Objective-Greedy Algorithms for Long-term Web Prefetching

Bin Wu
Univ. of Illinois at Chicago
bwu@cs.uic.edu

Ajay D. Kshemkalyani
Univ. of Illinois at Chicago
ajayk@cs.uic.edu

Abstract

Web prefetching is based on web caching and attempts to reduce user-perceived latency. Unlike on-demand caching, web prefetching fetches objects and stores them in advance, hoping that the prefetched objects are likely to be accessed in the near future and such accesses would be satisfied from the cache rather than by retrieving the objects from the web server. This paper reviews the popular prefetching algorithms based on Popularity, Good Fetch, APL characteristic, and Lifetime, and then makes the following contributions. (1) The paper proposes a family of prefetching algorithms, Objective-Greedy prefetching, wherein each algorithm greedily prefetches those web objects that give the highest performance as per the metric that it aims to improve. (2) The paper shows the results of a performance analysis via simulations, comparing the objective-greedy algorithms with the existing algorithms in terms of the respective objectives – the hit rate, bandwidth, and the H/B metrics. The proposed prefetching algorithms are seen to provide the best objective-based performance. (3) The paper also proves that the algorithms based on Good Fetch and on the APL characteristic, although using different criteria, are equivalent in terms of their choice of objects selected for prefetching.

1 Introduction

To reduce the user-perceived latency of object requests, web caches are widely used in the current Internet environment. One example is the proxy server that intercepts the requests from the clients and serves the clients with the requested objects if it has the objects stored in it; if the proxy server does not have the requested objects, it then fetches those objects from the web server and caches them, and serves the clients from its cache. Another example is local caching that is implemented in most of the current web browsers. In the simplest cases, those caching techniques may store the most recently accessed objects in the cache and generally use a LRU replacement algorithm that does not take into account the object size and object download cost. Cao and Irani [3] developed a Greedy Dual-Size algorithm that is a generalization of the LRU replacement algorithm to deal with variable object sizes and download times, and it was shown to achieve better performance than most other web cache evicting algorithms in terms of hit rate, latency reduction, and network traffic reduction. However, all these techniques use on-demand caching, or short-term caching, and the cached objects are determined by the recent request patterns of the clients. Prefetching, on the other hand, needs to obtain an overall knowledge of the characteristics of all objects within the entire system and maintains copies of some objects in the cache in advance, according to their access frequencies, update intervals, sizes, etc.. Intuitively, to increase the hit rate, we want to prefetch those objects that are accessed most frequently; to minimize the bandwidth consumption, we may want to choose those objects with longer update intervals.

This paper first reviews the popular prefetching algorithms based on *Popularity* [12], *Good Fetch* [14], *APL characteristic* [9], and *Lifetime* [9]. Their performance can be measured using the different criteria discussed. The paper then makes the following contributions.

1. The paper proposes a family of prefetching algorithms directed to improve the performance in terms of the various metrics that each algorithm is aimed at. The metrics are derived from the H/B metric that combines the effect of increasing hit rate (H) and minimizing the extra bandwidth (B) consumed by prefetching. This criterion was first proposed by Jiang et al. [9] as a performance metric in terms of both hit rate and bandwidth usage. Jiang et al. [9] compared different algorithms, i.e., *Prefetch by Popularity* [12], *Good Fetch* [14], *APL characteristic* [9], and *Lifetime* [9], using this criterion, but did not give any algorithm to optimize this criterion.

2. The paper shows the results of a simulation analysis comparing the performance of all the above algorithms in terms of the hit rate, bandwidth, and H/B metrics. Each of the proposed prefetching algorithms is seen to provide the best performance based on the respective prefetching objective. In particular, the best trade-off between increasing hit rate and minimizing the bandwidth is obtained when

the H/B metric is used. Our simulation results show that the proposed *H/B-Greedy algorithm* offers reasonable improvement of the H/B metric over the best known algorithm. The cost of selecting objects based on any objective criterion is $O(|S| \lg |S|)$ (where $|S|$ is the total number of web objects), which is the same as the cost incurred by the previous algorithms [14],[9],[12]. The performance comparison based on the simulations is summarized in Table 1 in Section 5.1.

3. The paper also proves that the algorithms based on *Good Fetch* [14] and on the *APL characteristic* [9], although using different criteria, are equivalent in terms of their choice of objects selected for prefetching.

