Asian Journal of Information Technology 7 (3): 100-108, 2008
ISSN: 1682-3915

Improving Proxy Cache Performance by Domain-Behavior Classification and On-Demand Caching

[1]Ray-I Chang, [2]Yen-Liang Chen and [2]Yu-Ying Wu
[1]Department of Engineering Science, National Taiwan University, Taipei, Taiwan
[2]Department of Information Management, National Central University, ChungLi, Taiwan

Abstract: To increase the hit ratio on proxy and reduce the access latency of clients, different proxy caching techniques are designed to predict clients' surfing behavior for fetching their request pages ahead. However, these techniques have two deficiencies. First, the prediction is based on the history of clients' references, but these historical data are not always credible due to the variability of clients' behavior. Second, although these techniquszxes can achieve high hit ratios on proxy, their total traffic loads on network are high. In this study, we propose a novel predictive caching method called ODBC (On-demand Domain-Behavior Classification). The ODBC method first follows Pareto's 80/20 law to clean data. It applies the concepts of entropy and sliding window to identify the exploratory requests and removes them when making predictions. Then, it tags popular pages and let those pages stay in the proxy longer than the normal ones. Experiments on real traces show that ODBC can improve not only the proxy hit ratio but also the network traffic loads.

Key words: Proxy caching, client behavior classification, domain-favorite, on-demand caching

INTRODUCTION

Due to the exponential growth of network traffic, users bear the hassle of waiting for the requested Web pages to come up. How to speed up Web surfing has become one of the hottest topics in Internet research (Wang, 1999; Barish and Obraczka, 2000). Presently, the best known solution is to cache pages at proxy (Cao and Irani, 1997; Aggarwal *et al*., 1999; Shim *et al*., 1999) (or other network nodes, i.e., client (Kim and Kim, 2003; Xu and Ibrahim, 2004) and server (Padmanabhan and Mogul, 1996; Cuhna and Jaccoud, 1997; Markatos and Chronaki, 1998). An effective proxy needs to reduce not only the server load, but also the network overhead for increasing the system's availability and dependability. However, conventional proxy caching methods usually capitalize on temporal locality (Jin and Bestavros, 2000). They cache pages which are just requested. As the trend of client behavior is not considered, the benefits are limited (Abrams *et al*., 1995; Kroeger *et al*., 1997). To resolve this drawback, some proxy caching techniques are investigated to capitalize the reference locality (Almida *et al*., 1996). They predict clients' future behavior for prefetching Web pages into the cache before requested (Nonopoulos *et al*., 2003). Generally, these methods can be classified into two categories. The first one is the informed approach which asks clients to disclose their future requests for caching (Patterson *et al*., 1995). The other one is the predictive approach which uses the history of clients' references to make predictions (Padmanabhan and Mougul, 1996; Markatos and Chronaki, 1998; Nanopoulos *et al*., 2003). The informed approach is inapplicable for normal clients as they will not know their future request pages in advance. Therefore, we focus on the predictive approach in this study (Our proposed method can be easily extended to support the informed approach if users can provide their future requests ahead). As we know, users usually request pages from their favorite domains (e.g., the Website or domain of CNN is usually bookmarked as the "favorites" in browsing). Based on this domain-favorite phenomenon, Shin *et al*. (2000) proposed the domain-top method for proxy caching. They first identify the popular domains which are visited frequently in the historical traces of clients' requests. In each popular domain, a constant number of frequently-visited pages were determined as the popular pages. Then, these popular domains and corresponding popular pages are applied to make predictions. If one of the popular pages is requested, all the related popular pages in the same domain are gotten into the cache before requested. Otherwise, a common caching mechanism is applied. When the cache space is full, the *LRU* (Least Recently Used) replacement policy is applied to replace the cached pages.

Corresponding Author: Yu-Ying Wu, Department of Information Management, National Central University, ChungLi, Taiwan

The domain-top method is based on the hypothesis the surfing behavior is domain-preferential in the entire trace. Results in Shin *et al.* (2000) show that this method can raise the hit ratios with significant improvements in their LAN environment. However, this assumption is not always true as people may explore the Internet aimlessly. In some time periods of the trace, there may have no relation among the pages accessed. Since, the domain-top method does not consider clients' exploratory behavior, it may acquire wrong predictions. Moreover, although this method can achieve a high hit ratio on proxy, the total traffic load on network is also high. It may burst network traffic and increase average queue sizes in network switches (Kleinrock, 1975; Crovella and Barford, 1998). These drawbacks motivate us to design a new proxy caching method to remove the incorrect information in making predictions. Moreover, we will try to reduce not only the server load, but also the network overhead for increasing the system's availability and dependability.

