

# Web Prefetching: Costs, Benefits and Performance

Yingyin Jiang, Min-You Wu, and Wei Shu  
*Department of Electrical and Computer Engineering*  
*The University of New Mexico*  
*Albuquerque, NM 87131, USA*  
*{jyy, wu, shu}@ece.unm.edu*

## Abstract

*Due to the fast development of internet services and a huge amount of network traffic, it is becoming an essential issue to reduce World Wide Web user-perceived latency. Although web performance is improved by caching, the benefit of caches is limited. To further reduce the retrieval latency, web prefetching becomes an attractive solution to this problem. Prefetching reduces user access time, but at the same time, it requires more bandwidth and increases traffic. Performance measurement of prefetching techniques is primarily in terms of hit ratio and bandwidth usage. A significant factor for a prefetching algorithm in its ability to reduce latency is deciding which objects to prefetch in advance. This paper presents a solution space of prefetching according to various object-selecting criteria and a comparison of their performance is provided.*

## 1 Introduction

In today's common web configurations, the proxy server exists between clients and web servers. The proxy server intercepts the requests from clients, serves with the requested object if it is present in proxies, or retrieves new objects from the appropriate web server, then caches the new objects or updates the existing objects if it has been modified since the last reference.

Web caching has been recognized as an effective solution to minimize user access latency. But its benefit has proved to be significantly limited by the rapid changes of objects in the web [1]. One way to further increase the cache hit ratio is to anticipate future requests and prefetch these objects into a local cache. On the other hand, prefetching consumes more network bandwidth. Venkataramani et al. [2] mentioned that system resources in terms of bandwidth usage and response time in terms of hit ratio are not directly comparable quantities. In this paper, by considering that network bandwidth usage is cost and hit ratio is the performance of prefetching, we introduce a concept which defines our notion of the value of prefetching by measuring the balance between the response time improvement and the extra amount of system resources consumed by prefetching compared to demand caches.

We use threshold-based algorithms and fetch those objects with values that exceed the threshold. The object selection criterion of prefetching determines which object to replicate in advance. We have several options of criteria such as object popularity and lifetime. Based on the analytical statistics model in [2], we examine these prefetching criteria and obtain a solution space in which these different criteria span different performance results. These analytical results provide a proof of concept for different prefetching criteria, e.g., the criterion of lifetime favoring the long-lived objects results in low bandwidth cost but low hit ratio. The key contribution of our work is a new prefetch-

ing algorithm that balances object access frequency and object update frequency. The algorithm can also be tuned to emphasize fetching long-lived or popular objects rather than always maximizing benefit/cost by balancing object lifetime vs. popularity.

The rest of the paper is organized as follows. We present the most significant characteristics of web workload in Section 2 and describe the caching/prefetching architecture in Section 3. A number of prefetching algorithms according to different object-selecting criteria are introduced in Section 4. Section 5 describes the analytical model to formulate the steady-state hit ratio and bandwidth. Section 6 compares and analyzes the results of these prefetching algorithms. Future work is outlined in Section 7 followed by the conclusion in Section 8.

## 2 Web Object Characteristics

By studying the characteristics of web access, we can get some hints from web workload and user access patterns. Researchers have analyzed traces collected at proxy servers and routers and identified some major characteristics of web objects.

### 2.1 Zipf-Like Distribution

The Zipf-like request model is widely accepted and used in web traffic modeling literature [3]. It is a general case of Zipf’s law, which is originally applied to the relationship between a word’s popularity in terms of rank and its frequency of use [4]. It states that if one ranks the popularity of words used in a given text as  $i$ , the frequency of use as  $p$ , then  $p = k/i$ , where  $k$  is a constant,  $k = \frac{1}{\sum_{i=1}^N (\frac{1}{i})}$ .

Glassman was the first to use Zipf’s law to model the distribution of web page requests [5], where about 100,000 HTTP requests were gathered. It is found that the request distribution fits Zipf’s law,  $k/i$ , quite well. Cunha et al. [6]

gathered 500,000 web accesses. From observations, they generalized that the distribution of web requests follows a Zipf-like distribution:

$$p = k/i^\alpha, \text{ where } k = \frac{1}{\sum_{i=1}^N (\frac{1}{i^\alpha})} \quad (1)$$

where  $\alpha = 0.986$ , which we use as a default parameter. Breslau et al. [7] investigated the requests on six different web servers. The  $\alpha$  value varies from trace to trace, ranging from 0.64 to 0.83. Nishikawa et al. [8] studied an access log of 2,000,000 requests and found that the request distribution followed the Zipf’s law with  $\alpha = 0.75$ . According to the Zipf’s distribution, web prefetching could be effective by fetching the most popular documents ahead of time.

### 2.2 Web Object Size

By [9, 10, 11, 12, 13, 14], the average object size retrieved by clients is on the order of 10-15KBytes. Studies by Barford and Crovella [15] show that web object sizes exhibit a distribution that is a hybrid of a lognormal and a heavy-tailed Pareto distribution. Breslau et al. [7] showed that there does not seem to be a strong correlation between document size and its access frequency for several traces. The calculated correlation coefficients say that the correlation, if any, is weak and can be ignored.

Since web object size is not the important parameter in our experiment, for simplicity we let all the web objects in our study be the same size of 10K bytes and assume no correlation between document size and its access frequency.

### 2.3 Web Object Lifetime

Kroeger [16] found that even with the unlimited size cache, the latency improved only 26% from caching due to a large amount of misses from the modification of the existing objects in the cache. Hence, the lifetime of web objects, which is the indication of modification, is an

important factor to determine which object to fetch in order to make prefetching effective.

Douglis et al. [1] used live traces to evaluate the rate and nature of change of web resources. They studied three aspects: the change rate for a resource, age at time of references and the modification interval. From their plots, it can be seen that the design of a prefetching algorithm must confront the high change rate in the web if we wish to provide significant latency improvement. Also as shown in [1, 7], there is also only a weak correlation between object access frequency and its rate of change.

### 3 Caching/Prefetching Architecture

As in [2], the caching/prefetching hierarchical structure envisions collaboration between proxy servers and content distribution servers, as well as between content distribution servers. The lower level proxies cater to users by mainly servicing for their requests, and the higher level content distribution sites use prefetching on effective content distribution. They maintain the content collection using prefetching selection algorithms. New objects are added to the collection of data based on server assistance and user access.