Section 2 provides an overview of the web object access model and other characteristics. It also reviews the known prefetching algorithms that are based on different object selection criteria. Section 3 gives a simple theoretical analysis of the steady state hit rate and bandwidth consumption, and a description of the H/B metric and the metrics derived from it. The discussion in this section is based on [14],[9]. Section 4 proposes our new *Objective-Greedy prefetching* algorithms based on the objective metrics to be improved. Section 5 presents the simulation results for comparison with other known algorithms. Section 6 theoretically proves the equivalence of the *Good Fetch* and *Prefetch by APL* algorithms. Section 7 gives concluding remarks.

2 Preliminaries

2.1 Web object properties

To determine which objects to prefetch, we need to have some information about the characteristics of web objects such as their access frequencies, sizes, and lifetimes. Researchers have found that the object access model roughly follows Zipf-like distributions [2], which state that p_i, the access frequency of the i^{th} popular object within a system, can be expressed as $\frac{k}{i}$,where k is a constant and $k = \frac{1}{\sum_i \frac{1}{i}}$.

Using Zipf-like distributions to model the web page requests, Glassman [8] found that they fit $\frac{k}{i}$ quite well, based on his investigation of a collection of 100,000 HTTP requests. A better approximation provided by Cunha et al. [1] generalizes the web objects' access frequency p_i as following a form of Zipf-like distribution:

$$p_i = k/i^{\alpha}, \quad where \quad k = \frac{1}{\sum_i \frac{1}{i^{\alpha}}} \tag{1}$$

The value of α varies in different traces. Cunha et al. [1] recommend a value of 0.986 and Nishikawa et al. [13] suggest α = 0.75, based on their 2,000,000 requests access log.

Another characteristic that affects the hit rate and network bandwidth consumption is the web object's lifetime. Web objects are generally dynamic since they are updated from time to time. An access to a cached object that is obsolete would lead to a miss, and in turn, require downloading the updated version from the web server [10],[6]. An object's lifetime is described as the average time interval between consecutive updates to the object. A prefetching algorithm should take into account each object's lifetime in the sense that objects with a longer lifetime are better candidates to be prefetched in order to minimize the bandwidth consumption.

Crovella and Bestavros [4] have shown that the sizes of static web objects follow a Pareto distribution characterized by a heavy tail. Crovella et al. [5] showed that the actual sizes of dynamic web objects follow a mixed distribution of heavy-tailed Pareto and lognormal distribution. Breslau et al. [2] showed that the distribution of object size has no apparent correlation with the distribution of access frequency and lifetime. Using this observation, Jiang et al. [9] simply assumed all objects to be of constant size in their experiment. However, a more reasonable approach assumes a random distribution of object sizes, and is used in this paper.

2.2 Existing prefetching algorithms

Prefetch by Popularity: Markatos et al. [12] suggested a "Top Ten" criterion for prefetching web objects, which keeps in cache the ten most popular objects from each web server. Each server keeps records of accesses to all objects it holds, and the top ten popular objects are pushed into each cache whenever they are updated. Thus, those top-ten objects are kept "fresh" in all caches. A slight variance of the "Top Ten" approach is to prefetch the n most popular objects from the entire system. Since popular objects account for more requests than less popular ones, *Prefetch by Popularity* is expected to achieve the highest hit rate [9].

Prefetch by Lifetime: Prefetching objects leads to extra bandwidth consumption, since in order to keep a prefetched object "fresh" in the cache, it is downloaded from the web server whenever the object is updated. Starting from the point of view of bandwidth consumption, it is natural to choose those objects that are less frequently updated. *Prefetch by Lifetime* [9], as its name indicates, selects n objects that have the longest lifetime, and thus intends to minimize the bandwidth consumption.

Good Fetch: Venkataramani et al. [14] proposed a *Good Fetch* criterion that balances the access frequency and update frequency of web objects. In the *Good Fetch* algorithm, the objects that have the highest probability of being accessed during their average lifetime are selected for prefetching. Assuming the overall object access rate to be a, the frequency of access to object i to be p_i, and the average lifetime of this object to be l_i, the probability that object i is accessed during its lifetime can be expressed as

$$P_{goodfetch} = 1 - (1 - p_i)^{al_i} \tag{2}$$

The *Good Fetch* algorithm prefetches a collection of objects whose $P_{goodfetch}$ exceeds a certain threshold. The intuition behind this criterion is that objects with relatively higher access frequencies and longer update intervals are more likely to be prefetched, and this algorithm tends to balance the hit rate and bandwidth in that it increases the hit rate with a moderate increasing of bandwidth usage. Venkataramani et al. [14] argued that this algorithm is optimal to within a constant factor of approximation. However, it could behave inefficiently under some specific access-update patterns.

