An On-Line Page-Structure Approximation Scheme for Web Proxies

Jiunn-Jye Lee, Pan-Lung Tsai, and Chin-Laung Lei
Department of Electrical Engineering, National Taiwan University
{jye, charles}@fractal.ee.ntu.edu.tw, lei@cc.ee.ntu.edu.tw

Abstract

To render a Web page, a browser must first download an HTML document, parse it, and then issue a sequence of additional requests to fetch the embedded objects according to the content of the HTML document. Therefore, it should be straightforward for Web proxies to accurately predict future client requests by considering the characteristics of such regular behavior. However, the strong bindings between embedded objects and their containing documents are often ignored by modern Web proxies because there still exists no efficient solution for Web proxies to obtain the knowledge of page structures without performing the computation-intensive operations of HTML parsing. In this paper, we propose an effective and low-overhead scheme for Web proxies to approximate page structures and refine the approximation as new client requests arrive. The results of simulation show that the approximation converges quickly and reaches high accuracy after a relatively small number of incoming requests have been processed.

Keywords: Page structure, embedded object, Web proxy, lifespan characteristics.

1. Introduction

For many recent Web applications, Web-page structures are highly associated with clients' access sequences. Taking Web browsing, one of the most popular applications as an example, when a user clicks on a link pointing to a Web site, the browser first sends an HTTP request to the server and waits for the response from the server. After retrieving the Web page, the browser analyzes the page content and transmits the requests for the embedded objects required for rendering the page to the server. Thus, for a proxy server located between clients and servers, to predict what are the remaining parts that clients will request for after receiving the first few requests may help a lot in many aspects like caching, prefetching, and content filtering.

It is not difficult for Web servers to obtain the knowledge of the container-embedded object relationships. However, for a proxy server positioned in the middle of many clients and many servers, difficulties arise. First, parsing every incoming Web page is infeasible for a loaded Web proxy. Matching the prefix of each request's URL seems to be a good idea, but the embedded objects may not have the same prefix as the container does, and, furthermore, even merely pattern matching still incurs a heavy burden for proxy servers. Thus, to dig out such information, proxy servers must adopt other methods.

As a normal Web access, the request sequence exhibits spanning characteristics. That is, after parsing the container page and retrieving the URLs of the embedded objects, browsers request the Web servers for these objects. Thus, from the viewpoint of a proxy server, a request span is observed. According to this spanning behavior, we focus on the leaf proxy servers that are deployed closest to the clients and propose a simple and lightweight scheme to approximate the page structures. The results of simulations over real proxy logs show that our scheme can achieve high accuracy with a fast converging speed.

The rest of the paper is organized as follows. Section 2 discusses former researches that have been proposed to improve the performance of Web proxies, and section 3 explains the proposed scheme in detail. Experimental results and analysis of our scheme are described in section 4. Section 5 concludes the paper.

2. Related Works

The primary difficulty for a proxy server to obtain Web page structures comes from the interleaving arrival of client requests. The heavy workload limits the resources a proxy server can use to analyze the Web page. Thus, to design an on-line page-structure approximation scheme, simplicity and low overhead should be taken into consideration.

Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05)
0-7695-2281-5/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

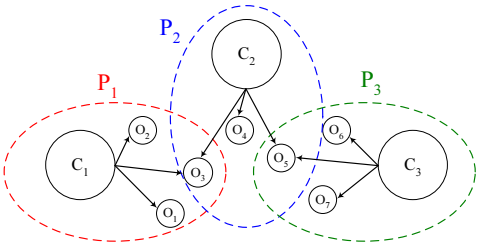


Figure 1. Page structures of page P_1, P_2, and P_3.

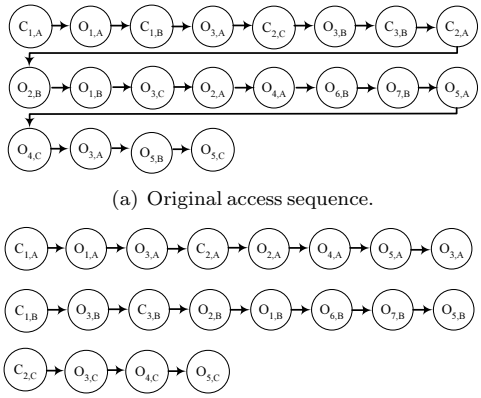
The Markov-chain model [1][2] is famous in predicting Web access behavior. Subsequent researches based on this model such as prediction by partial match (PPM) [3][4], error-pruning methods [5] further improve the prediction accuracy and reduce the space requirement. However, most of these researches focus on the Web-site link prediction and user access patterns, rather than single-page structures.

