Integrating Web Caching and Web Prefetching in Client-Side Proxies

Wei-Guang Teng, Cheng-Yue Chang, and Ming-Syan Chen, *Fellow*, *IEEE*

Abstract—Web caching and Web prefetching are two important techniques used to reduce the noticeable response time perceived by users. Note that by integrating Web caching and Web prefetching, these two techniques can complement each other since the Web caching technique exploits the temporal locality, whereas Web prefetching technique utilizes the spatial locality of Web objects. However, without circumspect design, the integration of these two techniques might cause significant performance degradation to each other. In view of this, we propose in this paper an innovative cache replacement algorithm, which not only considers the caching effect in the Web environment, but also evaluates the prefetching rules provided by various prefetching schemes. Specifically, we formulate a normalized profit function to evaluate the profit from caching an object (i.e., either a nonimplied object or an implied object according to some prefetching rule). Based on the normalized profit function devised, we devise an innovative Web cache replacement algorithm, referred to as Algorithm *IWCP* (standing for the *Integration of Web Caching and Prefetching*). Using an event-driven simulation, we evaluate the performance of Algorithm *IWCP* under several circumstances. The experimental results show that Algorithm *IWCP* consistently outperforms the companion schemes in various performance metrics.

Index Terms—Web Proxy, caching, prefetching.

1 Introduction

The growth of the World Wide Web over the past few years has imposed a significant traffic burden upon the Internet. A significant amount of research effort has been elaborated upon investigating effective and scalable solutions to reduce the noticeable response time perceived by users. Among others, Web caching and Web prefetching are two important techniques to this end.

1.1 Web Caching and Prefetching

According to the locations where objects are cached, Web caching technology can be classified into three categories, i.e., client's browser caching, client-side proxy caching, and server-side proxy caching [6]. In client's browser caching, Web objects are cached in the client's local disk. If the user accesses the same object more than once in a short time, the browser can fetch the object directly from the local disk, eliminating the repeated network latency. However, users are likely to access many sites, each for a short period of time. Thus, the hit ratios of per-user caches tend to be low. In client-side proxy caching, objects are cached in the proxy near the clients to avoid repeated round-trip delays between the clients and the origin Web servers. To effectively utilize the limited capacity of the proxy cache, several cache replacement algorithms (e.g., [4], [9], [33], [34], [35]) are proposed to maximize the delay savings obtained from cache hits. Such advanced caching algorithms differ from the conventional ones (e.g., LRU or LFU algorithms) in their consideration of newly identified parameters in the Web environment, i.e., the size, fetching delay, reference rate, invalidation cost, and invalidation frequency of a Web object. Incorporating these parameters into their designs, these cache replacement algorithms show significant performance improvement over the conventional ones. In addition, cooperative caching architectures, proposed in [18], [31], enable the participating proxies to share their cache content with one another. Since each participating proxy can seek for a remote cache hit from other participating proxy's cache, the overall hit ratio can be further improved. On the other hand, server-side Web caching and *content distribution network* (*CDN*) [1], [2], [3] are recently attracting an increasing amount of attention. It is noted that, as the Web traffic grows exponentially, overloaded Web servers become the sources of the prolonged response time. Server-side Web caching, which distributes/routes the users' requests to the proper server-side proxies, is able to release the Web server's load and, thus, to shorten the user perceived response time.

Unlike Web caching which exploits the temporal locality of the Web objects, Web prefetching technique takes advantage of the spatial locality of them. Generally, a user's browsing sequence will follow the hyperlinks between Web objects. That is, if object A has a hyperlink to object B, the probability that B will be accessed, given A has been accessed already, will increase significantly. Hence, if we prefetch those objects which are very likely to be referenced in the client's subsequent requests, part of the network latency can be hidden within the time between client's consecutive requests [8], [11], [12], [13], [17], [19], [22]. From this aspect, it is noted that the performance of a Web prefetching scheme mainly relies on the accuracy of the employed prediction algorithm. Markov model [28] is one useful tool for modeling and predicting a user's browsing behavior over the Web. The prefetching schemes proposed in [16], [27], [32] take advantage of Markov models to derive

• *The authors are with the Department of Electrical Engineering, National Taiwan University, No. 1, Sec. 4, Roosevelt Road, Taipei, Taiwan, R.O.C. E-mail: {eev, cychang}@arbor.ee.ntu.edu.tw, mschen@cc.ee.ntu.edu.tw.*

Manuscript received 5 Sept. 2003; revised 3 June 2004; accepted 28 July 2004; published online 22 Mar. 2005.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0154-0903.

prefetching rules from a Web server's access log. Another emerging tool for predicting users' access patterns is Web mining technique [10]. The prefetching schemes devised in [23], [26], [30] use the Web mining technique to discover frequent user access patterns from the Web server's access log as well. Moreover, the work in [15] proposed an approach of predicting the users' next actions based on the contents of Web pages. In summary, various techniques have been utilized for improving the accuracy of predicting user access patterns, making the prefetching of Web objects more effectively.

Since caching and prefetching are both well recognized for improving user perceived response time, the integration of both strategies has been generally discussed in recent works. In [8], an integrated model for file systems was explored. A simple framework of prefetching referenced pages from hyperlinks embedded in the current object was proposed in [11]. The idea was further employed in [17] by considering the access frequency of the hyperlinks. In addition, Web caching and prefetching are integrated using different prediction models of aggregated user behavior in [21] and [38].

In recent works [24], [25], the caching and prefetching for pages of query results in Web search engines are studied. Unlike the system environment designated in this paper, these works focus on the distribution of query popularity rather than the Web page structures. In addition, unlike most related works which make a prediction based on historical access logs, the work in [15] predicted the user's next action based on the analysis of the content of recent requested pages.

1.2 Motivation

Note that by integrating Web caching and Web prefetching, these two techniques can complement each other since the Web caching technique exploits the temporal locality, whereas the Web prefetching technique utilizes the spatial locality of the Web objects as mentioned above. However, without circumspect design the integration of these two techniques might cause significant performance degradation to each other. For instance, to provide the proxy with rich information, a Web server may deliberately send all possible prefetching hints with various levels of confidences to the proxy. Without any control, a proxy will prefetch every implied object into its cache, despite that the confidences of some prefetching rules may be low. In this case, a significant portion of the cache content will be replaced because a proxy may concurrently serve a large amount of client requests and each of these requests may trigger certain prefetching rules. As a result, the state of the cache content will become unstable and the cache hit ratio will drop sharply. On the contrary, if the prefetching control is over strict, a proxy will tend to discard some beneficial hints provided by the Web server, thus whittling down the advantage of Web prefetching. In view of this, the motivation for our study is to design an innovative cache replacement algorithm, not only considering the caching effect in the Web environment, but also evaluating the prefetching rules provided by various prefetching schemes.

The challenges of devising such a cache replacement algorithm are mainly due to the following phenomena.

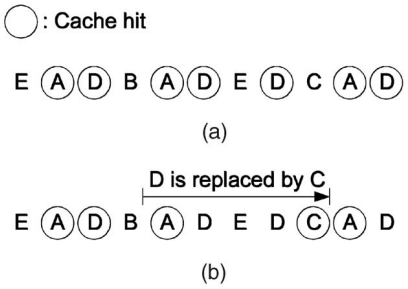


Fig. 1. The execution scenario of Example 2. (a) Without prefetching and (b) with prefetching.

First, the caching parameters of the implied object (i.e., the size, fetching cost, reference rate, invalidation cost, and invalidation frequency of the implied object) can affect the strength of the corresponding prefetching rule. This phenomenon can be illustrated by the following examples.

Example 1. Suppose two prefetching rules, say $(A \longrightarrow B)$ and $(C \longrightarrow D)$, have the prefetching confidences 0.8 and 0.5, respectively. Assume that the fetching delay of object B is 2 and that of object D is 10. The expected delay saving from caching B is then equal to $1.6 = 0.8 * 2$, whereas the expected delay saving from caching D is equal to $5 = 0.5 * 10$. In this case, caching D (i.e., the implied object with lower prefetching confidence) is in fact more beneficial than caching B (i.e., the implied object with higher prefetching confidence).