Our prefetching selection algorithms use as input such global access statistics as (1) estimates of object reference frequencies and (2) estimates of object lifetimes. These estimation can be well maintained by content distribution servers if they can collaborate between each other. Content servers collect user access statistics and publish information of objects popularity and patterns of usage; gather usage reports from their users, aggregate and analyze them and make them available to each other. They can also send prefetching hints to each other or actively push to each other objects that are likely to request in the near future. Whenever the replicated object is updated in the original server, the new version of it will be sent immediately to any cache that has subscribed

to it. Although it is a key issue to maintain these global access statistics estimates for our prefetching algorithms, this is not the main concern of our paper. We assume these estimates are available in our prefetching system and will not address how it is being done in detail.

## 4 Prefetching Algorithms

The benefit of web prefetching is to provide low retrieval latency for users, which can be explained as high hit ratio. Prefetching also increases system resource requirements in order to improve hit ratio. Resources consumed by prefetching include server CPU cycles, server disk I/O's, and network bandwidth. Among them, bandwidth is likely to be the primary limiting factor. So, we add moderate bandwidth requirements as an important performance measure for a good web prefetching model. Our solution space for web prefetching spreads according to these two performance evaluation factors: *hit ratio* and *bandwidth consumption*. There are four algorithms in our solution space together with demand caching and prefetching all objects in the cache. These prefetching algorithms are a set of threshold-based algorithms according to different criteria for choosing which objects to prefetch in advance of requests from users. One of them attempts to maintain contents containing the most popular objects in the system. The second approach mostly considers the network resource demands and tries to prefetch relatively static objects with a less frequent change rate. In this way, it consumes the least bandwidth. The other two algorithms work in different ways, however both attempt to balance the benefit and the cost.

### 4.1 Two Extreme Cases

One extreme is to prefetch all objects into the local cache, which would achieve a 100% cache hit ratio, but would consume huge amounts of

unnecessary bandwidth to fetch and update objects that would never be accessed by clients.

The other extreme is to prefetch nothing where the cache works completely passively on demand. It consumes the least network bandwidth with the lowest hit ratio and yields a large latency for users.

## 4.2 Prefetching by Popularity

In 1998, Markatos et al. proposed a top ten approach for prefetching [17]. Their idea is to keep the top ten popular documents for each web server, by this means, clients or proxy servers can prefetch only these popular documents without significantly increasing network traffic. Their result shows that this approach expects more than 40% of client requests and achieves close to a 60% hit ratio at the cost of increasing network traffic by no more than 10% in most cases.

Their experiment is close to the prefetching by *Popularity* algorithm. The algorithm keeps copies of  $n$  most popular objects in the cache and updates them immediately whenever these objects are modified. From the Zipf-like distribution, we know that popular objects are responsible for majority of requests from users. If we keep in our cache copies of popular objects which are most likely to be requested, this will definitely achieve the highest possible hit ratio. On the other hand, its bandwidth consumption is high.

## 4.3 Prefetching by Lifetime

The lifetime of an object is the interval between two consecutive modifications of object. The longer mean lifetime an object has, the less frequently it changes. The bandwidth cost to update stale objects hence decreases. If we select  $n$  objects with the longest lifetime to replicate in the local cache, we can envision the least network traffic usage.

## 4.4 Prefetching by Good-Fetch

This algorithm was proposed by Venkataramani [2]. It balances object access frequency and object update frequency. One of the problems with prefetching is that the prefetched object may not end up being used or being used before it gets stale. If an object ends up being referenced before it goes stale, it is considered as a *good fetch*. This algorithm calculates the probability of a good fetch of an object. It only prefetches objects whose probability of being accessed before being modified is above a given value. As for object  $i$ , assume the object's lifetime  $l_i$ , its probability of being accessed  $p_i$ , and user request arrival rate denoting how many requests arrive per second is  $a$ . Then the probability of object  $i$  to be accessed before it dies is

$$P_{goodFetch} = 1 - (1 - p_i)^{a \times l_i} \quad (2)$$

where  $a \times l_i$  is the number of requests arriving during the lifetime of object  $i$ ,  $(1 - p_i)^{a \times l_i}$  represents the probability that none of requests arriving during the lifetime of object  $i$  are to object  $i$ . Thus,  $1 - (1 - p_i)^{a \times l_i}$  denotes the probability of object  $i$  to be accessed before it dies.

## 4.5 Prefetching by APL

As above, for object  $i$ , we assume that the object's lifetime is  $l_i$ , its probability of being accessed is  $p_i$ , and user request arrival rate is  $a$ . Then,  $a \times p_i$  is the user request rate for object  $i$ , and  $a \times p_i \times l_i$  represents the number of requests for object  $i$  that arrive before it dies. We propose an algorithm using the  $ap_i l_i$  value to select objects. Objects with the highest value of  $ap_i l_i$  will be included in the prefetching set, meaning those objects with most possible requests will be prefetched in caches.

This algorithm can be modified to emphasize the importance of object popularity as follows. The criterion by which we choose objects thereby changes to  $a(p_i)^n l_i$ . In this way, when  $n > 1$ , this algorithm is working closer

to prefetching by *Popularity*, that is, higher bandwidth is used to improve the response time. When  $n < 1$ , this algorithm is closer to prefetching by *Lifetime* to reduce network bandwidth usage. By using different  $n$  values, prefetching can be made adaptive. When the network has a huge amount of abundant bandwidth, a larger value of  $n$  can be used to fetch more objects. When the network has congestions, a smaller value of  $n$  can be used or prefetching can even be disabled.

We will see in Section 6 that how this threshold-based prefetching by *APL* algorithm can balance between the increased network resource costs and response time reduction.

## 5 Methodology

We use a simulation based on a synthetically generated workload of one million objects to compare the performances of prefetching algorithms in term of hit ratio improvement and network bandwidth cost. The synthetically generated workload experiments enable us to model global object popularities and give a proof of concept for the performances of these threshold-based prefetching algorithms.