Many researches also focus on identifying user transactions from Web-proxy servers [6][7]. To construct Web page structures, [8] mines the Web log in an off-line fashion. Owing to the computation cost, these works are not suitable for a real-time deployment.

Thus, based on the observation of lifetime behavior [9] and Web object relationships [10], we propose an effective and low-overhead scheme for Web proxies to approximate page structures in an on-line fashion. The details of the proposed scheme and the simulations are described in the following sections.

3. The Proposed Scheme

From the proxy server's point of view, the arrivals of client requests are interleaved. For example, assume there exist three clients (A, B, and C) requesting Web pages (P_1, P_2, P_3) independently. The page structures are shown in Figure 1, where O_3 is an embedded object of both P_1 and P_2, and O_5 is embedded in both P_2 and P_3, respectively. Figure 2(a) illustrates the arrival sequence of the requests on the proxy server, where $C_{i,j}$ represents the container object C_i requested by the client j and similarly for embedded object $O_{i,j}$. To analyze this sequence, an intuitive thought is to classify the request according to where they come from. Thus, the sequence can be divided into three subsequences, as shown in Figure 2(b). However, it confuses the proxy server when trying to find out the page structures. For example, the server cannot decide whether object $O_{2,B}$ is embedded in container object $C_{3,B}$ or in $C_{1,B}$.



(a) Original access sequence.

(b) Three subsequences divided according to the source addresses of different clients.

Figure 2. An example of subsequence division with respect to client source addresses.

By concentrating on leaf proxy servers that sit right in front of a local area network (LAN), we assume uniform network latency for each client; that is, the lifespan of fetching a complete Web page does not vary a lot. The idea used in the proposed scheme, making use of page structures from chaotic access sequences, is quite simple: *Just include every object seen within the lifespan, and statistics will tell.*

3.1. Notations and Data Structures

As the first step, some parameters and data structures must be defined. They are listed and explained in the following paragraph.

Three kinds of nodes are used in our scheme: Embedded Nodes, Container Nodes, and Queue Nodes. The functionality and the data fields of these nodes are shown in Figure 3.

Embedded Nodes, abbreviated as *ENode*s, are used for storing information of embedded objects. Each *ENode* encapsulates an embedded object's URL (url), timestamp (t), and client IP address (ip).

For each container object, there exists a corresponding Container Node (*CNode*) to retain necessary records. In additional to the same data field as an *ENode*'s, the IP address, and embedded count fields, a *CNode* maintains an access count (ac), a sliding window (sw) for relative accesses, and an Embedded Object Table (EOT). The EOT keeps information of embedded-object candidates of a *CNode*, including references (en) to the related *ENode*s, the corresponding access counts (ac) and sliding windows (sw). How the

IEEE COMPUTER SOCIETY

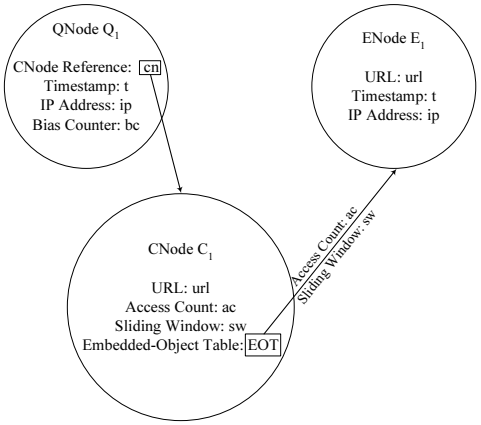


Figure 3. Data structures used in the proposed scheme and their relationships.

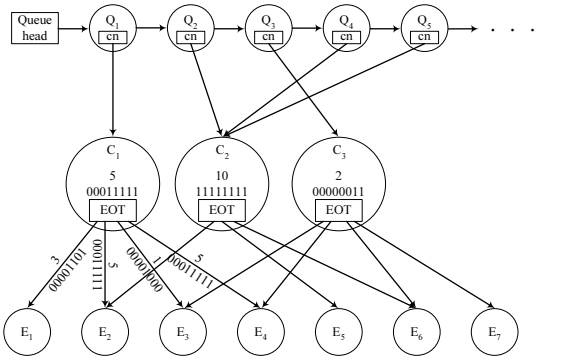


Figure 4. A snapshot of data structures in action.

sliding-window field helps will be explained in the next subsection in detail.