Example 2. Suppose $\{EADBADEDCAD\}$ is the reference sequence received by the proxy within the time interval w. In this sequence, the numbers of the references to objects A, B, C, D, and E are $3, 1, 1, 4$, and 2, respectively. Assume that the prefetching rule $(B \longrightarrow C)$ holds with the confidence of 100 percent within the time interval w, and the cache size of the proxy is "two" (objects). If we always choose objects A and D to cache (i.e., the two objects with the highest reference rates), we can obtain seven cache hits as shown in Fig. 1a. However, if taking the prefetching rule $(B \longrightarrow C)$ to cache C when the client requests B, one will only achieve five cache hits as shown in Fig. 1b. In this case, although the implied object C has the prefetching confidence of 100 percent, prefetching C contrarily decreases the system performance.

Note that similar situations exist when using other caching parameters. Consequently, not only the confidences of the prefetching rules, but also the caching parameters of the implied object shall we take into consideration when devising the cache replacement algorithm for the integration of Web caching and Web prefetching schemes.

Second, even if two implied objects have the same values of the caching parameters, the one with higher prefetching confidence is not necessarily more beneficial to be prefetched or cached. This phenomenon can be best understood by the two examples in Fig. 2.

Example 3. As shown in Fig. 2a, assume that the client 1 and client 2 have referenced object A, and the client 3 has referenced object E in their precedent requests. According to the confidences of the prefetching rules $(E \xrightarrow{0.8} H)$

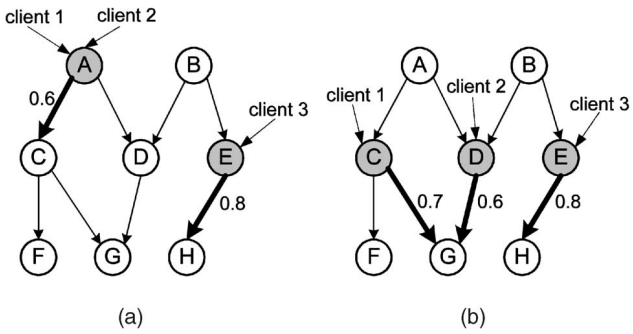


Fig. 2. Execution scenarios of Examples 3 and 4.

and $(A \xrightarrow{0.6} C)$, the probability that C will be referenced at least once in the clients' subsequent requests is, hence, $0.84 = 0.6 + 0.6 - 0.6 * 0.6$ and the probability that H will be referenced at least once in the clients' subsequent requests is 0.8. As a result, although the implied object H has a higher prefetching confidence according to the prefetching rule $(E \xrightarrow{0.8} H)$ than the implied object C according to the prefetching rule $(A \xrightarrow{0.6} C)$, C is more beneficial to be prefetched.

Example 3 illustrated the situation when two clients have referenced the same object (i.e., object A) and triggered the same rule (i.e., $(A \xrightarrow{0.6} C)$). The situation when two prefetching rules are triggered to prefetch the same object is shown in the following example.

Example 4. As shown in Fig. 2b, assume that the client 1 has referenced object C, the client 2 has referenced object D, and the client 3 has referenced object E in their precedent requests. Since two prefetching rules, $(C \longrightarrow G)$ and $(D \longrightarrow G)$, suggest to prefetch the object G with confidences 0.7 and 0.6, respectively, the probability that G will be referenced at least once in the clients' subsequent requests is $0.88 = 0.7 + 0.6 - 0.7 * 0.6$. In this case, caching G is more beneficial even though the confidence of the rule $(E \xrightarrow{0.8} H)$ is higher than the confidences of the rules $(C \xrightarrow{0.7} G)$ and $(D \xrightarrow{0.6} G)$, respectively.

From the above examples, we observe that the effort to integrate Web caching and Web prefetching in client-side proxies is intrinsically difficult since the effects of Web caching and prefetching are in fact entangled. The phenomenon illustrated in Fig. 2 even makes the integration of these two techniques more complicated. To maximize the performance improvement by integrating Web caching and Web prefetching, we propose in this paper an innovative cache replacement algorithm, not only considering the caching effect in the Web environment, but also evaluating the prefetching rules provided by various prefetching schemes. Explicitly, rather than using a static confidence threshold to control the prefetching level, we design a *normalized profit function* to dynamically evaluate the profit from caching an object (either a nonimplied object or an implied object according to some prefetching rule). This *normalized profit function* considers not only such factors as the size, fetching delay, reference rate, invalidation cost, and invalidation frequency of a Web object, but also the prefetching effect caused by various Web prefetching schemes. Based on the *normalized profit function* devised, we devise an innovative Web cache replacement Algorithm *IWCP* (standing for the *Integration of Web Caching and Prefetching*). Using an event-driven simulation, we evaluate the performance of our algorithm under several circumstances. By varying the simulation parameters, we observe the performance impacts of the cache capacity, of the confidence threshold, of the interarrival time of two consecutive requests in a user's access sequence, and of the distribution of object popularity. In the experimental studies, Algorithm *IWCP* is comparatively evaluated with an extended version of algorithm *LNC-R-W3* [33]. The experimental results show that Algorithm *IWCP* consistently outperforms this companion scheme in terms of the primary performance metric, delay saving ratio, and the secondary performance metric, hit ratio.

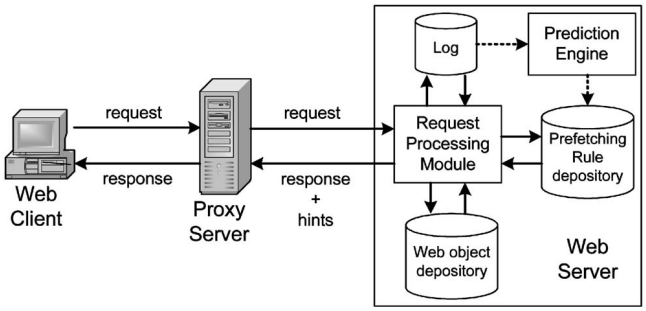


Fig. 3. The system model of Web prefetching and Web caching.

1.3 Organization

The remainder of this paper is organized as follows: In Section 2, we describe the system environment where the integration of Web caching and prefetching is considered. Algorithm *IWCP* is developed in Section 3. In Section 4, we comparatively evaluate the performance of Algorithm *IWCP* and the extended version of algorithm *LNC-R-W3*. This paper concludes with Section 5.

2 System Environment

The system model is presented in Section 2.1. The prefetching rules are described in Section 2.2.

2.1 System Model

This paper studies the integration of Web caching and Web prefetching in client-side proxies. A typical system model is depicted in Fig. 3. The proxy is located near the Web clients to avoid repeated round-trip delays between the clients and the origin Web servers. The origin Web server in our model is an enhanced Web server which employs a prediction engine to derive prefetching rules from the server's access log periodically. These derived rules are assumed to be frequent. That is, only rules with supports larger than the minimum support are derived and provided by Web servers. The derived prefetching rules are stored in the prefetching rule depository of the Web server.

As shown in Fig. 3, the proxy serves the requests sent from the Web clients. In the case that a cache miss occurs, the proxy will forward the request to the origin Web server for resolution. Upon receiving the request, the origin server will log this request into record, fetch the requested object

form the Web object depository, and check the prefetching rule depository at the same time. If this request triggers some prefetching rules in the prefetching rule depository, the objects implied by these prefetching rules and their corresponding confidences will be piggybacked to the responding message as hints and returned to the proxy. After the proxy receives the response with the hints piggybacked from the origin Web server, the proxy will first send the requested object back to the client and then determine whether it is worth caching the piggybacked implied objects in the proxy. Our study is thus to devise such a cache replacement algorithm for the integration of Web caching and Web prefetching techniques. Note that, in the case that a cache hit is found (i.e., the client's request can be satisfied directly with the proxy's local cache), we assume that the proxy will still communicate with the origin Web server to obtain the prefetching hints related to that request after the proxy has sent the response to the client. As such, we are able to investigate each request the prefetching hints from the origin Web server to ensure that the discovered prefetching hints are always up-to-date.