### 5.1 Analytical Model

We use the analytical model derived in [2] for our study. In this model, assuming the knowledge of global popularity, a statistical model is used to formulate the steady state hit ratio and bandwidth. The user request model is assumed to follow a Poisson distribution. Request arrival rate is  $a$  requests per second. Each object is indexed by  $i$ . The possibility of  $i$ th popular object to be accessed is inversely proportional to  $i$ . This property is known as Zipf distribution we described above. The  $i$ th object has the corresponding popularity  $p_i = k/i^\alpha$ . The lifetimes of an object are shown distributed exponentially with mean  $l_i$  [18]. Each object is assumed to have a fixed size  $s_i$ .

#### 5.1.1 Steady-State Hit Ratio

As in [2],  $P_{A_i}(t)$  is defined as the probability that the previous access to object  $i$  was  $t$  time units before the present time, and  $P_{B_i}(t)$  is the probability that no updates were done to object  $i$  since its last access  $t$  time units in the past. Then, the probability of a hit on a request to a demand cache is

$$\begin{aligned} P_{hit_d} &= \sum_i p_i \int_0^\infty P_{A_i}(t) P_{B_i}(t) dt \\ &= \sum_i p_i \left( \frac{ap_i l_i}{ap_i l_i + 1} \right) \end{aligned} \quad (3)$$

The fraction  $\frac{ap_i l_i}{ap_i l_i + 1}$  represents the hit ratio among accesses to the object  $i$ . Stated otherwise, this denotes the probability of object  $i$  being fresh when it is being accessed. It is called a *freshness factor* of object  $i$  denoted as  $ff(i)$  in [2].

For threshold-based algorithms, an object  $i$  is always kept fresh by prefetching it whenever there is any change, if its corresponding *Popularity*( $i$ ),  $P_{goodFetch}(i)$ ,  $a(p_i)^n l_i$ , or *Lifetime*( $i$ ) is above its threshold value  $T_p$ . Then we will always have a hit on object  $i$ , while the hit ratio for other objects remains same as  $ff(i)$ . Thus, the hit ratio of threshold-based algorithms is

$$P_{hit_p}(i, T_p) = \sum_i p_i h_i, \text{ where} \quad (4)$$

$$h_i = \begin{cases} 1 & , \text{ if } Threshold(i) > T_p \\ \frac{ap_i l_i}{ap_i l_i + 1} & , \text{ otherwise} \end{cases}$$

$Threshold(i)$  can be *Popularity*( $i$ ),  $P_{goodFetch}(i)$ ,  $a(p_i)^n l_i$ , or *Lifetime*( $i$ ), and  $T_p$  is different for each algorithm.

#### 5.1.2 Steady-State Bandwidth

The estimated steady-state bandwidth consumed by the demand caches [2] is

$$BW_{ss_d} = \sum_i s_i (1 - ff(i)) ap_i \quad (5)$$

For threshold-based prefetching, the steady state bandwidth consumed is

$$BW_{ssp} = \sum_i s_i \times h_i, \text{ where} \quad (6)$$

$$h_i = \begin{cases} 1/l_i & , \text{ if } Threshold(i) > T_p \\ ap_i(1 - ff(i)) & , \text{ otherwise} \end{cases}$$

Similarly  $Threshold(i)$  can be  $Popularity(i)$ ,  $P_{goodFetch}(i)$ ,  $a(p_i)^{nl_i}$ , or  $Lifetime(i)$ , and  $T_p$  is different for each algorithm.

## 5.2 Performance Measurement of Prefetching

Any speculative prefetching will make inaccurate prediction and, therefore, will make more requests than a nonprefetching system that is presented with the same stream of user requests. These extra requests contribute to the costs of prefetching: the extra load imposed on the original servers and the extra network bandwidth consumed. In our work, we focus on quantifying the latter. It is important to evaluate the costs since they can lead to worse performance for prefetching client. The extra bandwidth is what can limit the effectiveness of prefetching. So the key issue with prefetching is to find an appropriate balance between response time improvement and the increase in network resources consumption.

Here we provide a model  $H/B$  for the measurement of their balance, where the hit ratio and network resource consumption of demand cache serve as baseline for comparison. This model defines the ratio of hit ratio improvement over increased bandwidth cost.

$$H/B = \frac{Hit_{prefetching}/Hit_{demand}}{BW_{prefetching}/BW_{demand}} \quad (7)$$

$Hit_{prefetching}/Hit_{demand}$  is the hit ratio improvement of prefetching over *demand caching*, and  $BW_{prefetching}/BW_{demand}$  is network bandwidth increase over *demand caching*. The  $H/B$  ratio is a quantity that shows how much possible advantage the prefetching algorithms can

deliver compared to *demand caching* with respect of their ability to balance hit ratio improvement against increased bandwidth cost. It is always desirable that prefetching can achieve the same amount of latency reduction at as less expense as possible, which yields a higher  $H/B$  value.  $H/B$  model works as a evaluation metric for a given prefetching algorithm to find out, at which prefetching stage, e.g., how many objects to replicate in advance, it can obtain the optimal value if  $H/B$  is concerned. Or it can also be considered as a metric for the comparison between prefetching algorithms. In later sections, we will examine what  $H/B$  values can be achieved by the four prefetching algorithms, which prefetching approach can obtain the best balance of increased resource consumption and improved response time when  $H/B$  is concerned and what it suggests.

We then further develop the  $H/B$  ratio to encourage growth of the hit ratio as follows

$$H^k/B = \frac{(Hit_{prefetching}/Hit_{demand})^k}{BW_{prefetching}/BW_{demand}} \quad (8)$$

Compared with  $H/B$  ratio,  $H^k/B$  never does better in comparison between prefetching algorithms. However,  $H^k/B$  ratio is a better metric for a given prefetching system to decide at which prefetching point it achieves the best benefit/cost balance. By increasing the power of hit ratio improvement ( $k$ ), we increase the significance of hit ratio improvement on evaluating the benefit/cost balance. When measuring with  $H^k/B$ , we consider that even a relatively small hit ratio improvement at cost of a relatively large amount of bandwidth can be justified if there is plenty of spare network resource.