APL Algorithm: Jiang et al. [9] provided another approach for choosing prefetched objects. Again by assuming a, p_i, and l_i as in the *Good Fetch* algorithm, they used $ap_i l_i$ as the criterion. Those objects whose $ap_i l_i$ value exceeds a given threshold will be selected for prefetching. The $ap_i l_i$ value of an object i represents the possible number of accesses to this object during its lifetime. The higher this value, the more the chances (and possibly the more the times) this object is accessed during its lifetime. Thus, prefetching such objects seems to have a better effect on improving the overall hit rate. This algorithm also intends to balance the hit rate and bandwidth consumption.

3 Objective metrics

3.1 Steady state hit rate

The access pattern of an object i is assumed to follow the Poisson distribution with the mean access rate being ap_i, and the update interval is assumed to follow the exponential distribution with the mean interval being l_i [14]. As per the analysis in [14], we define $P_{A_i}(t)$ as the probability that the last access occurs within time t in the past and $P_{B_i}(t)$ as the probability that no update occurs within time t in the past. Then, the following hold.

$$P_{A_i}(t) = 1 - e^{-ap_i t} \tag{3}$$

$$P_{B_i(t)} = e^{-t/l_i} \tag{4}$$

For an object i that is not prefetched, a current access is a hit if the last access occurred after the last update, and the probability of an access to be a hit is given as follows.

$$P_{hit}(i) = \int_0^\infty P_{A_i}(t) dP_{B_i}(t) = \frac{ap_i l_i}{ap_i l_i + 1} \tag{5}$$

This probability is the hit rate of an object under on-demand caching and is also named the *freshness factor* of object i, or *f(i)*. For the prefetched objects, the hit rate is 1. So the hit rate of object i is expressed as:

$$h_i = \begin{cases} \frac{ap_i l_i}{ap_i l_i + 1} & , \ i \ is \ not \ prefetched \\ 1 & , \ i \ is \ prefetched \end{cases} \tag{6}$$

The overall hit rate of a prefetching algorithm is thus:

$$Hit_{pref} = \sum_i p_i h_i \tag{7}$$

3.2 Steady state bandwidth consumption

Let s_i be the size of object i. If object i is not prefetched, then only when an access results in a cache miss would this object be retrieved over the web. Thus the bandwidth for this object is $ap_i(1 - f(i))s_i$. If object i is prefetched, then this object is downloaded from its web server to the cache each time it is updated in the server, and the bandwidth is then $\frac{s_i}{l_i}$. So we have the bandwidth consumption for object i [14]:

$$b_i = \begin{cases} ap_i(1 - f(i))s_i & , \ i \ is \ not \ prefetched \\ \frac{s_i}{l_i} & , \ i \ is \ prefetched \end{cases} \tag{8}$$

The total bandwidth for a prefetching algorithm is:

$$BW_{pref} = \sum_i b_i \tag{9}$$

3.3 H/B model

Venkataramani et al. [14] did the performance evaluation of their *Good Fetch* algorithm based on computational simulations as well as trace-based experiments. The effects of prefetching were demonstrated with hit rate improvement and increase of bandwidth under various threshold values. A smaller threshold value for $P_{goodfetch}$ results in a higher hit rate and more bandwidth consumption because more objects are prefetched. According to how important the user-perceived latency is relative to the bandwidth consumption, different threshold values may be adopted in this algorithm. For example, if the waiting time is a critical requirement and there is a great amount of available bandwidth and hardware resources, a smaller threshold value could be chosen to achieve a higher hit rate.

A balanced measure of the prefetching algorithms was proposed by Jiang et al. [9]. It is called the H/B metric and is defined as:

$$H/B = \frac{Hit_{pref}/Hit_{demand}}{BW_{pref}/BW_{demand}} \tag{10}$$

Here, Hit_{pref} and Hit_{demand} are the overall hit rate, with and without prefetching, respectively; BW_{pref} and BW_{demand} are the total bandwidth with and without prefetching.