Note that the generation of prefetching rules can be also done on the proxy server. Specifically, it is known that the Web page access patterns can be discovered from proxy server logs [36]. The traffic due to the delivery of accessed object index can thus be saved. However, since there are possibly a large number of Web servers that proxy clients may access, the resulted overhead for computing and storing the prefetching rules could be a great burden for the proxy server. In addition, it is more reliable to perform such a computation based on the Web server's user access logs gathered from various clients.

2.2 Prefetching Rule

As explained in Section 1, a user's browsing sequence usually follows the hyperlinks between Web objects. This fact allows us to predict the user's browsing sequence or access pattern over the Web. Several algorithms based on Markov models and Web mining techniques are proposed in [16], [23], [26], [27], [30], [32], [39] to derive prefetching rules from the server's access log. To facilitate our later discussion in Section 3, we define the prefetching rule and relative terms below.

Let $\mathcal{O} = \{o_1, o_2, \ldots, o_n\}$ be a set of Web objects in a Web server and $\mathcal{D}$ be a set of users' access sequences, where each access sequence is a set of objects in $\mathcal{O}$ ordered by time.

Definition 1. *A prefetching rule is an implication of the form $o_1, \ldots, o_i \xrightarrow{c} o_{i+1}$, where $o_1, \ldots, o_i, o_{i+1} \in O$, $o_1, \ldots, o_i, o_{i+1}$ is an access sequence in $\mathcal{D}$, and c is the confidence of the prefetching rule.*

The prefetching rule $o_1, \ldots, o_i \xrightarrow{c} o_{i+1}$ implies that if the objects $o_1, \ldots, o_i$ have been referenced in a client's precedent requests, the object o_{i+1} will also be referenced in the client's subsequent requests with confidence c.

Definition 2. *The confidence c of the prefetching rule $o_1, \ldots, o_i \xrightarrow{c} o_{i+1}$ is a conditional probability of $P(o_1, \ldots, o_i, o_{i+1})/P(o_1, \ldots, o_i)$, where $P(o_1, \ldots, o_i, o_{i+1})$ is the probability that the sequence $o_1, \ldots, o_i, o_{i+1}$ is contained in an access sequence in $\mathcal{D}$ and $P(o_1, \ldots, o_i)$ is the probability that the sequence $o_1, \ldots, o_i$ is contained in an access sequence in $\mathcal{D}$.*

Definition 3. *An object o_{i+1} is referred to as an "implied object" if and only if the prefetching rule $o_1, \ldots, o_i \xrightarrow{c} o_{i+1}$ is triggered by some client who has already referenced the objects $o_1, \ldots, o_i$ in its precedent requests. Otherwise, the object o_{i+1} is called a "nonimplied" object.*

Note that, in [26], a more generalized form of the prefetching rule $o_1, \ldots, o_i \xrightarrow{c} o_{i+1}, \ldots, o_{i+j}$ is considered. However, we notice that if the prefetching rule $o_1, \ldots, o_i \xrightarrow{c} o_{i+1}, \ldots, o_{i+j}$ holds with the confidence c, the rules $o_1, \ldots, o_i \xrightarrow{c} o_{i+1}$, $o_1, \ldots, o_i \xrightarrow{c} o_{i+2}$, $\ldots$, $o_1, \ldots, o_i \xrightarrow{c} o_{i+j}$ will also hold with confidences greater than or equal to c. Hence, without loss of generality, we use the form of $o_1, \ldots, o_i \xrightarrow{c} o_{i+1}$ throughout this paper. In addition, most existing prediction algorithms use parameter w to limit the time between two consecutive accesses. In this paper, w is referred to as the *time window* of the prefetching rule and is employed in Section 3 to formulate the *normalized profit function*.

It is noted that different impacts arise when adopting different approaches for generating fetching rules. Specifically, as mentioned in Section 2.1, the rule discovery can be done on proxy servers [36]. Therefore, when there exist spatial localities of user access patterns, the rules discovered at proxy servers are able to reflect the local user behavior more precisely than those discovered at Web servers.

3 Cache Replacement Algorithm

With the background of the system environment described in Section 2, we present the proposed cache replacement algorithm for integrating Web caching and Web prefetching in client-side proxies in this section. In Section 3.1, we formulate the *normalized profit function* which will be used to determine the profit from caching either a nonimplied object or an implied object in our algorithm. Utilizing this *normalized profit function* as the eviction function, we design in Section 3.2 an innovative cache replacement algorithm, referred to as Algorithm *IWCP* (standing for the *Integration of Web Caching and Prefetching*). In Section 3.3, an example is presented to illustrate the execution of Algorithm *IWCP*.

3.1 Normalized Profit Function

Most cache replacement algorithms employ an *eviction function* to determine the priorities of objects to be replaced (or to be cached). We devise a *normalized profit function* to determine the profit from caching an object (either a nonimplied object or an implied object by a certain prefetching rule). As will become clear later, our eviction policy will depend on the profits of caching individual objects.

Let o_i denote object i. Assume that the intersite reference rate (i.e., the reference from cross-site hyperlinks) to object o_i is denoted by λ_i. The mean delay to fetch object o_i from the origin server to the proxy is denoted by d_i, where we define the delay to fetch an object as the time interval between sending the request and receiving the last byte of response. The size of object o_i is denoted by s_i. In addition, let $r_{i,j}$ denote the prefetching rule $j \longrightarrow o_i$, where the antecedent j is an access sequence as defined in Section 2. The confidence of the prefetching rule $r_{i,j}$ is denoted by $c_{i,j}$.

TABLE 1
A List of the Symbols Used

Symbol	*Description*
o_i	the object i
λ_i	the mean reference rate from cross-site hyperlinks to object i
d_i	the delay of fetching object i from the original server to the proxy
s_i	the size of object i
u_i	the mean validation rate to object i
v_i	the delay of performing a validation check for object i
$r_{i,j}$	the prefetching rule $j \longrightarrow o_i$ where j is an access sequence
$c_{i,j}$	the confidence of the prefetching rule $r_{i,j}$
w	the time window of the prefetching rule $r_{i,j}$

For better readability, a list of the symbols used is given in Table 1.

To determine the profit from caching o_i, we shall first calculate the expected number of references to object o_i within the time window w. Assume that the interarrival time of two consecutive intersite requests for o_i follows the exponential distribution. Then, the number of intersite references to o_i within the time window w will follow the Poisson distribution. For a nonimplied object o_i, the expected number of references to o_i within w is equal to

$$\lambda_i w = \sum_{x=0}^{\infty} x * \frac{(\lambda_i w)^x e^{-(\lambda_i w)}}{x!}$$

since the mean reference rate to o_i is λ_i. However, for an implied object o_i by a prefetching rule $r_{i,j}$, the expected number of references to o_i within the time window w becomes $\lambda_i w + c_{i,j} * 1$ since the probability that o_i will be referenced once within the time window w is increased by $c_{i,j}$. Therefore, to determine the expected number of references to o_i within the time interval w, we shall examine whether o_i is a nonimplied object or an implied object by a certain prefetching rule.

Note, however, that object o_i, as explained in Section 1, may be implied by one prefetching rule triggered by several clients (e.g., the rule $(A \xrightarrow{0.6} C)$ in Fig. 2a triggered by clients 1 and 2) or implied by several prefetching rules triggered by several clients (e.g., the rules $(C \xrightarrow{0.7} G)$ and $(D \xrightarrow{0.6} G)$ in Fig. 2b triggered by clients 1 and 2, respectively). In both cases, object o_i is called a *multiimplied object*, and the expected number of references to object o_i can be calculated as follows.