## 6 Experimental Results

In our simulation based on the analytic model, we have a fixed number (one million) of objects that exhibit Zipf popularity distribution with  $\alpha = 0.986$ , and for simplicity, all web objects are with the same size of 10K bytes. We choose

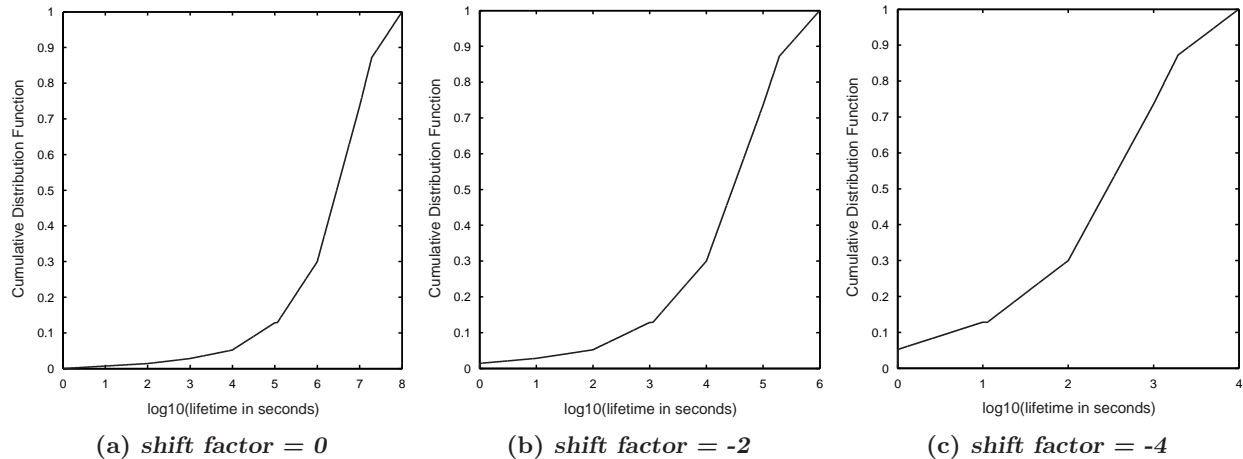


Figure 1: Distribution of object lifetimes

one million as the size of workload since we found many proxy traces studied have roughly the same amount of unique documents [1, 7, 8]. We use  $\alpha = 0.986$  as in [2]. According to our study of the sensitivity of results to  $\alpha$ , all results are consistent as  $\alpha$  changing.  $\alpha = 0.986$  works as a good representative example. We assume infinite cache size, and the number of objects that are prefetched in cache ranges from one through the entire workload.

The cumulative distribution function (CDF) of object lifetimes is shown in Figure 1(a). It is taken from [2] where Venkataramani et al. used as their synthetic workload’s object lifetimes distribution. These objects are relatively stable web resources, with a mean of 3.8 months and median of 33.5 days. To illustrate the sensitivity of our results to the assumptions of lifetimes, we also vary the mean lifetime of objects by shifting the distribution along the lifetime axis across several orders of magnitude. Each represents a change of average lifetime by a factor of 10. Figure 1(b) plots the cumulative distribution function when shifting factor = -2. Note here when shifting left the CDF curve by 2 orders, some portion of objects will get lifetime less than a second, which we believe it is unrealistic. So, we make these lifetimes equal to a second instead of letting them be so small as less than a second, which results in the CDF

shown in Figure 1(b). Figure 1(c) shows the cumulative distribution function when shifting factor = -4. The average object lifetimes are  $9.98 \times 10^6$ ,  $9.98 \times 10^4$ ,  $9.98 \times 10^2$  seconds respectively for shifting factor = 0, -2, and -4.

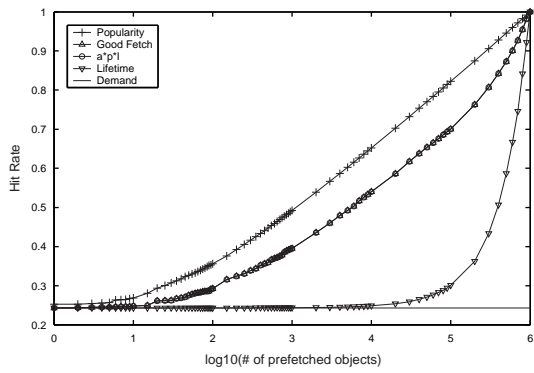
There exists no correlation among object lifetimes, sizes, and popularities [7, 4]. We use a request arrival rate at 1 req/sec. Our analytical results are numerically calculated from the expressions in Section 5.

## 6.1 Threshold-Based Prefetching Algorithms

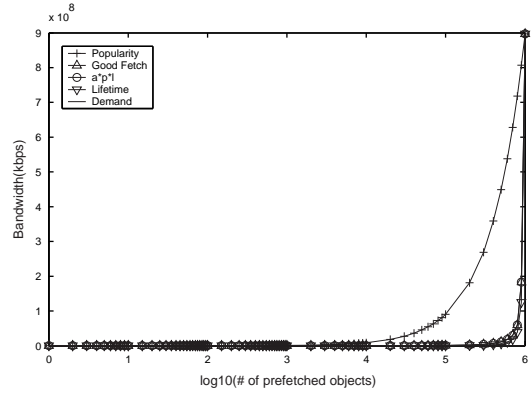
Here, we characterize the nature of four threshold-based prefetching algorithms with respect to their behavior when the number of prefetched objects varies by comparing with demand caches.