In this study, we propose the ODBC (On-demand Domain-Behavior Classification) method for proxy caching. For improving the hit ratio, ODBC tries to make the training data more reliable. Following Pareto's 80/20 law (Pareto, 1935), ODBC uses the concepts of entropy and sliding window to identify and remove the exploratory requests. Additionally, ODBC reflects domains' popularities to assign each domain a suitable number of popular pages. To reduce the network overhead, ODBC tags these popular pages and keeps them in proxy longer than the normal ones for replacement. Experiments show that ODBC can not only improve the hit ratio but also reduce the network traffic effectively. We summarize the main contributions of ODBC in the following:

Using the sliding window to model individual clients in the entire trace: Conventional approaches use the requests with the same IP address to model a client (Shin *et al.*, 2000). It is not proper for modern networks with DHCP (Dynamic Host Configuration Protocol), where an IP address is shared by different clients and a client has more than one IP address. As requests in the same time period with the same IP address usually belong to the same client, we apply a time sliding window to these requests to make a more accurate model of individual clients.

Using the entropy to discriminate the domain mode client behavior from the exploratory mode one: Conventional approaches use the entire trace to make predictions. However, some clients may explore the Web pages aimlessly. Using these *exploratory* access patterns in prediction may lead to wrong actions. In this study, we calculate the entropy value of requests in each time window to identify the exploratory behavior. By removing these unreliable access patterns, we can make a more accurate prediction.

Assigning the suitable number of popular pages to reflect the popularity of each domain: Conventional approaches assign a constant number of popular pages to each popular domain for caching. However, different domains usually have different popularities. In this study, to reflect the popularities of different domains, suitable no. of popular pages are assigned to improve the hit ratio.

Alleviating the network traffic by the on-demand method: Conventional approaches are designed to increase the hit ratio. They pay less attention to reduce the network traffic. To avoid the unnecessary network access, we apply the on-demand method in caching. It tags the predicted popular pages but fetches them only when they are on-demand. Then, the tagged pages can stay longer in the proxy and thus improve not only the hit ratio but also the network traffic.

Related works: The predictive caching problem (which proactively preloads data from the server into the cache to facilitate future requests) has been studied over many years (Padmanabhan and Mougul, 1996; Cunha and Jaccoud, 1997; Markatos and Chronaki, 1998; Nanopoulos *et al.*, 2003; Shin *et al.*, 2000; Chinen and Yamaguchi, 1997; Chen and Zhang, 2003; Xu *et al.*, 2004). Based on a predictive file caching method proposed in Griffioen and Appleton (1994), Padmanabhan and Mogul (1996) presented a Web predictive caching algorithm to reduce the access latency. They represented clients' requests by a weighted URL (Uniform Resource Locator) dependency graph. In the graph, each edge illustrated a relationship between a pair of URLs. The edge weight depicted the transfer probability from one URL to the other. When a request came, the graph was updated dynamically to predict the future request.

As the graph structure was complicated in implementation, Schechter *et al.* (1998) used a sequence prefix tree instead. The next request was predicted by the longest MFS (most-frequent sequence) matched. On the other hand, Sarukkai (2000) used the Markov chain to propose a probabilistic sequence generation model by the history of requests from the same client. When receiving a client request, a probabilistic link prediction was made. To further improve the accuracy of prediction, the data structure was designed to be updated for each coming request. Notably, as the above methods are all based on complex probability theory and data structure, they need

large storage space and high computation complexity to maintain the access relation of each client. They are difficult in implementation and have never been applied in any real world proxy.

Generally, a user usually access more pages in his favorite website (called *domain* in this study). For example, users who go to browse one page in the NBA website always request more (popular) pages of that website. This phenomenon is called *domain-favorite*. In this case, if the proxy has prefetched the popular pages from that website, access latency will be extremely reduced. Based on this idea, Markatos and Chronaki (1998) propose a simple top-10 technique in predictive caching. Different from the present proxy caching methods, the top-10 method is *server-initiated*. Inputting the history logs, each server (website) will measure its access frequencies to maintain a list of the 10 most popular pages on the server. The servers need to periodically calculate and push their popular pages to proxy. Unfortunately, modern websites are not designed to support the calculation of any list for popular pages.

Wong and Yeung (2001) propose the site-based mechanism to deal with the domain-favorite phenomenon on clients. As a *client-initiated* method, the site-based mechanism collects clients' requests by using a modified browser to periodically calculate a list of the most frequently requested websites (called hot-sites). Based on this list, browsers forward the requests in hot-sites to the proxy for caching. Other requests will be bypassed the proxy and be directly sent to the corresponding websites. Notably, different from the Top-10 method, only the hot-sites will use the proxy and there is no popular pages calculated. The site-based mechanism doesn't need to redesign the website or the proxy. However, till now, there is no existing browser has supported this mechanism.

