Mining Web Logs for Prediction Models in WWW Caching and Prefetching

Qiang Yang
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada V5A 1S6
qyang@cs.sfu.ca

Haining Henry Zhang
IBM E-business Innovation Center
Vancouver
Burnaby, BC, Canada V5G 4X3
haizhang@ca.ibm.com

Tianyi Li
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada V5A 1S6
tlie@cs.sfu.ca

ABSTRACT

Web caching and prefetching are well known strategies for improving the performance of Internet systems. When combined with web log mining, these strategies can decide to cache and prefetch web documents with higher accuracy. In this paper, we present an application of web log mining to obtain web-document access patterns and use these patterns to extend the well-known GDSF caching policies and prefetching policies. Using real web logs, we show that this application of data mining can achieve dramatic improvement to web-access performance.

Keywords

Web Log Mining, Application to Caching and Prefetching on the WWW

1. INTRODUCTION

As the World Wide Web is growing at a very rapid rate, researchers have designed various effective caching algorithms to contain network traffic. The idea behind web caching is to maintain a highly efficient but small set of retrieved results in a cache, such that the system performance can be notably improved since later user requests can be directly answered from the cache. Another performance improvement strategy is to prefetch documents that are highly likely to occur in the near future. Both techniques have been studied in the literature extensively.

An important advantage of the WWW is that many web servers keep a server access log of its users. These logs can be used to train a prediction model for future document accesses. Based on these models, we can obtain frequent access patterns in web logs and mine association rules for path prediction. We then incorporate our association-based prediction model into proxy caching and prefetching algorithms to improve their performance. This strategy works because of the availability of vast amounts of data. We empirically show that this approach indeed improves the system performance over existing algorithms dramatically!

The organization of the paper is as follows. In the next section, we review the work in web caching and prefetching. In Section 3 we introduce the formal association rule based prediction models and show how it integrates with the caching algorithms. Then, in Section 4, we present our experimental results related to this new model. In Section 5, we integrate prefetching into the caching model, and conclude in Section 6.

2. PREVIOUS WORK IN PROXY CACHING AND PREFETCHING

Web caching is an important technique for improving the performance of WWW systems. 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. The basic idea behind most of these caching algorithms is to rank objects according to a key value computed by factors such as size, frequency and cost. When a replacement is to be made, lower-ranked objects will be evicted from the cache. The most successful replacement algorithm is GDSF[9]. It computes the key value of a page p as K(p)= L+F(p)*C(p)/S(p), where L is an inflation factor to avoid cache pollution, F(p) is the past occurrence frequency of p, C(p) is the cost to fetch p and S(p) is the size of p.

Researchers have also considered prefetching popular documents in order to reduce perceivable network latency [7, 10 , 11, 12]. [10] discussed an integrated model of prefetching and caching in a file system. In [11] Chinen and Yamaguchi prefetch the referenced pages from hyperlinks embedded in the current object. [12] improved this idea by also considering the frequency of accesses of the hyperlinks.

We plan to use web-log mining to improve the performance of web caching and prefetching systems. Web log mining is an important part of web mining. It extracts useful knowledge from large-scale web logs for application in other domains. The closest work done previously is [5]. Pitkow and Pirolli studied the pattern extraction techniques to predict the web surfer's path. Su et al. [13] has built an n-gram model to predict future requests. In data mining area, [2] has looked at sequential data mining for transaction data, but they are not applied caching and prefetching. Though they pointed out the possible application in

KDD 01 San Francisco CA USA
Copyright ACM 2001 1-58113-391-x /01/08...$5.00

web caching and prefetching, no actual algorithm was designed for such a task. In contrast, in this paper, we discuss an integrated model by combining association-based prediction, caching and prefetching in a unified framework, and demonstrate that the resulting system outperforms caching alone.

3. BUILDING ASSOCIATION-BASED PREDICTION MODELS

In this section, we present our approach to establish an association-based prediction model on a large-scale web log. Our goal is to find out frequent access path patterns in order to extract association rules that can be utilized to predict future requests.

3.1 Extracting Embedded Objects

HTML documents are the building blocks of the Web. They define the linkage structure of web resources. HTML documents also act as containers of other web objects, such as images, audio and video files. These objects are usually displayed as part of their owner HTML documents; hence, they are called embedded objects. References to embedded objects are usually preceded by their HTML container, therefore they are easy to be recognized from a web log. They appear as a burst of requests from the same client shortly after an HTML access. If an object is observed that its references always come immediately after accesses to particular HTML documents, these HTML documents can be regarded as its containers.