### 6.1.1 Steady-State Hit Ratio

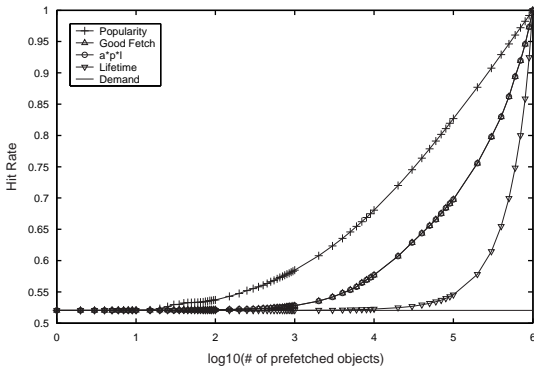
Figure 2 plots the steady state hit ratios that are achieved by five models, *Demand Caching*, prefetching by *Popularity*, *Good-Fetch*, *APL* using  $a(p_i)^n l_i$ , and *Lifetime*. Figure 2 shows our results with different shift factors. The shift factor denotes the horizontal displacement along the lifetime axis (on log scale) of the



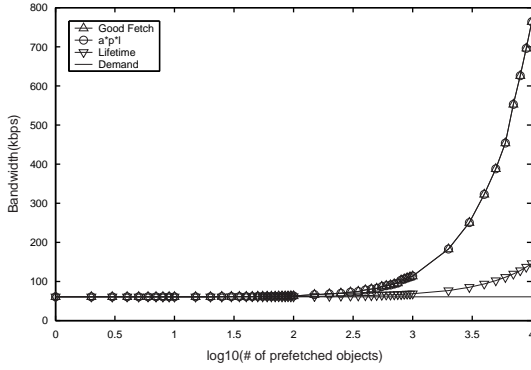
(a) *shift factor = -4*



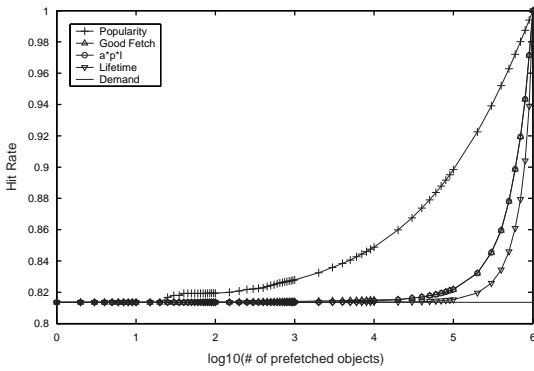
(a) *Bandwidth*



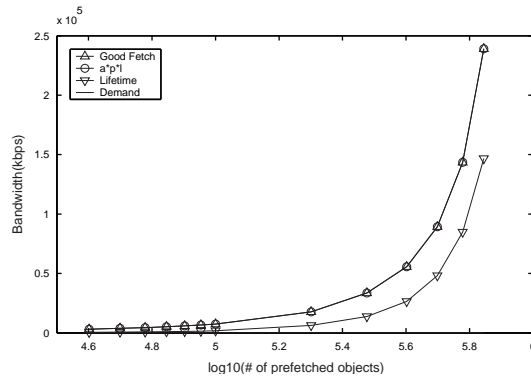
(b) *shift factor = -2*



(b) *Region I*



(c) *shift factor = 0*



(c) *Region II*

Figure 2: Hit Ratio

Figure 3: Bandwidth (shift factor = -4)

probability density function corresponding to the CDF. To focus on the trends across a wide range, the graphs show the hit ratio with a logarithmic scale on the x-axis; the y-axis remains on a linear scale to emphasize the most common values. The hit ratio of demand cache serves as the baseline for comparison of other prefetching methods. From the plots, we see that upon the same number of prefetched objects, prefetching by *Popularity* always gets the highest hit ratio. With the number of prefetched objects going up, all algorithms obtain a higher hit ratio. When all objects in the workload are prefetched in the cache, the hit ratio reaches one for all these four prefetching models.

For prefetching by *APL*, we choose  $n = 1$  as a representative in this subsection. We will show performances of other members in this family when  $n$  takes different values in subsection 6.2. When  $n = 1$ , prefetching by *APL* works very close to prefetching by *Good-Fetch*. Their curves even overlap.

The figures show that when object lifetimes are small, e.g., shift factor = -4, the demand cache gets a lower hit ratio. Thus, we get more hit ratio improvement when prefetching more objects into the cache. For longer lifetimes, demand cache already attains a high hit ratio, e.g., 0.81 for shift factor = 0, there is little space for improvement. At the same time, prefetching by *Good-Fetch* and *APL* ( $n = 1$ ) works closer to prefetching by *Lifetime* at longer object lifetimes. The result is expected because these two approaches both try to balance object access frequency and update frequency. When objects are updated less frequently, these approaches choose preferably longer-lived objects. On the other hand, when average lifetime is shorter, which means that objects are updated quite frequently, the popular objects will be chosen by these two approaches, making them work closer to prefetching by *Popularity*.

Viewed from a different perspective, when a smaller amount of object are allowed to prefetch, e.g., 1% of objects, our results suggest that it is not convincing to use prefetching to

reduce client latency when most of web objects are with relatively long lifetime. As shown in Figure 2(c), most of prefetching could not deliver much advantage to the system if the average object lifetime is long. However, prefetching might be a suitable option for retrieving documents if there are high percentage of short-lived documents.

### 6.1.2 Steady-State Bandwidth

Figure 3(a) presents bandwidth consumption plots of five models with increasing the number of prefetched objects, and Figure 3(b)(c) show the details of the other four except prefetching by *Popularity*. We exclude prefetching by *Popularity* because of its much higher values of bandwidth usage. Here as a representative sample, we just show the plot with shift factor = -4, where mean objects lifetime is about 1000 seconds (several minutes). The remaining plots with different shift factors yield similar results.

The x-axis is on a logarithmic scale and the y-axis is on a linear scale. The bandwidth of demand cache serves as the baseline of 60.12 kbps. When the same number of objects are prefetched, prefetching by *Popularity* consumes the most network bandwidth and much higher than other prefetching algorithms. This is because prefetching by *Popularity* takes no consideration of object update frequency and may select short-lived objects resulting in high bandwidth consumption.