Recently, a proxy-initiated predictive caching method (Shin *et al*., 2000) called DT (domain-top) is introduced. The method bases on the domain-favorite concept to identify the most popular domains (called top-domains). Then, a constant number of popular pages (called top-pages) are assigned to each top-domain. It makes a prediction table. When a client request is received, the proxy will simply check whether the request is in the prediction table or not. If the answer is yes, DT will identify the top-domain related to this request and fetch all top-pages in this top-domain into the proxy. Otherwise, the common proxy caching mechanism is applied.

Notably, previous approaches usually make prediction based on the entire trace without data clearing. Moreover, they usually focus on increasing the hit ratio, but pay less attention to reduce the network traffic. This may cause burst traffic on networks. We need a more accurate method to model clients' behaviors. In this study, a novel proxy caching method is proposed to resolve these problems. The proposed method can not only improve the hit ratio but also alleviate the traffic load effectively.

MATERIALS AND METHODS

The ODBC predictive caching method consists of 2 phases: prediction and caching. The prediction phase is an off-line process that takes place during the off-peak time of the proxy server. On the other hand, the caching phase is an on-line process that works when a request comes. Figure 1 shows the system architecture of the proposed method. A log file contains a sequence of historical traces of clients' requests. They are frequently used to predict the future behaviors of the clients.

Preprocessing: Typically, a proxy server log contains millions of records. Before making predictions, a preprocessing module will filter out the unnecessary data of the log file. Table 1 shows an example of the log file, where each record refers to a visit by a user to request a certain web page served by a web server. Generally, the requests can be divided into two categories: static pages and dynamic pages. Dynamic pages are created by Web servers when the related contents are requested (*e.g.*, .cgi, .asp and .php scripts). They are changed with time and not suitable to be cached. Therefore, as previous approaches, we filter out logs for this kind of requests before modeling the prediction. In the following steps, only the static pages are focused. Other cached objects will apply the common proxy mechanism in caching.

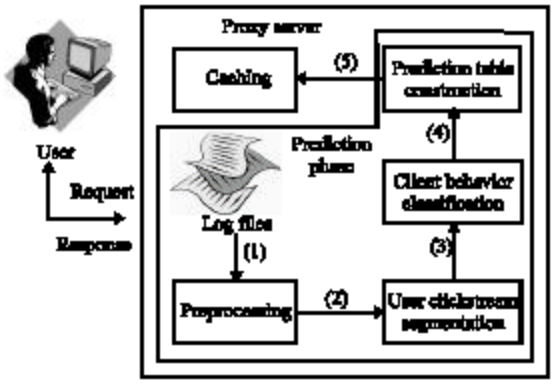

Fig. 1: Sydtem architecture of the proposed ODBC (On-demand domain behaveior classification) methods

Table 1: A simple example of the text data recorded in a proxy log file

1076121059.897 724 61.70.236.89 TCP_MISS/204 257 GET http://www.google.com.tw/url? - DIRECT/66.102.11.99 text/html…
1075967079.162 12 203.68.42.33 TCP_IMS_HIT/304 253 GET http://www.chiark.greenend.org.uk/~sgtatham/putty/ sitestyle.css - NONE/- text/css

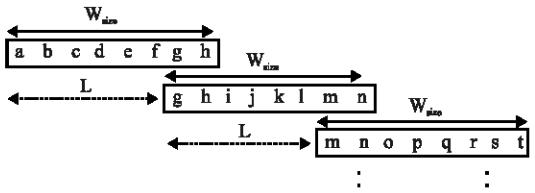

Fig. 2: The sliding window method applied to model individual clients

Table 2: An example of clients' clickstreams

Client	Clickstreams
U_1	CS_{11}, CS_{12}, CS_{13}, CS_{14}, CS_{15}, CS_{16}
U_2	CS_{21}, CS_{22}, CS_{23}
U_3	CS_{31}, CS_{32}, CS_{33}, CS_{34}, CS_{35}

User clickstream segmentation: Notably, a proxy server may serve different clients. We want to model individual users in the input log file to make a more precise prediction. There are several ways to identify individual users (Eirinaki, 2005). The most obvious solution is to assume that each IP address identifies a single user. However, in current networks, a user may access the Web from different IP addresses, or many users may use the same IP address. Conventional approaches that consider only the IP address in user identification are not insufficient.