The H/B metric expresses the ratio of hit rate improvement over the bandwidth increase. It is a quantitative evaluation of the hit rate improvement a prefetching algorithm can bring relative to excessive bandwidth consumption. In addition, a more generalized form, H^k/B [9], can be used to give relative importance to hit rate or bandwidth by varying the value of k.

$$H^k/B = \frac{(Hit_{pref}/Hit_{demand})^k}{BW_{pref}/BW_{demand}} \tag{11}$$

With this form of the H/B metric, $k > 1$ indicates an environment with abundant available bandwidth and hardware

resources and the improvement of hit rate is more favored than the economy on bandwidth. When the network bandwidth is scarce, a smaller k (possibly $k < 1$) is used instead.

Jiang et al. [9] used the H/B metric and its H^k/B form to evaluate the performance of *Good Fetch* [14], *Popularity* [12], as well as the *Lifetime* and *APL* algorithms that they proposed. However, Jiang et al. did not propose any algorithm that would attempt to optimize either the H/B or the H^k/B metric.

4 Objective-Greedy prefetching

In this section, we first discuss some drawbacks of existing algorithms (Section 4.1). In Sections 4.2 to 4.4, we provide a detailed explanation and theoretical analysis of our *Objective-Greedy prefetching* algorithms. This family of algorithms is greedy because each algorithm always chooses to prefetch those objects that would most significantly improve the performance metric it is aimed at. In Section 4.2, we formulate the theory for and then derive the *H/B-Greedy* algorithm that greedily improves the performance as measured by the H/B metric. In Sections 4.3 and 4.4, the *Hit Rate-Greedy prefetching* and the *Bandwidth-Greedy prefetching* algorithms, which are shown to be special cases of the *H^k/B-Greedy prefetching* with k set to infinity and to 0, respectively, are discussed and analyzed.

4.1 Problems with existing algorithms

The existing prefetching algorithms (*Good Fetch* [14] and *APL* [9]) stemmed from some kinds of intuition that balance the hit rate and bandwidth. However, they are actually far from being ideal in choosing the prefetched objects to achieve the optimal performance with regards to hit rate and bandwidth. Consider the *Good Fetch* algorithm, which prefetches those objects that have the highest probability of being accessed during the objects' average lifetime. If the accesses and updates to an object i are alternated (i.e., there is exactly one access during each time interval of updates to object i), the on-demand hit rate is 0 and the bandwidth consumption of this object is $\frac{s_i}{l_i}$; if object i is prefetched, the hit rate of this object is improved to 1 and the bandwidth is unchanged which indicates that object i is an ideal object to prefetch. However, based on the definition of $P_{goodfetch}$, this object is less likely to be selected for prefetching by *Good Fetch* algorithm. The *APL* algorithm, on the other hand, prefers those objects with a larger product of lifetime and access rate. Consider two objects, A and B, of the same size; the access rate for A is 30 accesses/day and that for B is 3 accesses/day, the average life time for A is 1 days and that for B is 15 days. Which object should *APL* choose to prefetch? This algorithm would more likely choose B rather than A, but a careful analysis would tell us that, for better hit rate, we should choose A; for less bandwidth, we should choose B and for a higher H/B metric, we need to know the characteristics of all other objects to make decision!

The reason for the problems mentioned here is primarily due to the fact that the existing prefetching algorithms simply choose to prefetch an object based on its individual characteristics with no consideration for issues having broader impact, such as the following: (1) How much effect would it have on the overall performance, to prefetch a specific object? (2) Between two objects with different access-update characteristics, which one would have greater influence on the total hit rate and bandwidth?

Another problem worth noting is the actual sizes of objects. The existing algorithms simplified the object sizes to be constant. However, a random distribution may reflect a better approximation of object sizes in the Internet.

4.2 *H/B*-Greedy prefetching

Observe that when an object i is prefetched, the hit rate is increased from $\frac{ap_i l_i}{ap_i l_i + 1}$ to 1, which is $\frac{1}{f(i)}$ times that of on-demand caching; the bandwidth of object i is increased from $\frac{s_i}{l_i + \frac{1}{ap_i}}$ to $\frac{s_i}{l_i}$, which is also $\frac{1}{f(i)}$ times the bandwidth of object i under on-demand caching. Prefetching an object leads to the same relative increase on its hit rate and bandwidth consumption.