Property 1. *Let $p_1, p_2, \ldots, p_n$ denote the probabilities that the object o_i will be referenced once within the time window w by clients $1, 2, \ldots, n$, respectively. The expected number of references to object o_i within the time interval w is then equal to $p_1 + p_2 + \ldots + p_n$.*

For example, suppose the probabilities that o_i will be referenced within the time window w by clients 1, 2, and 3 are 0.4, 0.6, and 0.8, respectively. Let $P(x)$ denote the probability that o_i will be referenced x time(s) within the time interval w. Then, the expected number of references to object o_i within the time window w can be calculated as

$$\begin{aligned}
&\textstyle\sum_{x=1}^{3} x * P(x) \\
&= 1 \cdot (0.4 \cdot 0.4 \cdot 0.2 + 0.6 \cdot 0.6 \cdot 0.2 + 0.6 \cdot 0.4 \cdot 0.8) \\
&\quad + 2 \cdot (0.4 \cdot 0.6 \cdot 0.2 + 0.4 \cdot 0.4 \cdot 0.8 + 0.6 \cdot 0.6 \cdot 0.8) \\
&\quad + 3 \cdot (0.4 \cdot 0.6 \cdot 0.8) \\
&= 1.8 \\
&= 0.4 + 0.6 + 0.8.
\end{aligned}$$

With Property 1, we have the following theorem for the expected number of references to an object.

Theorem 1. *Let R denote the set of the prefetching rules that have been triggered to prefetch o_i. Let $n_{i,j}$ denote the number of the clients that triggered the rule $r_{i,j}$ in R. Then, the expected number of references to object o_i within the time window w is equal to $\lambda_i w + \sum_{r_{i,j} \in R} n_{i,j} * c_{i,j}$.*

Proof. Since each prefetching rule $r_{i,j} \in R$ implies that o_i has the probability $c_{i,j}$ to be referenced in the user's subsequent requests, according to Property 1, the expected number of references to object o_i increased by the prefetching rules in R is equal to

$$\sum_{r_{i,j} \in R} n_{i,j} * c_{i,j}.$$

Therefore, the overall expected number of references to object o_i will be $\lambda_i w + \sum_{r_{i,j} \in R} n_{i,j} * c_{i,j}$. □

With the expected number of references to object o_i within the time window w, we can thereafter determine the profit from caching object o_i. From the standpoint of client users, an optimal cache replacement algorithm is expected to minimize the response time perceived. In other words, an optimal cache replacement algorithm will be designed to maximize the delay saving by caching a certain set of Web objects. Thus, the profit from caching object o_i can be determined by Theorem 2 below.

Theorem 2. *The profit from caching object o_i is equal to*

$$P_f(o_i) = \left(\lambda_i w + \sum_{r_{i,j} \in R} n_{i,j} * c_{i,j} \right) * d_i - u_i w * v_i,$$

where R is the set of the prefetching rules that have been triggered to prefetch o_i, and $n_{i,j}$ is the number of the clients that triggered the rule $r_{i,j}$ in R.

Proof. The expression $(\lambda_i w + \sum_{r_{i,j} \in R} n_{i,j} * c_{i,j}) * d_i$ calculates the expected delay saving obtained by caching object o_i,

whereas the expression $u_i w * v_i$ calculates the extra delay incurred by validation checks for object o_i within the time window w. Thus, the net delay saving obtained by caching object o_i is thus $(\lambda_i w + \sum_{r_{i,j} \in R} n_{i,j} * c_{i,j}) * d_i - u_i w * v_i$. □

Finally, the profit function has to be normalized by the size of object i to reflect the object size factor of the caching parameter. Hence, the *normalized profit function* we devised in this paper is

$$P_f^N(o_i) = \frac{(\lambda_i w + \sum_{r_{i,j} \in R} n_{i,j} * c_{i,j}) * d_i - u_i w * v_i}{s_i}.$$

3.2 Algorithm *IWCP*

In Section 3.1, we have formulated the *normalized profit function*, P_f^N. Based on this *normalized profit function*, we design Algorithm *IWCP* in this section. The main idea behind Algorithm *IWCP* is: Order the objects (either a nonimplied object or an implied object) according to their values of the *normalized profit function*. Then, select the object with the highest value, one by one, until there is not enough space to hold any object more. The resulting set of selected objects is thus the objects to be cached in the proxy, and the rest objects are to be evicted.

However, similar to the problems that the caching algorithms proposed in [33], [37] suffered from, some of the assumptions made in Section 3.1 are unrealistic: The mean reference rates to Web objects are not static, but changes by time. This phenomenon is quite common, especially in a news Web site where the reference rates to the news documents decrease as time goes by. Similarly, the mean validation rates of Web objects also dynamically change by time. In addition, the delays of fetching objects from the origin Web servers and the costs for validation checks are not known a priori. In [33], [37], their algorithms employ the *sliding average* functions to practically estimate the running parameters in the Web proxy. In our algorithm, we continue using the *sliding average* functions in dealing with these problems. The *sliding average* functions we used are

$$\begin{aligned} d_i &= (1-\alpha) \cdot d_i + \alpha \cdot d_i^{new} \\ \lambda_i &= \frac{K}{t_{i,\lambda} - t_{i,\lambda}^K} \\ v_i &= (1-\beta) \cdot v_i + \beta \cdot v_i^{new} \\ u_i &= \frac{K}{t_{i,u} - t_{i,u}^K}. \end{aligned}$$

d_i^{new} is the new measured delay of fetching object o_i from the origin server. $t_{i,\lambda}$ is the time when the new request to object o_i is received by the proxy, whereas $t_{i,\lambda}^K$ is the time when the last K request is received. v_i^{new} is the new measured delay of checking the validation of object o_i from the origin server. $t_{i,u}$ is the time when the latest validated version of object o_i is received by the proxy, whereas $t_{i,u}^K$ is the time when the last K validated version is received. Note that λ_i is refreshed whenever a reference to $o_{i,j}$ is received by the proxy, whereas d_i is refreshed whenever a cache miss caused by the reference to object o_i occurs and object o_i is fetched from the Web server. v_i is refreshed whenever the latest validated version of o_i is received by the proxy, whereas u_i is refreshed whenever a validation is finished.

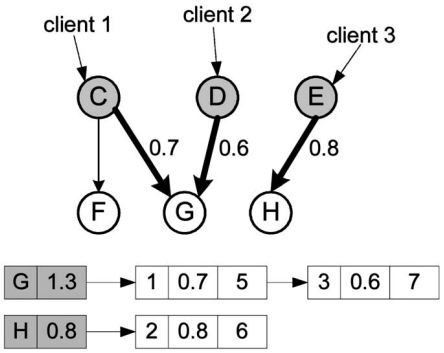


Fig. 4. An example to illustrate the linked list for the purpose of tracking the prefetching rules.

With the estimated running parameters, we shall devise the pseudocode of Algorithm *IWCP*. In Algorithm *IWCP*, we employ a heap data structure to implement the priority queue, C, which is used to store the objects cached in the proxy. For each object o_i in C, we maintain a record of the caching parameters (i.e., d_i, λ_i, v_i, u_i and s_i) and a linked list L_i to keep track of the prefetching rules triggered to prefetch o_i. An example of the use of a linked list is given below.

Example 5. Suppose client 1 has referenced object C and triggered the rule $(C \xrightarrow{0.7} G)$ at time 1, client 2 has referenced object D and triggered the rule $(D \xrightarrow{0.6} G)$ at time 3, and client 3 has referenced object E and triggered the rule $(E \xrightarrow{0.8} H)$ at time 2. Assume that the time window w of the prefetching rules is 4. Then, the linked lists L_G for object G is shown in the bottom of Fig. 4. The head node of the linked list L_G indicates that the cumulative confidence to prefetch o_G is $1.3 = 0.7 + 0.6$. The first node in the body of L_G records that client 1 has triggered one rule to prefetch o_G and the confidence of the rule is 0.7. The expired time of that rule is $5 = 1 + 4$ (i.e., the started time plus the time window). Similarly, the second node in the body of L_G records that client 2 has triggered one rule to prefetch o_G and the confidence of the rule is 0.6. The expired time of that rule is $7 = 3 + 4$.

It is noted that, in Example 5, if object G was not referenced by either client 1 or client 2 before the corresponding prefetching, rules expire. The corresponding node will be deleted from the linked list, and the cumulative confidence will decrease.