Our idea is based on an observation that, during a certain time interval, consecutive requests from the same IP address are usually from the same user. This time interval is called a time window (or window for short) in this study. After preprocessing, we first arrange the input data chronologically according to IP addresses. Then, a sliding window technique is applied to segment the consecutive requests into small clickstreams. As shown in Fig. 2, sliding window with size W_{size} and sliding step L ($1 \le L \le W_{size}$) is applied to segment the requests. Our experiments found that $W_{size} = 8$ and $L = 0.8 \times W_{size}$ work well for a wide range of problems. It can improve the hit ratio under adequate computation cost.

Table 2 presents an example of clickstreams where CS_{ij} indicates the jth clickstream of the client with IP address U_i. CS_{ij} contains a sequence of requests $<R_{ij}(1), R_{ij}(2), \ldots, R_{ij}(W_{size})>$. The k_{th} request $R_{ij}(k) = (T_{ij}(k), D_{ij}(k), P_{ij}(k))$ where $T_{ij}(k)$ is the request time, $D_{ij}(k)$ is the request domain and $P_{ij}(k)$ is the request page. Notably, when the next request occurred about a minute later, we may conclude that it is from the same user. However, when the time interval is over 60 min, it's hard to believe that the request is from the same user (Silverston, 2002). In this study, we set the value of ITI (inactivity-time-interval) between two consecutive requests as 30 min. in log analysis (http://www.abacre.com/ala/manual/visits_ hits_ req.htm.). Additionally, if a clickstream contains fewer than W_{size} requests, it will be dropped for the simplification of computation.

Client behavior classification: As reported in Cunha and Jaccoud (1997), there are usually two different kinds of client behaviors. One is the surfing behavior where the client is interested in exploring different pages and domains. The other is the conservative behavior where the client gets used to browsing certain pages and domains. Usually, proxy's hit ratio is improved if its future requests are predictable. According to Pareto's 80/20 law, 20% of the customers will account for 80% of the purchases (Schmittlein *et al*., 1993). Therefore, conservatory requests are more favorable for caching. In this study, we try to identify and remove surfing clickstreams by the concept of entropy (Shannon, 1948) - a thermodynamic quantity describing the amount of disorder in the clickstream.

Assume that n is the number of domains in clickstream j. Let p_i be the probability to request pages on domain i. For each clickstream, we can calculate its normalized entropy (Anishchenko *et al*., 2001) as follows.

$$E(j) = -\sum_{i=1}^{n} p_i \times \log(p_i)/\log(n) \quad (1)$$

E (j) can be viewed as the degree of uncertainty in the clickstream j regarding its domain-favorite behavior. Therefore, if E (j) is smaller than the assigned threshold value E_{max}, the clickstream j is a domain-mode clickstream. Otherwise, we believe the clickstream is in the exploratory mode and should be ignored while constructing the prediction table. A more detailed algorithm that applies behavior classification to clickstream segmentation is shown as follows:

```
procedure clickstream_segmentation (R)
// Let R be the set of requests with a certain IP in the log file.
// |R| is the number of requests
// W_size is the sliding window size
// E_max is the threshold of entropy
// R (j).time is the time of the j^th request of R
// recordList is a clickstream <R (i), …, R (j)>
// candidateList is a list of consecutive requests
{
```

```
for (i = 1; i <= |R|; i++) {
   overlapFlag = 1; // overlapFlag control the sliding step
   recordList = <R (i)>;
    for (j = i+1; j < i+W_size && j <= |R|; j++) {
        if (R (j).time - R (j-1).time > ITI) exit inner for loop;
        add R (j) to recordList;
    }
    if (j < i+W_size) overlapFlag = 0;
    else  {
          csEntropy = calculate_entropy (recordList);  //
          see Eq. (1)
          if (csEntropy <= E_max) add recordList to
          candidateList;
    }
    i = j - 1;
    if (overlapFlag == 1)
          i = i – (W_size  - L);  // sliding L requests for the
          next window
 }
} //end procedure
```

Notably, previous methods can be regarded as a special case of our classification method with a large E_{max}. They would not filter out the exploratory clickstreams.

Prediction table construction: Given a set of domain-mode clickstreams, we cumulate the number of requests m_i for each domain i as its popularity. The most popular dn_{size} domains are then selected as Top-Domains. For each Top-Domain i, a suitable number of popular pages (called Top-Pages) are picked. Notably, in the previous approaches, the number of Top-Pages is a constant where pg (i) = pg_{size} for all Top-Domains i. However, domains may have different popularities. Different numbers of Top-Pages should be assigned to reflect their popularities. As shown in Fig. 3, the prediction table is a two-dimension list called Top-List.