The curves of prefetching by *Good-Fetch*, and by *APL* ( $n = 1$ ) also overlap here. In particular, these two approaches show their effectiveness on balancing the performance improvement and the network resource consumption. They achieve a good overall hit ratio of 40.21% over 24.85% of demand cache at an expense of moderate bandwidth increase (113.23 kbps over demand cache of 60.12 kbps) when about 0.1% of objects are prefetched in the cache. Prefetching by *Lifetime*, which maintains copies of the least mutable documents in cache, receives the lowest hit ratio but costs the smallest

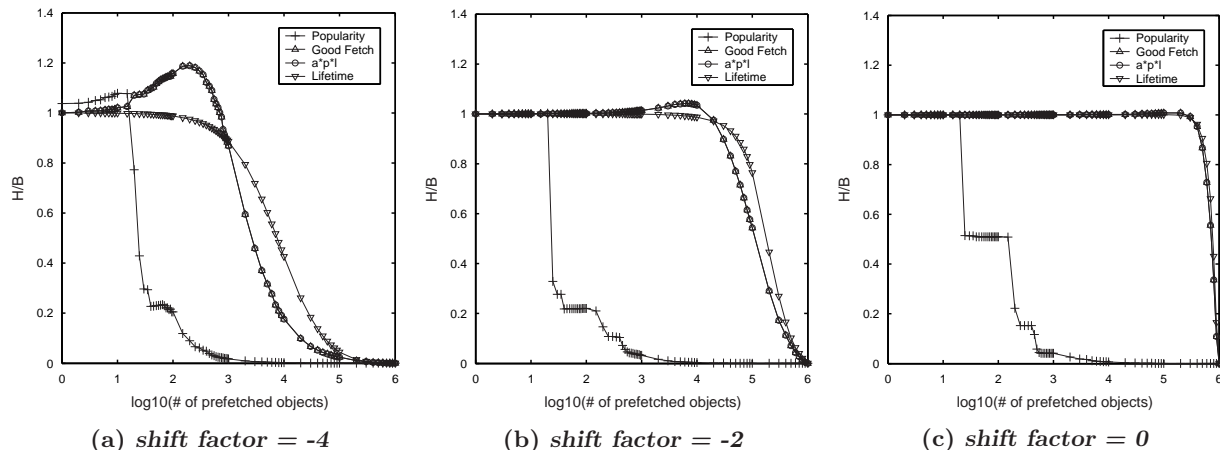


Figure 4:  $H/B$

amount of bandwidth among these prefetching approaches.

### 6.1.3 $H/B$ Measurement

In Figure 4, when the number of prefetched objects is small enough (e.g., less than 20 objects for shift factor = -4), prefetching by *Popularity* gets the highest  $H/B$  ratio, benefit from fetching the most popular objects. But after that, because it is regardless of object consistency, the  $H/B$  ratio drops nearly perpendicularly and would not be comparable with other prefetching methods. Its  $H/B$  behavior remains same when we change the average objects lifetime.

Overall, from the plots we see that prefetching by *Good-Fetch* and *APL* ( $n = 1$ ) attain the highest  $H/B$  value compared to other approaches, which demonstrates that the balance between network resource demands and web latency is effectively achieved by these two methods. After the  $H/B$  peak, they share the same characteristic of dropping with prefetching by *Popularity*, but less steeply. This suggests that for a certain number of prefetched objects, prefetching by *Good-Fetch* and *APL* ( $n = 1$ ) reach the optimal balance point for a specific average objects lifetime. For example, when lifetime shift factor = -4, the  $H/B$

value reaches its highest point at about 200 of prefetched objects (0.02% of total), shown in Figure 4(a). For longer average object lifetime, the  $H/B$  value reaches its highest point at a larger number of prefetched objects, e.g., about 8,000 of prefetched objects for shift factor = -2, and 200,000 for shift factor = 0, but with a lower maximum value of  $H/B$ . This indicates these two prefetching methods tend to support more objects be replicated in a cache when most objects are relatively stable with longer life.

We notice from Figure 4 that the  $H/B$  ratio of prefetching by *Lifetime* starts to decrease all the way from the beginning in a slow pace. With longer object life, prefetching by *Good-Fetch* and by *APL* ( $n = 1$ ) work closer to prefetching by *Lifetime* with respect to  $H/B$  ratio, because the ability of their control over balance between increased bandwidth and improved response time is limited by larger object lifetimes.

One important point worth to be discussed is the crossover of prefetching by *Lifetime* and *APL* in Figure 4. For example, for the shift factor = -2, when the number of the fetched objects are over about 20,000, prefetching by *Lifetime* instead outperforms prefetching by *Good-Fetch* and by *APL* ( $n = 1$ ) by obtaining higher  $H/B$  ratio. This suggests that if we want to fetch a large number of objects, it could be a

good choice if we consider prefetching by *Lifetime* for use when the  $H/B$  ratio is what to be optimized.

## 6.2 Performance of Prefetching by *APL* Family

Now we study the performance of the *APL* algorithm with different  $n$ . We only show plots with a lifetime shift factor = -4 in Figure 5. These plots present comparison among  $n = 0.5, 0.8, 1, 1.5, 2, 5$ . As expected, if  $n$  takes bigger values ( $n > 1$ ), prefetching by *APL* is working closer to prefetching by *Popularity*, and closer to prefetching by *Lifetime* if  $n$  takes smaller values ( $n < 1$ ). For example, as shown in Figure 5(a) and (b), when  $n = 5$ , its hit ratio performance is pretty similar as prefetching by *Popularity*, but bandwidth cost is favorably smaller. By adjusting the value of  $n$  according to the varying available network bandwidth in the system, this family is more flexible and easier to control compared to other prefetching approaches.

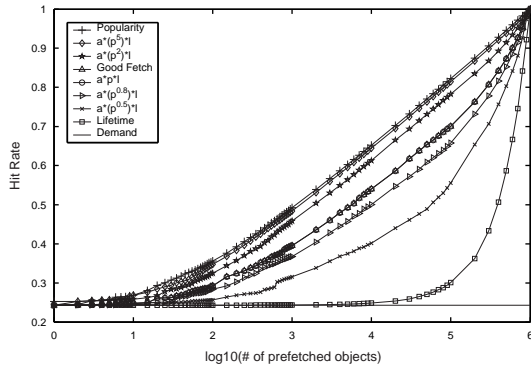
Our results in previous sections show that if  $n = 1$ , prefetching by *APL* works very close to prefetching by *Good-Fetch*, which achieves the highest effectiveness upon the balance of between latency reduction and bandwidth increase. As stated before, prefetching by *Good-Fetch* attempts to maintain the collection of objects with highest probability of being referenced before being modified. On the other hand, prefetching by *APL* ( $n = 1$ ) attempts to maintain the collection of objects with the highest expected number of accesses from users. These two algorithms behave very close although they address the problem from two different points of view. When  $n < 1$ , *APL* family works closer to prefetching by *Lifetime* instead. Based on the same number of objects that the prefetcher is allowed to fetch, *APL* with  $n = 0.8$  achieves a higher hit ratio than  $n = 0.5$  whose hit ratio is closer to that of prefetching by *Lifetime*. We can expect the curve of *APL* family moving towards prefetching by *Lifetime* when we lower down the value of  $n$  towards 0.