Since there is no linkage information inside embedded objects, they do not contribute to an access path. Therefore, we deal with HTML documents and embedded objects differently. While finding sequences in a session, we do not take embedded objects into considerations. Instead, we just associate them to their corresponding parent HTML document (which is the nearest HTML document requested in the past). We perform preprocessing to extract embedded objects and store them in an *Embedded Object Table* (henceforward referred as EOT).

3.2 Mining Frequent Sequences

After the preprocessing, only HTML documents remain in a request sequence. Every substring of length n is regarded as an n-gram. Unlike the subsequences used in [2], we do not allow gaps between adjacent symbols in our n-gram strings. These n-grams are used as the left-hand-side (LHS) of the association rules. This type of n-gram based association-rule encodes order and adjacency information, and is a special case of the general association rules in sequential data mining. The algorithm scans through all substrings with length ranging from 1 to n in each user session, accumulating the occurrence counts of distinct substrings, and pruning substrings with support lower than a minimum support θ. In our experiments, we set $n = 4$ and $\theta = 2$.

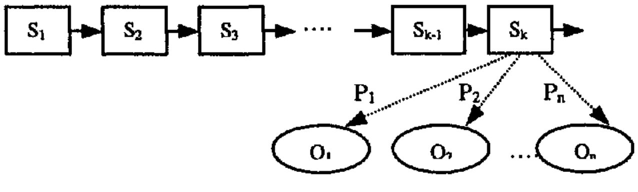


Figure 1. A sequence of requests

3.3 Constructing Association Rules

Figure 1 shows a user session in a web log. The blocks represent HTML documents and the ellipses stand for embedded objects. The solid arrows indicate access paths and the value conf is the conditional probability of transition from an n-gram to a next document. The dotted arrows depict the embed/parent relationship between an HTML file and its embedded objects. The value Pi on the arrow is the probability that object O_i belongs to the document. For illustration purpose, we draw embedded objects only for document S_k. In our analysis of web logs, most Pi are close one. Therefore, in subsequent discussions, we assume that $P_i = 1$.

From the graph, we know that once the frequent sequences have been found, it is straightforward to generate N-gram prediction rules. For each k-string $S_1S_2...S_k$ ($k >= 2$), we can create a rule in the follow format:

$$S_1S_2...S_{k-1} \rightarrow S_k \quad (\text{conf}) \qquad (3.1)$$

The confidence conf, i.e. the conditional probability $P(S_k| S_1S_2...S_{k-1})$, of this rule is expressed in terms of count of sequences:

$$conf = count(S_1S_2...S_k) / count(S_1S_2...S_{k-1}) \qquad (3.2)$$

Furthermore, if S_k has embedded objects, for each object O_i belonging to S_k, the following rules can be deducted immediately from the EOT:

$$S_1S_2...S_{k-1} \rightarrow O_i \quad (conf) \qquad (3.3)$$

Usually, the number of rules generated in this way is large. Hence, to reduce the memory space to store the model, we do not actually generate the rules by Equation (3.3); instead, we just put the EOT in memory and extract rules dynamically. Besides, for rules generated by Equation (3.1), we chop those with *conf* below a threshold h_c. Raising h_c decreases the number of rules needed to be stored. In our experiments, h_c is set between 0 and 0.3 . By this means, we reduce the number of rules and keep only the high confidence ones.

3.4 Prediction Algorithm

The process of building a set of association rules and an EOT is called *training*. Once the training is finished, we can apply these rules to give predictions of future visits. Intuitively, for any given observed sequence of URL's, we choose a rule whose LHS matches the sequence and has the longest length among all applicable rules. The detailed algorithm is given below.

```
Input: W: access sequence of current session; $h_c$: confidence threshold; T: prediction rule table; EOT of T
Output: S: a set of predicted documents with probability
Method:
S = Empty set
for i=W.length downto 1
{ C = rules in T with the form:   W → X    ( $conf_x$ )
  for each rule in C
    if $conf_x$>= hc  then
     { add X( $conf_x$ ) into S;
       for each embedded object Oi of X in table E,
         add $O_i$($conf_x$) into S;  }
  if S is not empty then return S;
  W = W with the first letter removed; }
return empty set;
```

Figure 2 Prediction Algorithm

overhead of hint transmission is negligible to the transmission of the actual web data.

In the previous section, we introduced GDSF algorithm [9], which is one of the best caching replacement algorithms in terms of byte hit rate and hit rate. Our predictive caching algorithm is an extension and enhancement of the widespread GDSF by incorporating a factor of predictive frequency.

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 the following equation:

$$W_i = \sum_j P_{i,j} \qquad (4.1)$$

To illustrate Equation 4.1, we map two sessions in Figure 3. Each of these sessions yields a set of redictions to web objects. Since sessions are assumed independent to each other, we use Equation 4.1 to compute their W_i. For example, object O_1 is predicted by two sessions with a probability of 0.70 and 0.60, respectively. From Equation 4.1, W_1 = 1.3. This means that, probabilistically, object O_1 will be accessed 1.3 times in the near future.