Algorithm IWCP (o_i)
01. **if** ($o_i \notin C$) {
02. fetch o_i from the origin server and then send o_i to the client;
03. insert o_i into C and extract the piggybacked hinted objects into H;
04. calculate $PF^N(o_i)$;
05. } **else** {
06. send o_i to the client;
07. search L_i to delete the node which can be satisfied by the request for o_i;
08. update $PF^N(o_i)$;
09. fetch from the origin server those hinted objects triggered by requesting o_i;

10. extract the hinted objects into H;
11. }
12. **for** (each hinted object $o_h \in H$) {
13. **if** ($o_h \in C$) {
14. insert $(client, c_{hj}, timer(now + w))$ into L_h;
15. update $PF^N(o_h)$;
16. } **else** {
17. insert o_h into C;
18. insert $(client, c_{hj}, timer(now + w))$ into L_h;
19. calculate $PF^N(o_h)$;
20. }
21. }
22. **BuildHeap** (C);
23. **while** ($S_{now} > cacheCapacity$) {
24. evict the first object o_j from C;
25. $S_{now} = S_{now} - s_j$;
26. }

Based on the above data structures, Algorithm *IWCP* is devised as follows: Whenever the proxy receives a request from the client, Algorithm *IWCP* is called by the proxy to handle the request. Algorithm *IWCP* takes one argument o_i, the client's request, as the input. In line 1, the proxy first checks whether o_i is already cached in C. If o_i is not cached in C, the proxy fetches o_i from the origin Web server and send o_i back to the requesting client immediately. o_i is then inserted into C and the piggybacked implied objects are extracted to H in line 3. The normalized profit from caching o_i is calculated in line 4. On the other hand, if o_i is already cached in C, the proxy sends o_i back to the client immediately. Then, the proxy searches L_i to examine if any node in the linked list matches the client's request for o_i. If some match is found, the corresponding node will be deleted and the normalized profit from caching o_i will be updated in line 8. In line 9, although o_i is already cached in C, the proxy contacts the origin server to check whether this request for o_i can trigger any prefetching rule. If some implied objects are piggybacked from the origin server, they will be extracted to H in line 10. After that, in the **for** loop of lines 12-21, the proxy, for each hinted object o_h in H, examines whether o_h is already cached in C. If it is, the proxy inserts a new node into L_h, indicating that o_h is implied by another prefetching rule triggered by the client. The normalized profit from caching o_h will be updated accordingly in line 15. However, if o_h is not in C, the proxy will insert o_h into C and then insert a new node into L_h, indicating that one prefetching rule implies to prefetch o_h. Since o_h is a newly inserted object, the proxy calculates the normalized profit from caching o_h in line 19. Thereafter, in line 22, the proxy calls the function **BuildHeap()** to maintain the order of the objects cached in the priority queue C since the normalized profits of some objects are changed and/or some new objects are inserted into C. Finally, in the **while** loop of line 23-26, the proxy evicts the object of the lowest caching priority from the cache, one by one, until the cumulative size S_{now} of C is lower than the cache capacity.

TABLE 2
The Profile of the Objects Used in the Illustrative Example

object (o_i)	*reference rate* (λ_i)	*related prefetching rules* ($r_{i,j}$)
A	2	$(A \xrightarrow{0.9} C)$
B	1	$(B \xrightarrow{0.6} D)$; $(B \xrightarrow{0.9} F)$ $w = 4$
C	0.5	$(C \xrightarrow{0.8} E)$; $(C \xrightarrow{0.6} F)$ $w = 4$
D	0.25	omitted
E	0.1	omitted
F	0.05	omitted

3.3 An Illustrated Example

In this section, we use an example to illustrate how Algorithm *IWCP* works. The profile of the objects used in the example is given in Table 2. For simplicity, except the mean reference rate, we assume that all objects used in the example have the same values of the other caching parameters (i.e., fetching delay, object size, validation rate, and validation cost), and those parameters are excluded from the profile.

Fig. 5a illustrates the snapshot of the current cache status in the proxy. The capacity of the proxy cache is 4, and there are already four objects A, B, C, and D cached in it. One node in the body of the linked list L_c indicates that object C was previous implied by rule $(A \xrightarrow{0.9} C)$ when A was referenced in client 1's precedent requests. Suppose a new request for object C made by client 1 comes to the proxy. Since C is already cached in the proxy, the proxy will respond to client 1 with object C immediately. Then, in line 7, the proxy finds that the node $(1, 0.9, 4)$ matches the request for object C made by client 1. Hence, this node will be deleted, and the normalized profit from caching object C will be updated. In addition, the proxy, in line 9, will contact the origin server to see if any prefetching rule can be triggered by this request for object C. Since two prefetching rules $(C \xrightarrow{0.8} E)$ and $(C \xrightarrow{0.6} F)$ will be triggered by the request for C, the proxy will receive two piggybacked implied objects E and F from the server. These two implied objects will be thereafter extracted to H in line 10. In the **for**

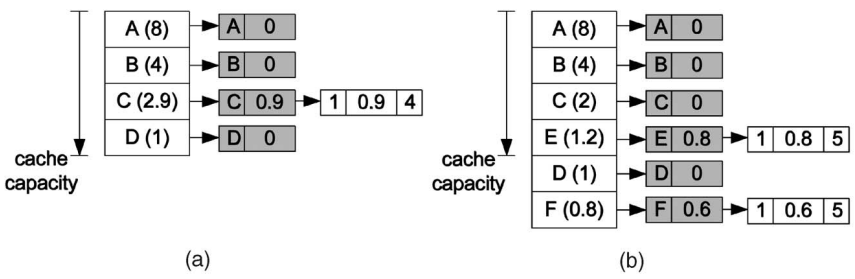


Fig. 5. An illustrative example for *Algorithm IWCP*.

TABLE 3
Parameters of the Simulation Model

Parameter	*Distribution/Default value*
Number of Web objects	$N = 1,000$
Object size	Lognormal dist. ($\mu = 9.375$)
Object popularity	Zipf-like dist. ($\alpha = 0.8$)
Fanout	Uniform dist. $[1, MaxFanout]$
Number of users' access sequences	$100,000$
Length of an access sequence	Inverse Gaussian dist. ($\mu = SeqLen$)
Interarrival time of consecutive access sequences	Exp. dist. ($\mu = InterSeqTime$)
Interarrival time of consecutive requests	Pareto dist. ($\mu = InterReqTime$)
Cache capacity	$CacheCapacity * (\sum$object size$)$
Object fetching delay	Exp. dist. ($\mu = 100$)
Time window	$w = 20$

loop of lines 12-21, the implied objects E and F will be inserted into the cache, and the corresponding linked list L_E and L_F will be initiated with new nodes inserted. Since the normalized profit of object C has been changed and objects E and F are inserted, the proxy calls **BuildHeap()** function in line 22 to determine the new caching priorities of the objects. The snapshot of the cache status after reordering is shown in Fig. 5b. Since the normalized profit from caching object E becomes larger than that from caching object D, D will be evicted from the cache, not to mention object F which is of the lowest caching priority.

4 Performance Analysis

In this section, we will evaluate the performance of our cache replacement algorithm by conducting an event-driven simulation. The primary goal of performing an event-driven simulation is to understand the effect on the performance of our cache replacement algorithm by varying system parameters. We will describe the simulation model in Section 4.1. The experimental results will be shown in Section 4.2.

4.1 Simulation Model

The simulation model is designed to reflect the system environment as described in Section 2. To generate synthetic client workloads, we assume that the number of total Web objects is N. The sizes of the Web objects are lognormally distributed with the mean value of 9.375 KBytes [5]. The popularity of the Web objects follows a Zipf-like distribution, which is widely accepted to be a model for real Web traces [4], [7], [33]. As shown in [14], [20], since small files are much more frequently referenced than large files, we assume that there is negative correlation between the object size and its popularity. In addition, to model the spatial locality of the Web objects, we assume that each object has hyperlinks to other objects. The fanout of each object (i.e., the number of its outgoing hyperlinks to other objects) is a random variable uniformly distributed between the interval $[1, MaxFanout]$. For each outgoing link, we assign a weight on it. The weight is in proportion to the popularity of the object pointed to by that hyperlink. For example, suppose object A has three outgoing hyperlinks pointing to objects B, C, and D. If the popularity of objects B, C, and D is $0.6, 0.4, 0.2$, respectively, the weight on the link $A \longrightarrow B$ will be assigned to be $0.5 = 0.6/(0.6 + 0.4 + 0.2)$. Similarly, the weights on the links $A \longrightarrow C$ and $A \longrightarrow D$ will be assigned to be 0.33 and 0.17, respectively.