Figure 5(c) plots the  $H/B$  ratio of prefetching by *APL* Family. When  $n = 1$ , it obtains the largest maximum value of  $H/B$  ratio among all  $n$ . What the results suggest here is that for *APL* Family, when  $n = 1$  it shows its best effort on maximizing benefit/cost by balancing web latency reduction and bandwidth usage increase. From  $H/B$  point of view, Figure 5(c) shows the importance of the  $n$  parameter in *APL* family that the generalization can be tuned to emphasize fetching long-lived or popular objects.

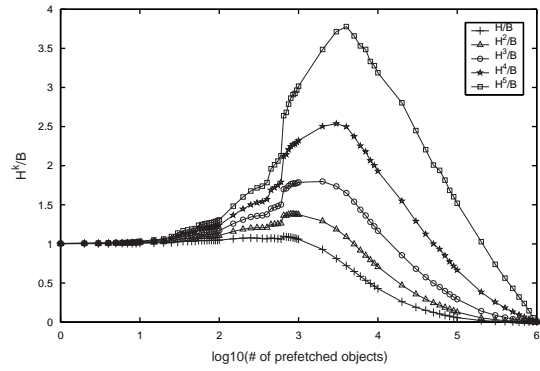
## 6.3 $H^k/B$ Behavior

Compared to  $H/B$ , when performance measurement of a given prefetching scheme is carried out in terms of  $H^k/B$ , we actually increase the proportional significance of hit ratio growth on prefetching performance measure. For example, let us consider as good performance if  $H/B$  ratio is equal to or greater than 1; however, under some circumstances, e.g., a system has plenty of spare bandwidth, we still consider it is justified to double bandwidth to improve hit ratios by 20%, which yields the corresponding  $H/B$  ratio =  $1.2/2 = 0.6$ . At this time, it is not wise to determine whether performance is good or bad by examining  $H/B$  ratio. Instead, we examine  $H^k/B$  ratio, which encourages even a small fraction of improvement on the hit ratio. Thereby, although the  $H/B$  ratio achieved is 0.6, much less than 1,  $H^k/B$  may obtain a value greater than 1 if  $k$  takes an appropriate value, e.g.,  $k = 5$ ,  $(1.2^5)/2 = 1.24 > 1$ . This prefetching performance is considered good when using  $H^5/B$ . The value of  $k$  is set to be greater than 1 to justify a small improvement on hit ratio when there is plenty of available bandwidth in the system. The  $H^k/B$  measure of different prefetching schemes is shown in Figure 6.

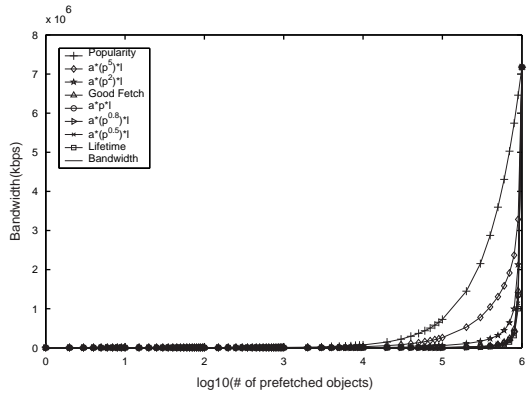
We can also explain it in this way that as for the case of *APL*( $n = 1$ ) in Figure 6(b), in order for  $H^k/B$  to be greater than 1, it can prefetch up to 700 objects when using  $k = 1$ , up to 2,000 with  $k = 2$ , 7,000 with  $k = 3$ , 30,000 with  $k$



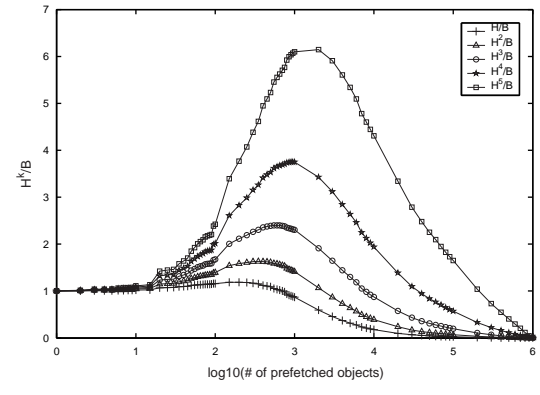
(a) Hit Ratio



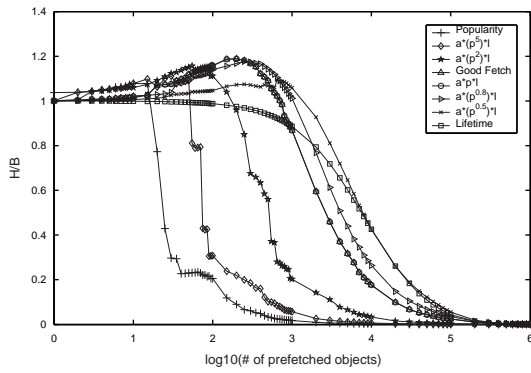
(a)  $APL(n = 0.5)$



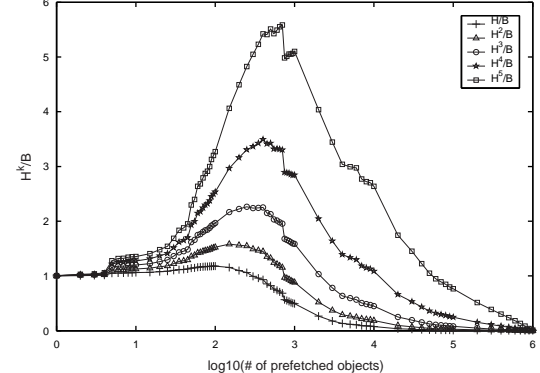
(b) Bandwidth



(b)  $APL(n = 1)$



(c)  $H/B$



(c)  $APL(n = 1.5)$

Figure 5:  $APL$  Family

Figure 6:  $H^k/B$

= 4, and 200,000 with  $k = 5$ . With higher  $k$ , it allows prefetching more objects, which implies that more growth of hit ratio is encouraged. Also note that the peak points of  $H^k/B$  shift right from the peak point of  $H/B$  as  $k$  increasing, which means that if using  $H^k/B$  for performance evaluation, a larger number of prefetched objects are supported at the optimal point.