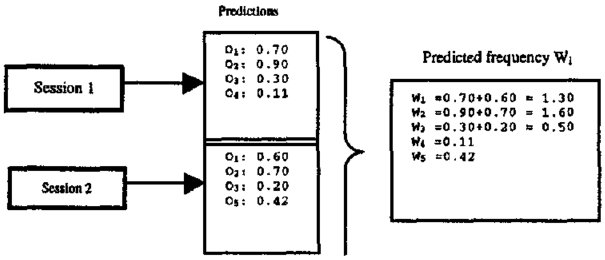


Figure 3 Prediction of future frequency

Once the future access frequency W(p) of a page p can be predicted, we extend GDSF to incorporate the W(p):

$$K(p)=L+(W(p) + F(p))*C(p)/S(p) \qquad (4.2)$$

We add W(p) and F(p) together in Equation 4.2, which 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 uncacheable URLs from the access logs. A URL is considered uncacheable when it contains dynamically generated content such as CGI scripts. We also filtered out requests with unsuccessful HTTP response code. In our experiments, we use two quantitative measures to judge the quality of our extended caching algorithm. Using test web log data, hit rate is defined as the percentage of web requests, out of all web requests in the testing data, that can be answered by the cache. Byte hit rate is the percentage of bytes that are answered directly by documents and objects in the cache, out of the total number of bytes that are requested.

The results illustrating both hit rates and byte-hit rates are shown in Figures 4 to 5. The algorithms under comparison are n-gram, GDSF, GD-Size, LFUDA, and the LRU method [1]. Overall, the n-gram-based algorithm outperforms the other algorithms using all of the selected cache sizes. It is clear 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.

5. INTEGRATED PREDICTIVE CACHING AND PREFETCHING

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 4 to 5, 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 are consistent with those of other researchers [4, 16]. 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.

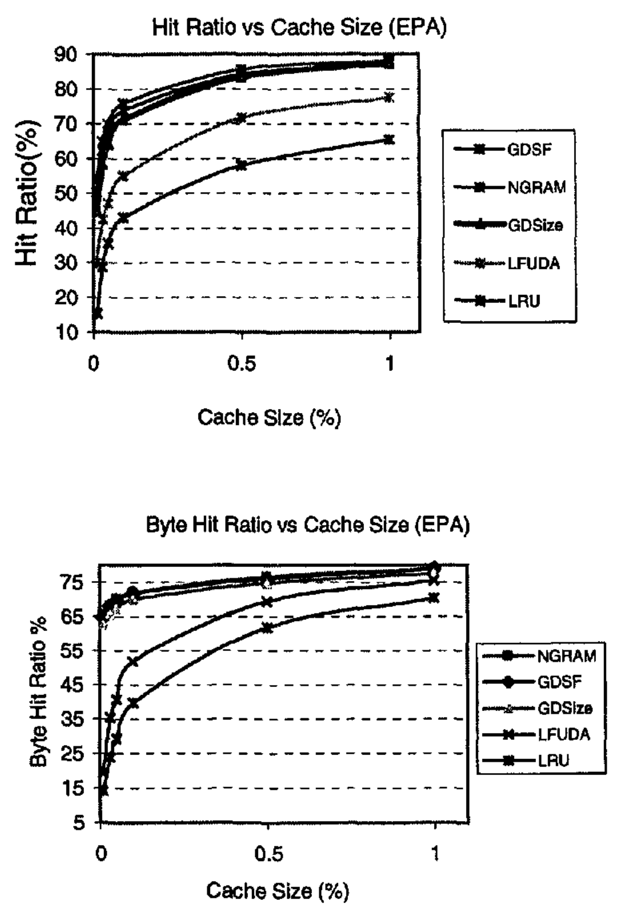
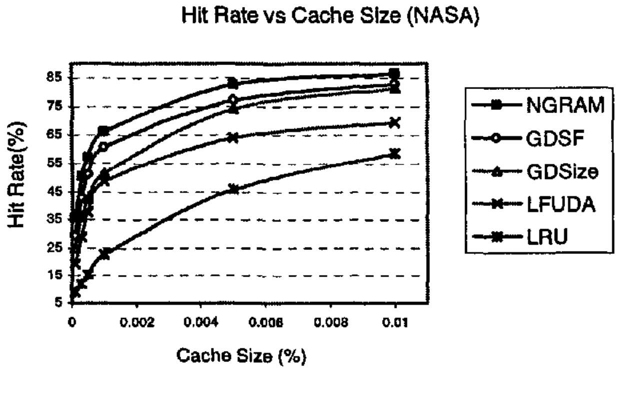