After setting up the attributes of every Web object, we generate several synthetic client workloads as follows: Each workload consists of $100,000$ users' access sequences. For each access sequence, we first choose a starting object from total N objects according to their popularity. Then, following an Inverse Gaussian distribution with the mean value of $SeqLen$, we determine the length of the client's access sequence [29]. Once the starting object and the length of the sequence have been decided, the subsequent objects in the access sequence can be chosen in accordance with the weight on the outgoing hyperlinks, one by one, until the length of the sequence is satisfied. To simulate the requests originating from different clients, two parameters are designated in our simulation model. The interarrival time between two consecutive access sequences is modeled by an exponential distribution with the mean value of $InterSeqTime$. The interarrival time between two consecutive requests (i.e., client's thinking time) within an access sequence is modeled by a Pareto distribution with the mean value of $InterReqTime$ [29].

As to the model of the proxy, we choose the default cache capacity to be $CacheCapacity * (\sum\ object\ size)$. The delays of fetching objects from Web servers are given by an exponential distribution. Since it is shown that the correlation between object size and fetching delay is relatively low [33], we assume that there is no correlation between Web object size and the delay of fetching it from the Web server. According to Theorem 2, the value of time window w linearly contributes to the first term of our profit function. Since the focus of this paper is to evaluate the effect of integrating the fetching terms into the cache replacement criteria, i.e., the profit function, we set the default value of time window to be $w = 20$. Table 3 provides a summary of the parameters used in the simulation model.

4.2 Experimental Results

Based on the simulation model devised, we implement a simulator using Microsoft Visual C++ package on IBM compatible PC with Pentium III 450 CPU and 192 Mbytes

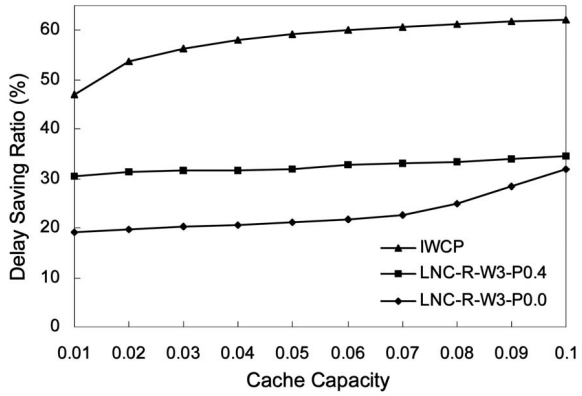


Fig. 6. Delay saving ratios under various cache capacities.

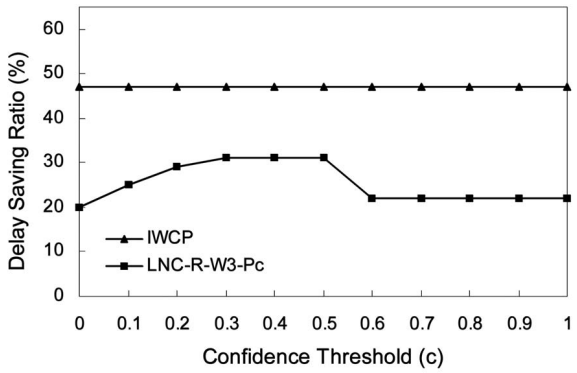


Fig. 8. Delay saving ratios under various confidence thresholds.

RAM. Each experimental result is obtained by 10 simulation runs with different seeds. The performance metric employed in our experiments is the *delay saving ratio* which is defined as the ratio of total delay saved from cache hits to the total delay incurred under the circumstance without proxy caching. Our yardstick is an extension to algorithm *LNC-R-W3* [33], which is denoted as *LNC-R-W3-Pc*. In *LNC-R-W3-Pc*, the cache replacement algorithm *LNC-R-W3* proposed in [33] is extended to prefetch/cache the piggybacked implied objects if the prefetching confidences of the implied objects are higher than a predefined confidence threshold c.

4.2.1 Impact of Cache Capacity

In the first experiment, we investigate the performance of our cache replacement algorithm by varying the cache capacity. The simulation results are shown in Figs. 6 and 7. As presented in Figs. 6 and 7, our algorithm outperforms the other ones in terms of the delay saving ratio and the hit ratio, respectively. Specifically, the mean improvements of the delay saving ratios over *LNC-R-W3-P0.4* and *LNC-R-W3-P0.0* algorithms are 78.46 percent and 146.42 percent, respectively. The main reason for algorithms *LNC-R-W3-P0.4* and *LNC-R-W3-P0.0* not to perform well is that they employ static confidence thresholds to control the prefetching level of the piggybacked implied objects. Specifically, these two algorithms neither take into account the relationship between the caching parameters and the prefetching confidences as mentioned in Section 1 nor consider the phenomena illustrated in Fig. 2. On the other hand, utilizing the normalized profit function devised in Section 3.1, Algorithm *IWCP* can dynamically determine whether an implied object is worth caching or not. Thus, Algorithm *IWCP* shows significant performance improvement over *LNC-R-W3-P0.4* and *LNC-R-W3-P0.0*.

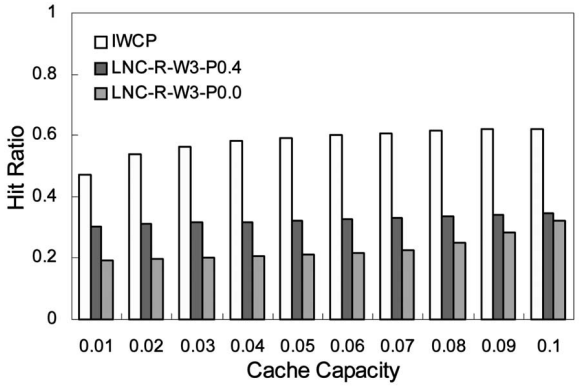


Fig. 7. Hit ratios under various cache capacities.

It is interesting to note that, as the cache capacity increases, the performance difference between *LNC-R-W3-P0.4* and *LNC-R-W3-P0.0* becomes smaller. The reason is that, as the cache capacity increases, the cache space becomes less precious. Thus, one may lower the confidence threshold and invest more space in caching those implied objects with relatively low prefetching confidences, so as to increase the overall cache hit ratio. Since Algorithm *IWCP* utilizes a dynamic threshold to determine whether to cache an implied object, Algorithm *IWCP* is not affected by the factor of cache capacity, thus able to consistently outperform other companion schemes.

4.2.2 Impact of Confidence Threshold

In the second experiment, we examine the performance improvements of Algorithm *IWCP* by varying the confidence threshold. The purpose of this experiment is to explore the impact of the confidence threshold upon the algorithm *LNC-R-W3-Pc*, so as to learn more insights into the performance improvements of Algorithm *IWCP*. The simulation results are shown in Fig. 8. As presented in Fig. 8, the delay saving ratio of the algorithm *LNC-R-W3-Pc* increases as the confidence threshold increases from 0 to 0.5. The reason is that, by filtering out those implied objects with relatively low prefetching confidences, the cache space can be saved to admit more beneficial objects. However, when the confidence threshold is set too high (i.e., above 0.6 in our experiment), all the prefetching hints will be filtered out by the proxy. As a consequence, the performance of algorithm *LNC-R-W3-Pc* will decrease to the level as no prefetching scheme is employed, indicating that the performance improvement by Web prefetching scheme is totally whittled down. From the simulation results, we can observe that Algorithm *IWCP* still consistently outperforms algorithm *LNC-R-W3-Pc*, even in the best case of algorithm *LNC-R-W3-Pc* (i.e. when c varies from 0.3 to 0.5).