To make a fair comparison, we assign a constant PG = $dn_{size} \times pg_{size}$ of pages in different methods. For each Top-Domain i, the number of Top-Pages pg (i) can be calculated using the following equation.

$$pg(i) = pg_{min} \times (1 + dn_{size} \times (m_i / \sum_{x=1}^{dnsize} m_x)) \quad (2)$$

The minimum number of Top-Pages is defined as $pg_{min} = pg_{size}/2$. When the Top-List is produced, it can be stored in the memory with a hash table to search Top-Pages in O (1) (Baboescu, 2001). In this study, the time period to rebuild the Top-List is 24 h (one day) as suggested by Shin *et al.* (2000) and Chen Zhang (2003). The procedure will be scheduled at 5:00 AM in the morning as there are few requests to the server.

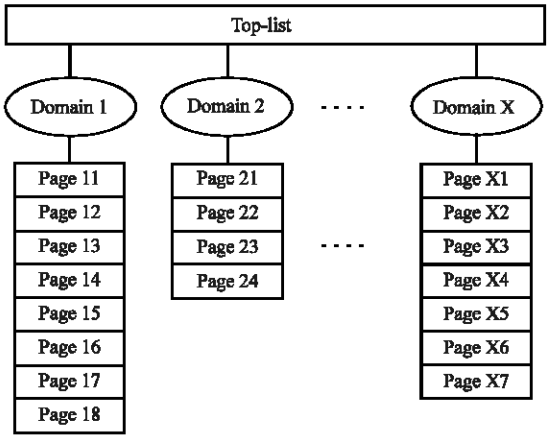

Fig. 3: The structure of top-list where the index X = dn_{size} (the number of top-domain)

Caching: Different from previous approaches, for each coming request, ODBC would not impetuously fetch corresponding Top-Pages. On the other hand, the pages are tagged and fetched only when they are demanded. When the proxy receives a request, the proposed caching method first checks whether the page is in the cache or not. If the page is cached (no matter if it is in the Top-List or not), we will assign a new access time to it. If the requested page is not in the cache, ODBC will fetch the page, store a copy of the page into the cache and then examine whether the page is in the Top-List or not. If the page is in the Top-List, a large cache life time (denoted by RefCount) will be assigned. Otherwise, the default cache life time is assigned. The detailed ODBC algorithm is listed as follows:

```
procedure ODBC_caching (p)
// p is the requested page
{
    if (p is in the cache) {
          reply p;
          refresh the access time of p;
    }
    else {
          fetch p from the original server;
          if (p is in the Top-List)
            assign a larger RefCount value to p;
          else
            assign a normal RefCount value (e.g., 1) to p;
          store a copy of p into cache;
          reply p;
    }
} //end procedure
```

When the disk space is full, the original *LRU* cache replacement policy is adopted. The least recently used page in the cache is selected to subtract its *RefCount* by 1. When the *RefCount* of page equals zero, it will be purged. If the page is a Top-Page and the *RefCount* is not equal to zero, ODBC will refresh its access time. Therefore, the hot pages in the Top-Domains will be cached in proxy longer than the normal ones with the LRU policy.

RESULTS AND DISCUSSION

In this study, we use the trace-drive simulation with a real dataset to examine the performance obtained by different algorithms. The traces (log files) are gathered from a proxy server (http://proxy.ncu.edu.tw:3128) in the computer center of National Central University, Taiwan, from May 10 to May 29. In these log files, each record refers to a request from a client to a server. It provides the time, the URL and the request's Web server and the user's IP address. Table 3 shows the number of clients and the number of requests for each date in the traces. In this study, we use two different metrics (hit ratio and traffic load) to evaluate caching performances. These metrics can be formally computed by the following Equations (Markatos and Chronaki, 1998; Chen and Zhang, 2003; Hu *et al*., 2003).

$$\text{Hit Ratio} = \text{Hit}_{REQ} / \text{Request}_{TOT} \quad (3)$$

$$\text{Traffic Load} = \text{Size}_{TOT.} \quad (4)$$

The variable Hit_{REQ} is the hit numbers of request pages in the cache. $Requests_{TOT}$ and $Size_{TOT}$ are the total request number and the total request size, respectively. Usually, the client latency is low if the hit ratio of proxy is high.

Before comparing our proposed ODBC with other proxy caching schemes, we use several log files to analyze the problem parameters. For fair comparisons, we set the number of Top-Domains as 20 and keep the cache size equal to 100 MB in all experiments. The number of Top-Pages is 160. Figure 4 shows the hit ratio obtained by ODBC due to different threshold values of normalized entropy. In this study, the normalized entropy is applied to indicate the messy degree of user navigations for filtering out the exploratory requests. A lower threshold may eliminate creditable requests. On the other hand, a higher threshold may leave the exploratory requests. As shown in Fig. 4, a steep rise of hit ratio happens at beginning and the hit ratio slowly drops after the threshold 0.6. We set the threshold value as 0.6 in our experiments because it is a reasonable middle value and can yield the highest hit ratio.