Recall the H/B metric (equation (10)) that measures the balanced performance of a prefetching algorithm. We observe that this measure uses the hit rate and bandwidth of on-demand caching as a baseline for comparison and as they are constants, the H/B metric is equivalent to:

$$(\frac{H}{B})_{pref} = \frac{Hit_{pref}}{BW_{pref}} = \frac{\sum\limits_i p_i h_i}{\sum\limits_i b_i} \qquad (12)$$

where h_i and b_i are hit rate and bandwidth of object i, respectively, as described in the previous section.

Consider the H/B value of on-demand caching:

$$(\frac{H}{B})_{demand} = \frac{Hit_{demand}}{BW_{demand}} = \frac{\sum\limits_i p_i f(i)}{\sum\limits_i \frac{s_i}{l_i} f(i)} \qquad (13)$$

What a prefetching algorithm does is choose an appropriate subset of objects from the entire collection – for each of those prefetched objects, say object i, we simply change the corresponding *f(i)* term to 1 in equation (13). Our *H/B-Greedy* algorithm aims to select a group of objects to be prefetched, such that the H/B metric would achieve a better value than that obtained by any other algorithm. Since the object characteristics such as access frequencies, lifetimes, and object sizes are all known to the algorithm, it is possible that, given a number n, we could select n objects to prefetch such that $(H/B)_{pref}$ reaches the maximum value for the given number n and this maximum value will be greater than the value obtained using any existing algorithm. This optimization problem can be formalized as finding a subset

S' of size n from the entire collection of objects, S, such that $(H/B)_{pref}$ is maximized:

$$S' = \underset{S' \subset S, |S'|=n}{argmax} \left[\left(\frac{H}{B} \right)_{pref} \right]$$
$$= \underset{S' \subset S, |S'|=n}{argmax} \left[\frac{\sum_{i \in S} p_i f(i) + \sum_{j \in S'} p_j (1 - f(j))}{\sum_{i \in S} \frac{s_i}{l_i} f(i) + \sum_{j \in S'} \frac{s_j}{l_j} (1 - f(j))} \right] \quad (14)$$

This is a variation of the maximum weighted average problem described by Eppstein and Hirschberg [7], which states that given an academic record of a student's curriculum, select n courses (each with different score and credit hours), from his curriculum that generate the highest GPA. Eppstein and Hirschberg gave an algorithm to solve this problem with time complexity of $O(n)$. However, we did not find any literature that gives a polynomial time solution to the optimization problem we mentioned above, that is equivalent to the case in which one or more prespecified courses *must be* included in the GPA calculation. Alternatively, we propose our *H/B-Greedy* algorithm as an approximation to the optimal solution in the context of web prefetching.

The *H/B-Greedy* algorithm provided here is designed with an attempt to select those objects that, if prefetched, would have most benefit for H/B. For example, suppose initially no object is prefetched and the H/B value is expressed as $(H/B)_{demand}$. Without loss of generality, assume $(H/B)_{demand}$ to be x; now if we prefetch object j, the H/B value will be updated to:

$$\frac{\sum_{i \in S} p_i f(i) + p_j (1 - f(j))}{\sum_{i \in S} \frac{s_i}{l_i} f(i) + \frac{s_j}{l_j} (1 - f(j))} = x \times \frac{\left(1 + \frac{p_j (1 - f(j))}{\sum_{i \in S} p_i f(i)} \right)}{\left(1 + \frac{\frac{s_j}{l_j} (1 - f(j))}{\sum_{i \in S} \frac{s_i}{l_i} f(i)} \right)}$$
$$= x \times incr(j) \quad (15)$$

Here, *incr(j)* is the factor that indicates the amount by which H/B can be increased if object j is selected. We call *incr(j)* the *increase factor* of object j.

Our *H/B-Greedy* prefetching algorithm thus uses *increase factor* as a heuristic and chooses to prefetch those objects that have the greatest *increase factors*. By sorting all objects according to their *increase factors*, we could simply select to prefetch those n objects with largest *increase factors*. Note here we use the *increase factor* as the selection criteria and we may also set a threshold such that any object with *increase factor* greater than this threshold is selected. This approximation algorithm is reasonable because in real-life prefetching, due to the cache size constraints, the number of prefetched objects n is usually small. Furthermore, the *increase factors* of individual objects are generally very close to 1 because of the large number of objects in the web. Thus the previous selection of prefetched objects does not have significant effect on upcoming selections with regard to optimizing H/B.