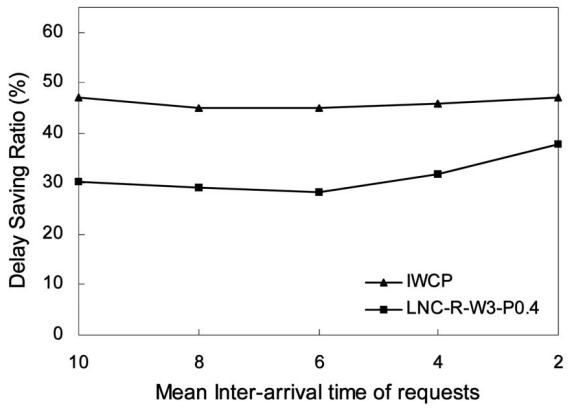


Fig. 9. Delay saving ratios under various interarrival time of consecutive requests.

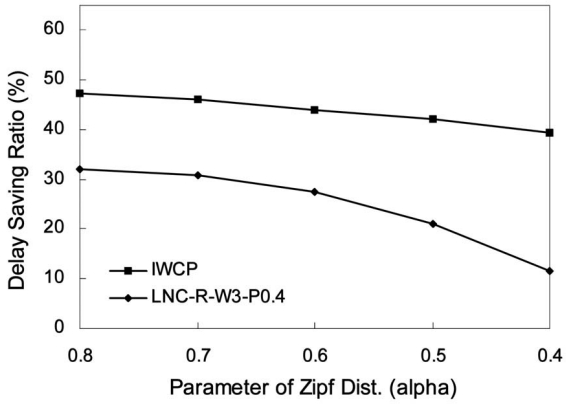


Fig. 10. Delay saving ratios under various distribution of object popularity.

4.2.3 Impact of InterReqTime

In the third experiment, we investigate the performance of Algorithm *IWCP* by varying the interarrival time of two consecutive requests within a client's access sequence (i.e., $InterReqTime$). The simulation results are shown in Fig. 9. As presented in Fig. 9, Algorithm *IWCP* still outperforms *LNC-R-W3-P0.4* in terms of the delay saving ratio. Note that although the delay saving ratio of algorithm *LNC-R-W3-P0.4* decreases slightly as the $InterReqTime$ decreases from 10 to 6, the delay saving ratio of the algorithm *LNC-R-W3-Pc* increases significantly when the $InterReqTime$ decreases from 6 to 2. The reason is that when the interarrival time of two consecutive requests becomes shorter, the prefetched object can be referenced sooner. Thus, the cache space, in turn, can be released earlier to admit other beneficial objects, thereby increasing the level of performance improvement by the Web prefetching scheme.

Note that, in our simulation model, the parameter $InterReqTime$ can be deemed as the time window w of the prefetching rules described in Section 2.2. It is interesting to observe that the simulation results in Fig. 9, which favor shorter $InterReqTime$, conform with the fact that the prefetching rules with higher confidences and shorter time window are more beneficial to be used in the Web prefetching scheme.

4.2.4 Impact of Object Popularity

In the fourth experiment, we observe the performance of our cache replacement algorithm by varying the distribution of object popularity (i.e., the value of the Zipf-like distribution parameter, α). The simulation results are shown in Fig. 10, where it can be seen that the delay saving ratios of Algorithm *IWCP* and algorithm *LNC-R-W3-P0.4* decrease as the value of α decreases (i.e., the skew of object popularity decreases). However, we note that the performance degradation of algorithm *LNC-R-W3-P0.4* is more severe than that of Algorithm *IWCP*. This can be explained by the fact that the distribution of the confidences of prefetching rules is in proportion to the distribution of object popularity. In other words, the less skewed the object popularity, the less skewed the distribution of the confidences of prefetching rules is. Hence, under a static threshold control, the number of prefetching hints filtered out increases as the value of α decreases, and the performance of Web prefetching scheme in turn degrades. On the other hand, Algorithm *IWCP* utilizes the normalized profit function as a dynamic threshold control, thus performing less sensitively to the value of α. The simulation results show that Algorithm *IWCP* still outperforms the algorithm *LNC-R-W3-P0.4* under various values of Zipf-like distribution parameter α.

4.2.5 Execution Overhead

In the fifth experiment, we evaluate the execution overhead of Algorithm *IWCP* under various cache capacity. It is observed that although the execution time of Algorithm *IWCP* is larger than that of algorithm *LNC-R-W3*, the net performance improvement is very prominent, considering the significant delay savings achieved by Algorithm *IWCP*. From our empirical measurements, the execution overhead in Algorithm *IWCP* is around one millisecond when cache capacity varies from 0.01 to 0.05, whereas the delay saving Algorithm *IWCP* obtains is around 100 milliseconds (In the worst case, the delay saving could be in seconds.) Note that according to the definitions provided in [33], the delay saving ratio is a fraction of communication delays saved by satisfying the requests from cache. Since the delay saving ratio is around 40 percent to 60 percent in our previous experiments, it follows that algorithm IWCP can successfully reduce the delay with negligible overhead.

In addition, as mentioned in [33], the execution time of the function-based cache replacement algorithm can be further improved by more sophisticated implementation. Overall, Algorithm *IWCP* can effectively decrease the user perceived response time by integrating Web caching and prefetching.

5 Conclusions

In this paper, we developed Algorithm *IWCP* by integrating Web caching and Web prefetching in client-side proxies. We formulated a *normalized profit function* to evaluate the profit from caching an object (either a nonimplied object or an implied object by a certain prefetching rule). This *normalized profit function* not only considered such factors as the size, fetching cost, reference rate, invalidation cost, and invalidation frequency of a Web object, but also exploited the effect

caused by various Web prefetching schemes. Utilizing the *normalized profit function* as the eviction function, we proposed Algorithm *IWCP* as the new Web cache replacement algorithm. Using an event-driven simulation, we showed that Algorithm *IWCP* consistently outperforms algorithm *LNC-R-W3-Pc* in terms of the primary performance metric, delay saving ratio, and the secondary performance metric, hit ratio.

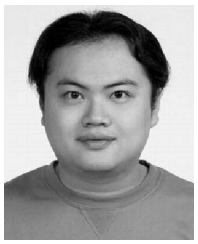
ACKNOWLEDGMENTS

The work was supported in part by the National Science Council of Taiwan, R.O.C., under Contract NSC93-2752-E-002-006-PAE.