Figure 5 shows the hit ratio obtained by ODBC with different sliding window sizes. The hit ratio rises radically when the window size is less than 3. It rises gently when the window size is increased from 3-8. However, the hit ratio decreases as the window size exceeds 8. The possible reason is that a window with less than 3 requests would not provide enough information to classify user behavior. Additionally, a user usually browses one domain with averagely 8 consecutive requests. Therefore, based on the pattern of user behaviors, our let the sliding window size be 8.

In ODBC, *RefCount* is a key value as regards to the effect. Figure 6 plots the hit ratio and the traffic load due to different *RefCount*. In general, the hit ratio increases with *RefCount*, whereas the traffic load decreases. If the *RefCount* is assigned to 1, the Top-Pages are treated as general pages. ODBC is equal to the LRU replacement policy. It has the minimum cache hit ratio and the maximum traffic load. When the value of *RefCount* is equal to 15, it attains the maximum hit ratio and the minimum traffic load. Our choice in terms of *RefCount* is 15.

Since, the cache size is limited, it is impossible to store all the requested pages. By the way, as many studies have found, the popularity of requests follows the

Table 3: The statistic information of proxy traces obtained from May 10 to May 29

Date	**05/10 (Mon.)**	**05/11 (Tue.)**	**05/12 (Wed.)**	**05/13 (Thu.)**	**05/14 (Fri.)**
Clients	2468	2406	2429	2317	2101
Requests	2181588	2099091	1740685	1959941	1625226
Date	**05/15 (Sat.)**	**05/16 (Sun.)**	**05/17 (Mon.)**	**05/18 (Tue.)**	**05/19 (Wed.)**
Clients	1259	1383	2210	2234	2260
Requests	1182322	1020683	1885906	1862264	2034726
Date	**05/20 (Thu.)**	**05/21 (Fri.)**	**05/22 (Sat.)**	**05/23 (Sun.)**	**05/24 (Mon.)**
Clients	2291	2072	1179	1007	1531
Requests	1940849	1159441	350383	721406	624886
Date	**05/25 (Tue.)**	**05/26 (Wed.)**	**05/27 (Thu.)**	**05/28 (Fri.)**	**05/29 (Sat.)**
Clients	1363	1505	1609	1579	946
Requests	875999	1203659	1291234	1116384	625507

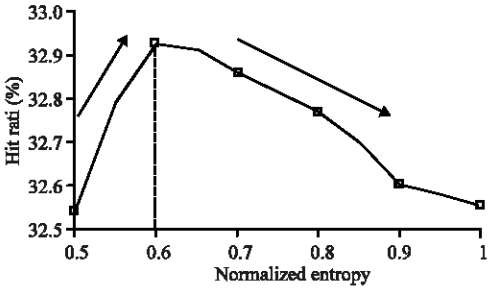

Fig. 4: The hit ratio obtained by ODBC due to different threshold values of normalized entropy

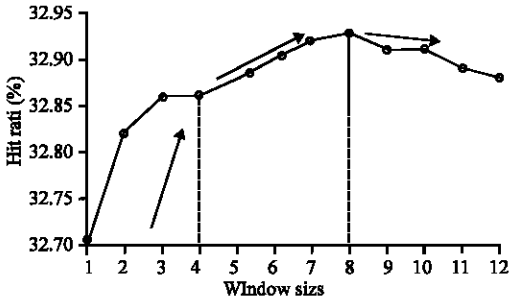

Fig. 5: The hit ratio obtained by ODBC due to different window size

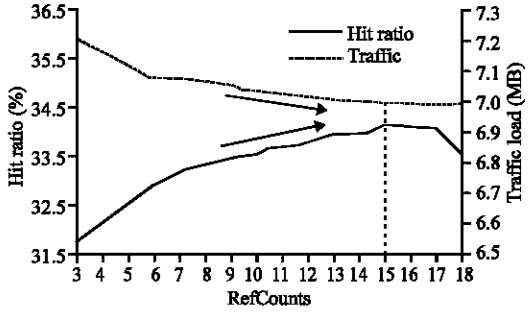

Fig. 6: The change of hit ratio and traffic load as the *RefCount* varies

Zipf distribution (Cunha *et al*., 1995; Lee *et al*., 1999). To satisfy a large fraction of users' requests, the cache need only store the pages those will be requested most-frequently in the future. In our experiments, we evaluate the benefits of the ODBC method and compare it with DT and LRU (the default cache replacement policy of Squid). Figure 7 presents the hit ratio obtained by different cache sizes. As ODBC has taken the clients' domain behavior into consideration and can remove exploratory requests in making prediction, its hit ratio outperforms those of LRU and DT for all cache sizes.

