1998

[Duc99] Dan Duchamp. Prefetching Hyperlinks. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS '99)*, Bouder, CO Oct 1999.

[JP99] Pitkow J. and Pirolli P. Mining longest repeating subsequences to predict www surfing. In *Proceedings of the 1999 USENIX Annual Technical Conference*, 1999.

[KL96] T. M. Kroeger and D. D. E. Long. Predicting future file-system actions from prior events. In *USENIX 96*, San Diego, Calif., Jan. 1996.

[Mar96] E. Markatos. Main memory caching of web documents. In *Computer networks and ISDN Systems*, volume 28, pages 893-905, 1996.

[MC98] K. Chinen and S. Yamaguchi. An Interactive Prefetching Proxy Server for Improvement of WWW Latency. In *Proceedings of the Seventh Annual Conference of the Internet Society (INEt'97)*, Kuala Lumpur, June 1997.

[OOW93] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 297-306, May 1993.

[SKS98] S. Schechter, M. Krishnan, and M.D. Smith. Using path profiles to predict http requests. In *Proceedings of the Seventh International World Wide Web Conference Brisbane, Australia.*, 1998.

[SYLZ00] Zhong Su, Qiang Yang, Ye Lu, and HongJiang Zhang. Whatnext: A prediction system for web requests using n-gram sequence models. In *Proceedings of the First International Conference on Web Information Systems and Engineering Conference*, pages 200-207, Hong Kong, June 2000.

[SYZ00] Zhong Su, Qiang Yang, and HongJiang Zhang. A prediction system for multimedia pre-fetching on the internet. In *ACM Muldimedia Conference 2000*. ACM, October 2000.