The pseudo-code is given below as algorithm HBGP(S, n, a). The *H/B-Greedy* prefetching algorithm takes the set of all objects in the web servers (S), the number of objects to be prefetched (n), and the total access rate (a), as the inputs. Each object i is represented by a tuple $\langle p_i, l_i, s_i \rangle$, where p_i, l_i, and s_i stand for the access frequency, the lifetime, and size of the object, respectively.

Algorithm 1 H/B-Greedy prefetching HBGP(S, n, a)

1: **Inputs:**
2: a set of objects of type $\langle p_i, l_i, s_i \rangle$: S
3: number of objects to be prefetched: n
4: the total access rate: a
5:
6: **for** each object $i \in S$ **do**
7: compute the freshness factor: $f(i) = \frac{ap_i l_i}{ap_i l_i + 1}$
8: **end for**
9: compute the on-demand overall hit rate $Hit_{demand} = \sum_{i \in S} p_i f(i)$
10: computer the on-demand overall bandwidth $BW_{demand} = \sum_{i \in S} \frac{s_i}{l_i} f(i)$
11: **for** each object $i \in S$ **do**
12: compute the increase factor $incr(i)$ as defined in equation (15)
13: **end for**
14: sort objects by $incr(i)$ in descending order
15: set the first n objects (with n largest increase factors) to be *prefetched* and the others to be *on-demand*
16:
17: **Output:**
18: the identifiers of the n objects to be prefetched.

Analysis: The computation of freshness factor $f(i)$ and increase factor $incr(i)$ each takes constant time. Hence, the for-loops in lines (6-8) and lines (11-13) each takes $O(|S|)$ time. The computation in line (9) and line (10) also takes $O(|S|)$ time each. In line (14), we use the randomized quick sort strategy for sorting $|S|$ objects in order of increase factors, and this sorting operation takes $O(|S| \lg |S|)$ time. Line (15) takes $O(n)$ time ($n < |S|$). Thus the total time complexity of the *H/B-Greedy* algorithm is $O(|S| \lg |S|)$.

The space requirement for this algorithm includes the space for storing $f(i)$, $incr(i)$, and the characteristic triple $\langle p_i, l_i, s_i \rangle$ for each object, which totals up to $O(|S|)$.

4.3 Hit Rate-Greedy prefetching

Sometimes it is desirable to maximize the overall hit rate given the number of prefetched objects, n. Jiang et al.

[9] claimed that *Prefetch by Popularity* achieves the highest possible hit rate. However a special form of our *Objective-Greedy* algorithms would actually obtain higher hit rate than *Prefetch by Popularity*.

Observe that if object i is prefetched, its contribution to the overall hit rate is

$$HR\_contr(i) = p_i(1 - f(i)) = \frac{p_i}{ap_i l_i + 1} \qquad (16)$$

Thus if we choose to prefetch those objects with the largest hit rate contributions, the resulting overall hit rate must be maximized. We call this algorithm *Hit Rate-Greedy* prefetching and it is obtained from our *Objective-Greedy* principle when we try to optimize the overall hit rate as the objective metric. *Hit Rate-Greedy* prefetching is an extreme case of H^k*/B-Greedy* prefetching algorithm: as we care only about the hit rate and the bandwidth is of no importance, we set the value of k in the H^k/B metric to infinity and this metric becomes the hit rate metric.

4.4 Bandwidth-Greedy prefetching

Another optimization problem in prefetching is to minimize the excessive bandwidth consumption, given the number of objects to be prefetched. Intuition would lead us to assume that *Prefetch by Lifetime* obtains the least bandwidth usage [9]. However, by applying our *Objective-Greedy* principle on bandwidth as the objective, we come up with an algorithm that results in even less bandwidth consumption.

Using analogous reasoning to that for the *Hit Rate-Greedy* algorithm, the extra bandwidth contributed by a prefetched object i would be:

$$BW\_contr(i) = \frac{s_i}{l_i}(1 - f(i)) = \frac{s_i}{ap_i l_i^2 + l_i} \qquad (17)$$

Hence if we prefetch those objects with the least *BW_contr(i)* values, we could finally expect the minimum excessive bandwidth for a given number of prefetched objects, n. We call this algorithm *Bandwidth-Greedy* prefetching and it is another extreme case of the H^k*/B-Greedy* prefetching algorithm: if we care only about the bandwidth usage and not the hit rate, we set the exponent k to zero and the H^k/B metric becomes the bandwidth metric.