Notably, when using prefetching for improving performance, all previous approaches need multiple

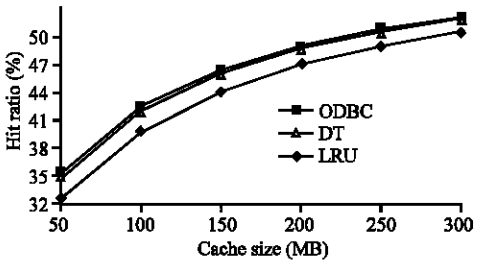

Fig. 7: The hit ratio obtained by different proxy caching methods due to different cache sizes

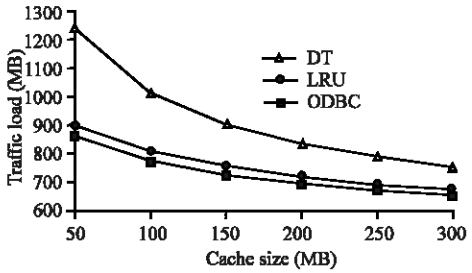

Fig. 8: The network traffic load required by different proxy caching methods due to different cache sizes

requests to the server. It causes the network traffic to increase. In DT, popular pages are prefetched in advance. It may raise the hit ratio but leads to additional bandwidth due to errors in prediction. Figure 8 depicts that DT causes higher traffic load than ODBC for all different cache sizes. It is important to note that the traffic load of ODBC is also lower than the LRU policy. The results demonstrate that ODBC is evidently more effective. Notably, when the cache size is large, more pages are cached and more future requests may be satisfied by the proxy. For a very large cache size, the performance of all methods converges since almost all the pages are cached.

CONCLUSION

In this study, we propose the ODBC method that takes client's domain-favorite behavior into account and uses the normalized entropy of sliding window to remove incredible requests. Moreover, we address the on-demand concept to further reduce the traffic load. Experiments show that ODBC can not only improve the hit ratio, but also alleviate the network traffic. As the proposed algorithms are very simple in implementation, we have developed an optional ODBC module in the Squid proxy server (Wu *et al*., 2004). We plan to explore more on the relationship between caching prediction and clients' behavior in our future works. For example, patterns of

users' behavior may be deeply and meticulously segmented by the data mining technique. We also intend to extend our method from isolation to cooperation. The requests from homogeneous users can be conducted to the same proxy server by clustering techniques.

REFERENCES

Abrams, M., C.R. Standridge, G. Abdulla, S. Williams and E.A. Fox, 1995. Caching proxies: Limitations and potentials. In: Proc. 4th Int. World Wide Web Conf., pp: 119-133.

Aggarwal, C., J. Wolf and P.S. Yu, 1999. Caching on the world wide web. IEEE Trans. Knowledge Data Eng., 11 (1): 94-107.

Almeida, V., A. Bestavros and M. Crovella, 1996. Oliveira Ad. Characterizing reference locality in the WWW. Proc. IEEE Conf. Parallel and Distributed Inform. Sys. (IEEE PDIS), pp: 92-103.

Anishchenko, T., N. Igosheva, T. Yakusheva, O. Glushkovskaya-Semyachkina and O. Khokhlova, 2001. Normalized entropy applied to the analysis of interindividual and gender-related differences in the cardiovascular effects of stress. Eur. J. Applied Physiol., 85 (3-4): 287-298.

Baboescu, F., 2001. Proxy caching with hash functions.

Barish, G. and K. Obraczka, 2000. World wide web caching: Trends and techniques. IEEE. Commun. Mag., pp: 178-184.

Cao, P. and S. Irani, 1997. Cost-aware WWW proxy caching algorithms. Proc. 1997 USENIX Symp. Internet Technol. Sys. (USITS), pp: 193-206.

Chen, X. and X. Zhang, 2003. A popularity-based prediction model for web prefetching. IEEE. Comput., 36 (3): 63-70.

Chinen, K. and S. Yamaguchi, 1997. An interactive prefetching proxy server for improvement of WWW latency. In: Proceeding of the 7th Annual Conference Internet Society (INET), Kuala Lumpur.

Crovella, M. and P. Barford, 1998. The network effects of prefetching. Proc. IEEE. INFOCOM, San Francisco, CA, pp: 1232-1239.