Figure 4 Hit rate and byte hit rate comparison on EPA data



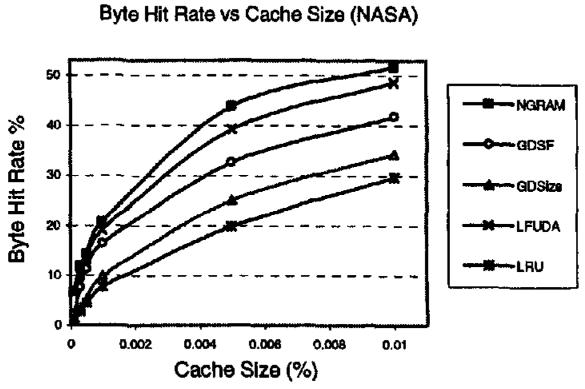


Figure 5 Hit rate and byte hit rate comparison on NASA data

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 W_i. 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 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:

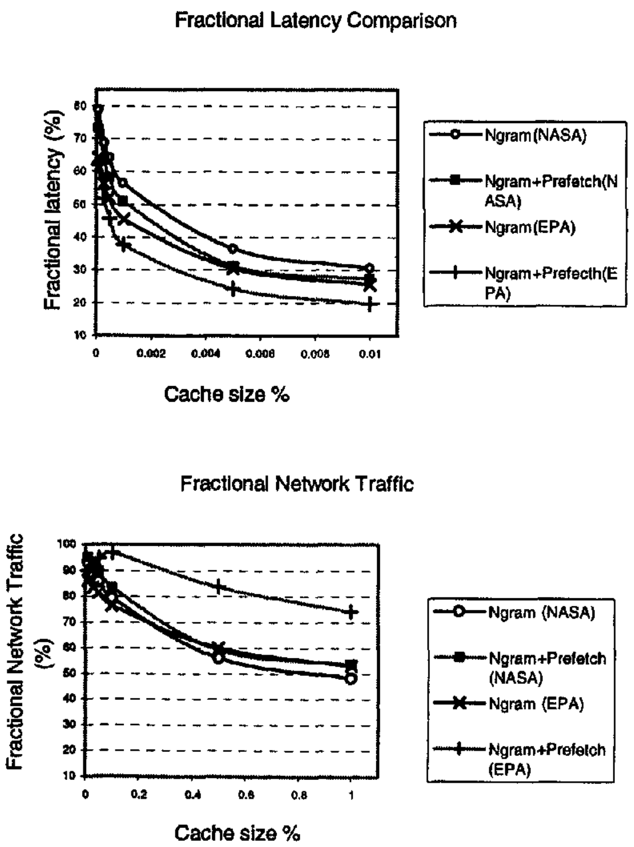
Fractional Latency: The ratio between the observed latency with a caching system and the observed latency without a caching system.

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 6(top), prefetching does reduce network latency in all cache sizes. On EPA data, when cache size is 1% of the dataset, fractional latency has been reduced from 25.6% to 19.7%. On NASA data, when cache size is 0.001% of the dataset, fractional latency has been reduced from 56.4% to 50.9%. However, as can be seen from Figure 6(bottom), 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 incurs.

6. 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.


Fractional Latency Comparison

Fractional Network Traffic

Figure 6 Fractional latency and fractional network traffic comparison

7. ACKNOWLEDGMENTS

We thank Canadian Natural Sciences and Engineering Research Council (NSERC) and Institute for Robotics and Intelligence Systems (IRIS) for their support for this research.

8. REFERENCES

[1] 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.

[2] R. Agrawal and R. Srikant. Minging Sequential Patterns. Proc. Of Int'l Conference on Data Engineering, Taipei, Taiwan, 1995

[3] 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.

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

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

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

[7] 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.

[8] 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.

[9] L. Cherkasova. Improving www proxies performance with greedy-dual-size-frequency caching policy. In HP Technical Report, Palo Alto, November 1998.

[10] P. Cao, E. W. Felten, A. R. Karlin, and K. 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.

[11] 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, Malaysia, June 1997.

[12] D. Duchamp. Prefetching hyperlinks. In Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS '99), Boulder, CO, October 1999.

[13] Z. Su, Q. Yang, Y. Lu, and H. Zhang. Whatnext: A prediction system for web requests using n-gram sequence models. In Proceedings of the First International Conference on Web Information System and Engineering Conference, pages 200-207, Hong Kong, June 2000.

[14] V. Padmanabhan and J. Mogul. Using predictive prefetching to improve world of the Seventeenth International Conference on very Large Database, pages 255-264, September 1991.

[15] E. Cohen, B. Krishnamurthy, and J. Rexford. Evaluating server-assisted cache replacement in the web. In

Proceedings of European Symposium on Algorithms, August 1998

[16] 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.