The performance comparison between the above two proposed algorithms and *Prefetch by Popularity* and *Prefetch by Lifetime* is shown in Section 5.2.

5 Simulation experiment and results

5.1 Evaluation of H/B-Greedy algorithm

In this part, we ran simulation experiments on the following five prefetching algorithms: *Popularity* [12], *Good Fetch* [14], *APL* [9], *Lifetime* [9], and *H/B-Greedy* which was proposed in this paper. The simulation model made the following assumptions.

- Object access frequency p_i follows Zipf's distribution with α=0.75 [9].
- Object size s_i is randomly distributed between 1 and 1M bytes. Unlike the previous experiments which assumed a fixed object size, this assumption aims to capture more realistic data on the performance.
- Object lifetime l_i is randomly distributed between 1 and 100,000 seconds.
- The overall access rate is 0.01/second.

We ran fours sets of simulations, in which the total number of objects was set to 1000, 10,000, 100,000 and 1,000,000, respectively. In each set of simulations, the five prefetching algorithms were experimented with. For each simulation experiment, the number of prefetched objects was varied, and we measured the resulting hit rates, bandwidth consumption, and the H/B metrics. For each setting of parameter values, we ran three runs. The values varied by less than 1% and so we report only the mean values. Figures 1, 2, 3, 4 show the H/B values for a total of 10^3, 10^4, 10^5, and 10^6 objects, respectively.

It is seen from the simulations that *H/B-Greedy* prefetching beats all other algorithms in terms of H/B for any number of prefetched objects. This is because the *H/B-Greedy* prefetching always chooses to prefetch those objects that most significantly improve H/B metric. However, its hit rate is generally lower than that of *Popularity* and *APL* algorithms. An additional advantage of the *H/B-Greedy* prefetching algorithm is that it is also effective in determining how many objects to prefetch such that the globally highest H/B value can be achieved. This is not discussed in any of the previous literature.

Based on our simulations, the comparative performance of each algorithm is summarized in Table 1. Although H/B is the main objective, we have also shown the relative performance in terms of the objectives H and B. A '1' indicates the best performance among the four, and a '4' indicates the worst performance. The graphs for the hit rate and bandwidth are not shown due to space constraints.

In general, *Prefetch by Popularity* attains the highest hit rate, *Lifetime* consumes the least bandwidth, and *H/B-Greedy* prefetching achieves the highest H/B value in all cases. An interesting observation is that in each case, when the number of prefetched objects is small, *H/B-Greedy* prefetching beats *Prefetch by APL* and *Good Fetch* in terms of hit rate as well; however, when this number exceeds some value in each case, *APL* and *Good Fetch* exhibit higher hit rates than our proposed *H/B-Greedy* prefetching.

A little surprisingly, it is found that *Prefetch by Good Fetch* [14] has almost the same performance as *Prefetch by APL* [9] — they appeared to choose the same group of objects. Hence, the simulation results in Figures 1 to