Cunha, C.R. and C.F.B. Jaccoud, 1995. M.EC. Characteristics of WWW client based traces: Boston University, Computer Science Department.

Cunha, C.R. and C.F.B. Jaccoud, 1997. Determining WWW user's next access and its application to prefetching. In: Proc. Sec. IEEE Symp. Comput. Commun. (ISCC), pp: 6-11.

Eirinaki, M., 2005. Web mining: A roadmap.

Griffioen, J. and R. Appleton, 1994. Reducing file system latency using a predictive approach. Proc. Summer USENIX Conf., pp: 197-207.

Hu, T.C., Y. Ikeda, M. Nakazawa and S. Hattori, 2003. Total cost-aware proxy caching with cooperative removal policy. IEICE Trans. Commun., E86-B (10): 3050-3062.

Jin, S. and A. Bestavros, 2000. Sources and characteristics of web temporal locality. Proc. IEEE/ACM Symp. Modeling, Analysis and Simulation of Computer and Telecommun. Sys. (MASCOTS), pp: 28-35.

Kim, Y. and J. Kim, 2003. Web prefetching using display-based prediction. Proc. IEEE/WIC International Conf. Web Intelligence (WI), pp: 486-489.

Kleinrock, L., 1975. Queueing Systems, Theory: John Wiley and Sons, Vol. 1.

Kroeger, T.M., D.D.E. Long and J.C. Mogul, 1997. Exploring the bounds of web latency reduction from caching and prefetching. Proc. USENIX Symp. Internet Technol. Sys., pp: 13-22.

Lee, B., C. Pei, F. Li, P. Graham and S. Scott, 1999. Web caching and zipf-like distributions: Evidence and implications. Proc. INFOCOM, pp: 126-134.

Markatos, E.P. and C.E. Chronaki, 1998. A top-10 approach to prefetching on the web. Proc. INET, pp: 276-290.

Nanopoulos, A., D. Katsaros and Y. Manolopoulos, 2003. A data mining algorithm for generalized web prefetching. IEEE. Trans. Knowledge and Data Eng., 15 (5): 1155-1169.

Padmanabhan, V. and J. Mogul, 1996. Using predictive prefetching to improve world wide web latency. Comput. Comm. Rev., 26 (3): 22-36.

Pareto, V., 1935. The Mind and Society: Non-Logical Conduct. London: Jonathan Cape.

Patterson, H., G. Gibson, E. Ginting, D. Stodolsky and J. Zelenka, 1995. Informed prefetching and caching. Proc. ACM Symp. Operating Sys. Principles (ACMSOSP), pp: 79-95.

Sarukkai, R., 2000. Link prediction and path analysis using markov chains. Computer Networks, 33 (1-6): 377-386.

Schechter, S., M. Krishnan and M.D. Smith, 1998. Using path profiles to predict HTTP request. Seventh International World Wide Web Conf., pp: 457-467.

Schmittlein, D.C., L.G. Cooper and D.G. Morrison, 1993. Truth in concentration in the land of (80/20) Laws. Marketing Science, 12 (2): 167-183.

Shannon, C.E., 1948. A mathematical theory of communication. Bell Sys. Tech. J., 27:379-423. 623-656.

Shim, J., P. Scheuermann and R. Vingralek, 1999. Proxy cache algorithms: Design, implementation and performance. IEEE Trans. Data Eng., 11 (4): 549-562.

Shin, S.W., B.H. Seong and D. Park, 2000. Improving world-wide-web performance using domain-top approach to prefetching. 4th Int. Conf. High-Performance Computing in the Asia-Pacific Region, pp: 738-746.

Silverston, L., 2002. Universal data models for clickstream analysis. DM Review Magazine.

Wang, J.A., 1999. Survey of web caching methods for the Internet. Comput. Commun. Rev., 29 (5): 36-46.

What is Visits, 2007. hits, requests? http://www.abacre.com/ala/manual/visits_hits_req.htm.

Wong, K.Y., K.H. Yeung, 2001. Site-based approach to web cache design. IEEE. Internet Comput., 5: 28-34.

Wu, Y.Y., R.I. Chang, T.C. Pai and J.M. Ho, 2004. Improving squid proxy server. Proc. Taiwan Acad. Network Conf. (TANet), pp: 328-333.

Xu, J., J. Liu, B. Li and X. Jia, 2004. Caching and prefetching for web content distribution. IEEE. Comput. Sci. Eng. Special Issue Web Eng., 6 (4): 54-59.

Xu, X. and T.I. Ibrahim, 2004. A keyword-based semantic prefetching approach in Internet news services. IEEE. Trans. Knowledge Data Eng., 16 (5): 601-611.