The last type of the nodes, the Queue Nodes (*QNode*s), are used to keep track of active container objects. A container is considered active if the time elapsed after the container objects being requested is smaller than the lifespan (LS). The data fields a *QNode* has to keep include a reference (cn) to the corresponding *CNode*, a timestamp (t) that is equal to the arrival time of the request, the client's IP address (ip), and a bias counter (bc) that cooperates with the sliding window of the *CNode*.

Lifespan(LS) and count threshold(CT) are two more parameters to be described. The former is a predefined span that represents the life time of *QNode*s, and the latter is the lower bound of *CNode*s access counts. Only those *CNode*s with access counts greater than the CT will be analyzed.

3.2. Approximating Page Structures

According to the span characteristic, once the proxy server received a client request with container object type, it is reasonable for the proxy server to expect the occurrence of some requests of the embedded objects in a short time. Thus every request sent by the same client acquiring an embedded-object will be counted as one of the embedded-objects of the container. After the access count of the same container reaches some extent, the objects with total and relative access frequency higher than the predefined thresholds are recognized as the true embedded-objects of the container. In the following sections, two kinds of counters are used: access count(ac) is used to record total access times of an object, and sliding window(sw) is a bit vector representing the latest n accesses of the object.

In our scheme, on receiving a request from the client, the proxy server first identify the requested object type after retrieving the HTTP header of the request. Requests are divided into two groups: container objects (e.g., text/html) and embedded objects (e.g., jpeg).

If the incoming request asked for a container object, the proxy server first looks up the *CNode* list. In the case no information is retained in the list, that is, none of the *CNode*s in the list has the same URL as the requested object, the proxy server setup a *CNode* for this new object. Otherwise, it updates the information of the *CNode* found by increasing the access count ac by one, left shifting the sliding window sw by 1, set the lowest bit of sw to 1, and left shifting the sliding windows of the entries in its EOT by 1. After that, a *QNode* referring to this *CNode* is created and put into the active container queue. Each incoming request of container type will generate a *QNode*, even if there already exists another *QNode* referring to the same *CNode*. In other words, same requests from different clients will be treated as different *QNode*s and the subsequent requests of embedded objects are processed separately. Thus, many *QNode*s may refer to the same *CNode*. Figure 4 illustrates such cases, *QNode*s Q_2, Q_4, and Q_5 with different IP addresses are referring to the same *CNode* C_2.

During the search process, if any *QNode* referring to the same *CNode* is encountered, each *QNode*'s bc is increased by one. This bias counter (bc) field indicates which bit of the sliding window this *QNode* represents and thus, bc is set to 0 initially. Finally, a *QNode* is removed from the queue after its lifespan is passed.

On the other hand, once an embedded object request is arrived, an *ENode* E_i is created if the proxy server can not find any other *ENode* owning the same URL

IEEE COMPUTER SOCIETY

Algorithm 1 Construction of the Embedded-Object Table

```
1: CList ← null; EList ← null; QList ← null;
2: procedure ConstructEOT
3:     for each incoming request object O do
4:         if O ∈ ContainerType then
5:             C_i ← Search(CList, O);
6:             if C_i = null then
7:                 Create(CNode C_i, O);
8:                 Insert(CList, C_i);
9:             else
10:                C_i.ac ← C_i.ac + 1;
11:                LeftShift(C_i.sw, 1);
12:                C_i.sw ← C_i.sw | 1;
13:                for each en_j ∈ C_i.EOT do
14:                    LeftShift(en_j.sw, 1);
15:                    en_j.sw ← en_j.sw | 1;
16:                end for
17:            end if
18:            Create(QNode Q_i, C_i);
19:            Insert(QList, Q_i);
20:            for each Q_j ∈ QList do
21:                if Q_j.cn = Q_i.cn then
22:                    Q_j.bs ← Q_j.bs + 1;
23:                end if
24:            end for
25:        else
26:            E_i ← Search(EList, O);
27:            if E_i = null then
28:                Create(ENode E_i, O);
29:                Insert(EList, E_i);
30:            else
31:                Update(E_i);
32:            end if
33:            for each Q_j ∈ QList&&E_i.t − Q_j.t > LS do
34:                Remove(QList, Q_j);
35:            end for
36:            for each Q_j ∈ QList&&Q_j.IP = E_i.IP do
37:                en_k ← Search(Q_j.cn.EOT, E_i);
38:                if en_k = null then
39:                    Create(EOTEntry en_k, E_i);
40:                    Insert(Q_j.cn.EOT, en_k);
41:                else
42:                    en_k.ac ← en_k.ac + 1;
43:                    temp ← LeftShift(1, Q_j.bc);
44:                    en_k.sw ← en_k.sw | temp;
45:                end if
46:            end for
47:        end if
48:    end for
49: end procedure
```