REFERENCES

[1] Akamai Technologies Inc., http://www.akamai.com/, 2004.
[2] Mirror Image Internet Inc., http://www.mirrorimage.com/, 2004.
[3] Sandpiper Networks/Digital Island, Inc., http://www.sandpiper.net/, 2004.
[4] C. Aggarwal, J.L. Wolf, and P.-S. Yu, "Caching on the World Wide Web," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 1, pp. 94-107, Jan./Feb. 1999.
[5] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proc. 1998 ACM SIGMETRICS Int'l Conf. Measurements and Modeling of Computer Systems,* 1998.
[6] G. Barish and K. Obraczka, "World Wide Web Caching: Trends and Techniques," *IEEE Comm. Magazine,* Internet Technology Series, pp. 178-185, 2000.
[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. IEEE INFOCOM 1999,* Mar. 1999.
[8] P. Cao, E.W. Felten, A. Karlin, and K. Li, "A Study of Integrated Prefetching and Caching Strategies," *Proc. 1995 ACM SIGMETRICS Int'l Conf. Measurements and Modeling of Computer Systems,* pp. 188-197, 1995.
[9] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," *Proc. 1997 USENIX Symp. Internet Technology and Systems,* 1997.
[10] M.-S. Chen, J.-S. Park, and P.S. Yu, "Efficient Data Mining for Path Traversal Patterns," *IEEE Trans. Knowledge and Data Eng.*, vol. 10, no. 2, pp. 209-221, Mar./Apr. 1998.
[11] K. Chinen and S. Yamaguchi, "An Interactive Prefetching Proxy Server for Improvement of WWW Latency," *Proc. Seventh Ann. Conf. Internet Soc.,* June 1997.
[12] E. Cohen, B. Krishnamurthy, and J. Rexford, "Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters," *Proc. ACM SIGCOMM 1998,* pp. 241-253, 1998.
[13] M. Crovella and P. Barford, "The Network Effects of Prefetching," *Proc. IEEE INFOCOM 1998,* pp. 1232-1240, 1998.
[14] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of WWW Client-Based Traces," technical report, Boston Univ., Apr. 1995.
[15] B.D. Davison, "Predicting Web Actions from HTML Content," *Proc. 13th ACM Conf. Hypertext and Hypermedia,* pp. 159-168, June 2002.
[16] M. Deshpande and G. Karypis, "Selective Markov Models for Predicting Web-Page Accesses," *Proc. First SIAM Int'l Conf. Data Mining,* 2001.
[17] D. Duchamp, "Prefetching Hyperlinks," *Proc. Second USENIX Symp. Internet Technologies and Systems,* pp. 127-138, Oct. 1999.
[18] L. Fan, P. Cao, J. Almeida, and A.Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Trans. Networking,* vol. 8, no. 3, pp. 281-293, 2000.
[19] L. Fan, P. Cao, W. Lin, and Q. Jacobson, "Web Prefetching between Low-Bandwidth Clients and Proxies: Potential and Performance," *Proc. 1999 ACM SIGMETRICS Int'l Conf. Measurements and Modeling of Computer Systems,* pp. 178-187, 1999.
[20] S. Glassman, "A Caching Relay for the World Wide Web," *Computer Networks and ISDN Systems,* vol. 27, 1994.
[21] A. Kraiss and G. Weikum, "Integrated Document Caching and Prefetching in Storage Hierarchies Based on Markov-Chain Predictions," *Very Large Databases J.,* vol. 7, no. 3, pp. 141-162, 1998.
[22] T. Kroeger, D.E. Long, and J. Mogul, "Exploiting the Bounds of Web Latency Reduction from Caching and Prefetching," *Proc. USENIX Symp. Internet Technologies and Systems,* pp. 13-22, 1997.
[23] B. Lan, S. Bressan, B.C. Ooi, and K. Tan, "Rule-Assisted Prefetching in Web Server Caching," *Proc. 2000 ACM Int'l Conf. Information and Knowledge Management,* 2000.
[24] R. Lempel and S. Moran, "Predictive Caching and Prefetching of Query Results in Search Engines," *Proc. 12th Int'l Conf. World Wide Web,* pp. 19-28, May 2003.
[25] R. Lempel and S. Moran, "Optimizing Result Prefetching in Web Search Engines with Segmented Indices," *ACM Trans. Internet Technology,* vol. 4, no. 1, pp. 31-59, Feb. 2004.
[26] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos, "Effective Prediction of Web-User Accesses: A Data Mining Approach," *Proc. Workshop Web Usage Analysis and User Profiling (WebKDD),* 2001.
[27] V. Padmanabhan and J.C. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency," *ACM SIGCOMM Computer Comm. Rev.,* vol. 26, no. 3, 1996.
[28] A. Papoulis, *Probability, Random Variables and Stochastic Processes.* McGraw Hill, 1991.
[29] J. Pitkow, "Summary of WWW Characteristics," *World Wide Web,* vol. 2, nos. 1-2, pp. 3-13, 1999.
[30] J. Pitkow and P. Pirolli, "Mining Longest Repeating Subsequence to Predict World Wide Web Surfing," *Proc. Second USENIX Symp. Internet Technologies and Systems,* 1999.
[31] K. Ross, "Hash-Routing for Collections of Shared Web Caches," *IEEE Network Magazine,* pp. 37-44, Nov.-Dec. 1997.
[32] R.R. Sarukkai, "Link Prediction and Path Analysis Using Markov Chains," *Proc. Ninth Int'l World Wide Web Conf.,* 2000.
[33] J. Shim, P. Scheuermann, and R. Vingralek, "Proxy Cache Algorithms: Design, Implementation, and Performance," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 4, pp. 549-561, July/Aug. 1999.
[34] S. Williams, M. Abrams, C.R. Standridge, G. Abdulla, and E. Fox, "Removal Policies in Network Caches for World Wide Web Documents," *Proc. ACM SIGCOMM 1996,* pp. 293-304, 1996.
[35] R.P. Wooster and M. Abrams, "Proxy Caching That Estimates Page Load Delays," *Proc. Sixth Int'l World Wide Web Conf.,* 1997.
[36] Y.-H. Wu and A.L. Chen, "Prediction of Web Page Accesses by Proxy Server Log," *World Wide Web,* vol. 5, no. 1, pp. 67-88, 2002.
[37] J. Xu, Q. Hu, D.-L. Lee, and W.-C. Lee, "SAIU: An Efficient Cache Replacement Policy for Wireless On-Demand Broadcasts," *Proc. 2000 ACM CIKM Int'l Conf. Information and Knowledge Management,* pp. 46-53, 2000.
[38] Q. Yang and H.H. Zhang, "Integrating Web Prefetching and Caching Using Prediction Models," *World Wide Web,* vol. 4, no. 4, pp. 299-321, 2001.
[39] Q. Yang, H.H. Zhang, and I.T. Li, "Mining Web Logs for Prediction Models in WWW Caching and Prefetching," *Proc. Seventh ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 473-478, Aug. 2001.

Wei-Guang Teng received the BS and PhD degrees from the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, R.O.C., in 1998 and 2004, respectively. His research interests include data mining, multimedia networking, and database.



Cheng-Yue Chang received the MS and PhD degrees in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1998 and 2003, respectively. He is currently a senior researcher at Arcadyan Technology, Taipei. His research interests include World Wide Web systems, multimedia networks, and home networks.

Ming-Syan Chen received the BS degree in electrical engineering from National Taiwan University, Taipei, Taiwan, and the MS and PhD degrees in computer, information and control engineering from The University of Michigan, Ann Arbor, in 1985 and 1988, respectively. Dr. Chen is currently a professor and the chairman of the Graduate Institute of Communication Engineering, a professor in the Electrical Engineering Department, and also a professor in the Computer Science and Information Engineering Department, National Taiwan University, Taipei, Taiwan. He was a research staff member at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, from 1988 to 1996. His research interests include database systems, data mining, mobile computing systems, and multimedia networking, and he has published more than 180 papers in his research areas. In addition to serving as a program committee member in many conferences, Dr. Chen served as an associate editor of *IEEE Transactions on Knowledge and Data Engineering* from 1997 to 2001, is currently on the editorial board of the *Very Large Data Base Journal*, the *Knowledge and Information Systems Journal*, the *Journal of Information Science and Engineering*, and the *International Journal of Electrical Engineering*, and was a Distinguished Visitor of the IEEE Computer Society for Asia-Pacific from 1998 to 2000. He served as the program chair of PAKDD-02 (Pacific Area Knowledge Discovery and Data Mining), program vice-chair of IEEE International Conference on Distributed Computing Systems (ICDCS) 2005, the International Conference on Parallel Processing (ICPP) 2003, program vice-chair of VLDB-2002 (Very Large Data Bases), and also general chair and program chair of several other conferences. He was a keynote speaker on Web data mining at the International Computer Congress in Hong Kong, 1999, a tutorial speaker on Web data mining in DASFAA-1999 and on parallel databases at the 11th IEEE International Conference on Data Engineering in 1995 and also a guest coeditor for *IEEE Transactions on Knowledge and Data Engineering* on a special issue for data mining in December 1996. He holds, or has applied for, 18 US patents and seven ROC patents in the areas of data mining, Web applications, interactive video playout, video server design, and concurrency and coherency control protocols. He is a recipient of the NSC (National Science Council) Distinguished Research Award and K.-T. Li Research Penetration Award for his research work, and also the Outstanding Innovation Award from IBM Corporate for his contribution to a major database product. He also received numerous awards for his research, teaching, inventions, and patent applications. Dr. Chen is a fellow of the IEEE and a member of the ACM.