## 7 Future Work

In our experiment, we ignore the nature of burstiness of web traffic by using a fixed average request arrival rate. This underestimates the hit ratio of demand cache since it ignores the temporal locality. Furthermore, burstiness of request traffic may hurt prefetching at a proxy point if there is no bandwidth available at the peak point of demand traffic. The trace-based simulations of real proxy log trace will be used to measure the performance benefits that might gain from those approaches.

We need also examine the influence of the dynamic popularity distribution model. The popularity of an object varies during its lifetime. Newly appeared objects exhibit typically a steeply rising start peak of user interest. And after these peaks, all objects share a general decrease of user interest in them. The Zipf's like distribution is a static distribution that includes no model for the life cycle of web documents. The Zipf distribution in itself is not well suited to simulated real world data. Because of this, it is not applicable to investigations that consider temporal changes and changes in viewer interest.

A popular approach for prefetching is to prefetch the contents that are related to the content that are recently accessed, e.g., pages whose hyperlink URLs are embedded in the pages that have been just referenced. In this way, the hit ratio can be increased. More study is to be done by including this approach into the algorithms presented in this paper.

## 8 Conclusion

Prefetching is characterized as one of the most efficient schemes to further reduce the user access latency, but runs the risk of increasing network traffic. Most of the approaches attempt to prefetch web objects according to some kind of criteria. We present a solution space for prefetching by examining the performance of these threshold-based algorithms along axes of hit ratio and bandwidth consumption.

When prefetching all objects in a workload into a cache, the hit ratio reaches 1 with enormous bandwidth consumption. Although prefetching by *Popularity* obtains the next highest hit ratio, it also incurs a very high bandwidth requirement. Prefetching by *Good-Fetch* and *APL* are attempting to balance object popularity and object update rate, thereby they can get good hit ratio improvement at a modest expense of bandwidth. In particular, for *APL* family, it could emphasize object popularity when selecting objects to fetch or de-emphasize popularity by altering  $n$ . Our results show prefetching by *APL*, when  $n=1$ , works very close to the prefetching by *Good-Fetch*. The *APL* family can achieve good balance of performance and is flexible for various situations.

Prefetching by *Lifetime* fetches the longest-lived web objects and replicates them in the cache, which requires the least bandwidth in order to keep objects refreshed compared to other prefetching algorithms. At the same time, this approach receives little hit ratio improvement over *Demand Caching*. Ultimately, *Demand Caching* obviously has the lowest hit ratio and the least bandwidth usage.

## References

- [1] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. C. Mogul, *Rate of change and other metrics: a live study of the World Wide Web*. In USENIX Sympo-

- sium on Internet Technologies and Systems, 1997.
- [2] Arun Venkataramani, Praveen Yalagandula, Ravindranath Kokku, Sadia Sharif, Mike Dahlin, *The potential costs and benefits of long-term prefetching for content distribution*. In Sixth International Workshop on Web Caching and Content Distribution, June 2001.
- [3] G. Zipf, *Human behavior and the principle of least effort*, Reading, MA: Addison-Wesley, 1949.
- [4] M. Crovella V. Almeida, A. Bestavros and A. Oliveira, *Characterizing reference locality in the WWW*. In Proceedings of IEEE PDIS' 96: The International Conference in Parallel and Distributed Information System, Miami Beach, FL, December 1996.
- [5] S. Glassman, *A caching relay for the World Wide Web*. In Proceedings of the 1st International Conference of World Wide Web, Geneva, Switzerland, 1994.
- [6] A. Bestavros, C.R. Cunha, and M.E. Crovella, *Characteristics of www client-based traces*. In Technical report, Boston University, July 1995.
- [7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, *Web caching and zipf-like distributions: Evidence and implications*. In Proceedings of IEEE Infocom, 1999.
- [8] N. Nishikawa, T. Hosokawa, Y. Mori, K. Yoshidab, and H. Tsujia, *Memory based architecture with distributed www caching proxy*. In Proceedings of the 7th WWW conference, April 1998.
- [9] M.F. Arlitt and C.L. Williamson, *Web server workload characterization: The search for invariants*. In Proceeding of the ACM SIGMETRICS '96 Conference, Philadelphia, PA, April 1996.
- [10] Bray, T. (1996). *Measuring the web*. In Proceedings of the 5th World Wide Web Conference, pp.993-1005.
- [11] Abdulla, G. (1998). *Analysis and modeling of World Wide Web traffic*. PhD thesis, Virginia Polytechnic Institute and State University.
- [12] Williams, S., Abrams, M., Standridge, C. R., Abdulla, G., and Fox, E. A. (1996). *Removal policies in network caches for World-Wide Web documents*. In Proceedings of the ACM SIGCOMM Conference, pp.293-305.
- [13] National Laboratory for Applied Network Research (NLNR) hierarchical caching system usage statistics – <http://www.ircache.net/Statistics/>.
- [14] Arlitt, M., Friedrich, R., and Jin, T. (1999). *Workload characterization of a web proxy in a cable modem environment*. Technical Report HPL-1999-48, Hewlett Packard Labs.
- [15] M. Crovella and P. Barford, *The network effects of prefetching*. In Proceedings of IEEE Infocom, 1998.
- [16] Thomas M. Kroeger, Darrell D. E. Long, *Exploring the bounds of web latency reduction from caching and prefetching*. In USENIX Symposium on Internet Technologies and Systems, 1997.
- [17] E.P.Markatos and C.E. Chironaki, *A top 10 approach for prefetching the web*. In Proceedings of INET'98: Internet Global Summit, July 1998.
- [18] B. E. Brewington and G. Cybenko, *How dynamic is the web?* WWW9 / Computer Networks, 33(1-6): 257-276, 2000.