as the request's. Or the server updates the timestamp and the IP address fields. Afterwards, based on the timestamp t_r of the request, the server removes obsolete *QNode*s(i.e., $\{Q_i | t_r - t_{Q_i} > LS\}$) from the queue.

For each remaining *QNode*, say Q_j, that has the same IP address as E_i's, the proxy server updates the embedded-object table *EOT* of the *CNode* referred by the Q_j's *cn* field. That is, if none of those entries in $Q_j.cn.EOT$ has the *en* field referring to E_i, the proxy server will create a new entry in the *EOT*, set both the *ac* and *sw* fields of the entry to 1. Otherwise, it will increase the entry's *ac* by one and turn the $Q_j.bc$-th lowest bit of the *sw* to 1. If none of the *QNode*s has the same IP as E_i's, then E_i is regarded as a single object not belonging to any container, and thus is dropped.

Algorithm 2 Output of page structures

```
procedure OutputPage(CNode C_i)
    for each en_j ∈ C_i.EOT do
        AR_j ← en_j.ac / C_i.ac;
        RAR_j ← NumOf1s(en_j.sw) / NumOf1s(C_i.sw);
        if en_j.ac > CT && AR_j > AC_RATIO && RAR_j >
            SW_RATIO then
            Output(en_j);
        end if
    end for
end procedure
```

The detailed algorithm is describe in Algorithm 1.

The metrics to determine if a *ENode* is embedded in a *CNode* is quite flexible. One can design his own metrics to fit the system environment. In this paper two kinds of confidence metrics are used: the Access Ratio (AR) and the Relative Access Ratio (RAR). To identify the embedded-objects of a container node C_i with its access count ac larger than the predefined count threshold CT, the server calculates the access ratio and the relative access ratio of each entry en_j in C_i's EOT. The access ratio of the entry en_j (AR_j) is defined as the the ratio of $en_j.ac$ to the $C_i.ac$, and the relative access ratio of en_j (RAR_j) is defined as the ratio of the total number of bits equaling to 1 in $en_j.sw$ to those in $C_i.sw$. If both AR_j and RAR_j exceeds the predefined thresholds AC_RATIO and SW_RATIO, then the embedded node E_k referred by en_j is regarded as one of C_i's embedded object. The decision-making algorithm is illustrated in Algorithm 2.

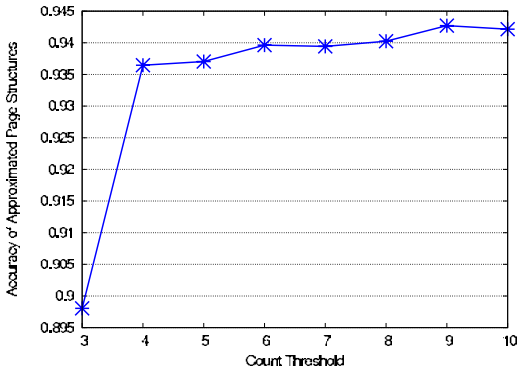
4. Experimental Results

In this section, a series of simulation is conducted based on the proxy log to demonstrate the accuracy of the proposed scheme. The environment consists of a Squid proxy server [11] deployed at the edge of a network containing 16 clients. Each client keeps randomly accessing a Web site without client caches from a group of 272 Web sites whose page structures contain mostly static embedded objects.

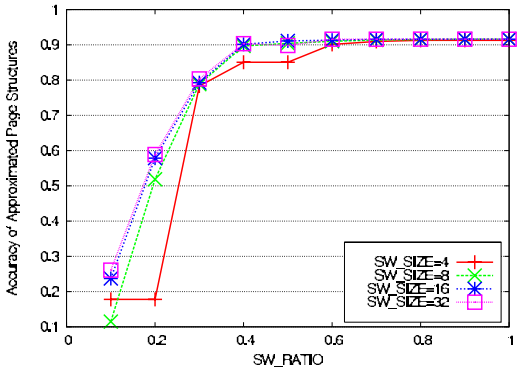
Figure 5(a) shows the effect of count threshold parameter. The lifespan is set to 3 seconds, the AC_RATIO and the SW_RATIO are both set to 0.5, and the size of sliding window is set to 32 bits. The result shows that our scheme achieves high prediction accuracy up to 93.5% with relatively as low a count threshold as 4, which implies the convergence speed is quite fast.

In Figure 5(b), various SW_RATIO is measured with SW_SIZE being 4, 8, 16, and 32 bits, respectively. For a sliding window of size 8, the prediction accuracy exceeds 90% with the SW_RATIO larger than 0.4. The less than 10% misses are mostly due to the dynamic part of the Web pages such as the advertisement banners.

IEEE COMPUTER SOCIETY



(a) Accuracy of approximated page structures vs. count threshold.



(b) Accuracy of approximated page structures vs. SW_RATIO.

Figure 5. Accuracy of approximated page structures under different base-count and SW_RATIO values, respectively.

The length of the lifespan LS parameter is also a dominant factor. For a too short lifespan, some requests for the embedded objects associate with the container object will be dropped. On the other hand, too long a lifespan causes more unrelated objects to be included, and further more lowers down the accuracy. Network latency and the computing power of the client computers are two dominating factors of setting of the lifespan. However, for leaf proxy servers that are set in front of a local area network (LAN), the network latency of each client can be treated as a constant, and moreover, the difference of the computing power on each client is eligible compared to the unit of lifespan. Thus, a not-too-short lifespan is sufficient for our scheme.

5. Conclusions

In this paper, we propose an effective and low-overhead on-line scheme for leaf Web proxies to approximate the page structures of Web pages. Based on the observation of lifespan characteristics and simple counting techniques, the proposed scheme produces approximated page structures with high accuracy once the corresponding numbers of requests for the same Web pages exceed the base-count threshold.

The results of simulation with actual proxy logs show that the chosen base-count threshold can be relatively small. That is, the approximation of page structures converges quickly and reaches high accuracy(more than 90%) after a small number of incoming requests have been processed.

References

[1] Ramesh R. Sarukkai, "Link Prediction and Path Analysis Using Markov Chains," Computer Networks, Vol. 33, No. 1-6, pp. 377-386, June 2000.

[2] Jianhan Zhu, Jun Hong, and John G. Hughes, "Using Markov Models for Web Site Link Prediction," Proceedings of the Thirteenth ACM Conference on Hypertext and Hypermedia 2002, College Park, Maryland, USA, pp. 169-170, June 11-15, 2002.

[3] Themistoklis Palpanas and Alberto Mendelzon, "Web Prefetching Using Partial Match Prediction," Proceedings of the Fourth International Web Caching Workshop, San Diego, CA, March 1999.

[4] Mukund Deshpande and George Karypis, "Selective Markov models for Predicting Web Page Accesses," ACM Transactions on Internet Technology (TOIT), Vol. 4, No. 2, pp. 163-184, May 2004.

[5] Şule Gündüz and M. Tamer Özsu, "A Web Page Prediction Model Based on Click-Stream Tree Representation of User Behavior," Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, D.C, pp. 535-540, August 24-27, 2003.

[6] Wenwu Lou, Guimei Liu, Honjun Lu, and Qiang Yang, "Cut-and-pick transactions for proxy log mining," Proceedings of the Eighth Extending Database Technology (EDBT) Conference, Prague, Czech, March 2002.

[7] Qiang Yang, Haining Henry Zhang, and Tianyi Li, "Mining Web Logs for Prediction Models in WWW Caching and Prefetching," Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, California, pp. 473-478, August 2001.

[8] Xin Chen and Xiaodong Zhang, "Popularity-Based PPM: an Effective Web Prefetching Technique for High Accuracy and Low Storage," Proceedings of 2002 International Conference on Parallel Processing, (ICPP '02), Vancouver, Canada, pp. 296-304, August 18-21, 2002.

[9] Xiangping Chen and Prasant Mohapatra, "Lifetime Behavior and its Impact on Web Caching," Proceedings of the 1999 IEEE Workshop on Internet Applications, pp. 54, July 26-27, 1999.

[10] Mikhail Mikhailov and Craig Wills, "Exploiting Object Relationships for Deterministic Web Object Management," Proceedings of the Seventh International Workshop on Web Content Caching and Distribution, Boulder, Colorado, August 2002.

[11] "Squid Web Proxy Cache," http://www.squid-cache.org/.

IEEE COMPUTER SOCIETY