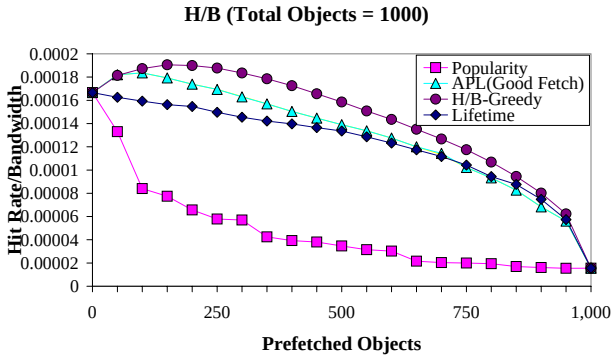

Figure 1. *H/B*, **for a total of 1,000 objects.**

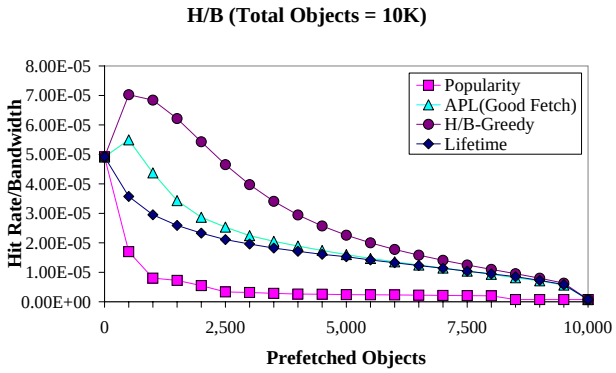

Figure 2. *H/B*, **for a total of 10,000 objects.**

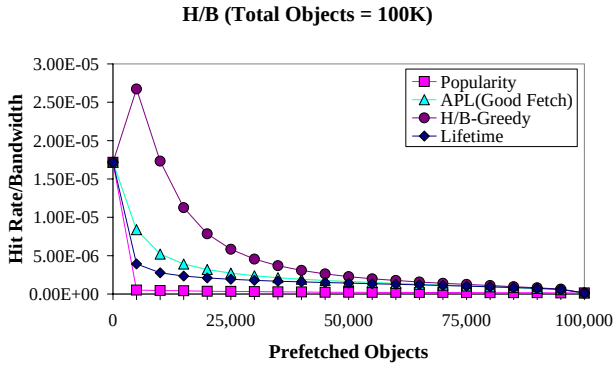

Figure 3. *H/B*, **for a total of 100,000 objects.**

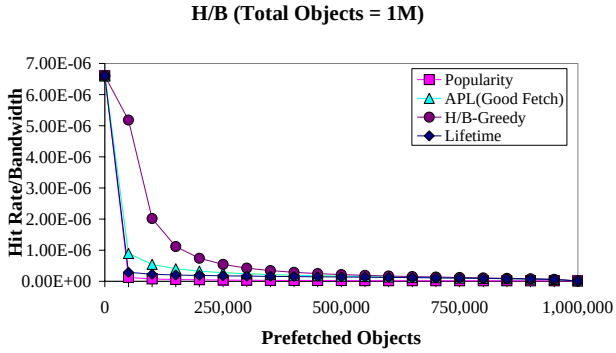

Figure 4. *H/B*, **for a total of 1,000,000 objects.**

4 show the same curves for these two algorithms. Their almost identical behavior prompted us to perform a theoretical analysis of these two criteria, as given in Section 6.

5.2 Evaluation of Hit Rate-Greedy and Bandwidth-Greedy algorithms

In this part, we set up our simulations to investigate the performance of our *Hit Rate-Greedy* and *Bandwidth-Greedy* prefetching algorithms in terms of hit rate and bandwidth, respectively. To show the optimality with regards to their corresponding performance metrics, we only need to compare them with *Prefetch by Popularity* and *Prefetch by Lifetime* (because they were claimed to achieve the highest hit rate and lowest bandwidth, respectively, in [9]).

In this simulation, a total of 1 million objects was assumed; object sizes were randomly distributed between 1 and 1M bytes; object lifetimes were randomly distributed between 1 and 5,000,000 seconds and the total access rate was set to 1.0/second. The results are shown in Figures 5 and 6.

The simulation results show that, as predicted, the *Hit Rate-Greedy* prefetching does achieve higher hit rate than *Prefetch by Popularity* and *Bandwidth-Greedy* prefetching achieves lower bandwidth than *Prefetch by Lifetime.*

6 Equivalence of Good Fetch and APL

Theorem 1 *Algorithm Good Fetch [14] and algorithm Prefetch by APL [9] behave equivalently in terms of their choice of objects to be prefetched.*

Proof. Suppose the objective function (object selection criterion) in *Good Fetch* algorithm to be $f(p,l)$ and that in the *APL* algorithm to be $g(p,l)$. Thus,

$$f(p,l) = 1-(1-p)^{al} \tag{18}$$

$$g(p,l) = apl \tag{19}$$

where p and l are the frequency (access possibility) and lifetime of an object, and a is the total access rate and is treated as a constant.

Consider the derivative of $f(p,l)$ over $g(p,l)$:

$$\begin{aligned}
\frac{df(p,l)}{dg(p,l)} &= \frac{\partial f(p,l)}{\partial p}\frac{\partial p}{\partial g(p,l)} + \frac{\partial f(p,l)}{\partial l}\frac{\partial l}{\partial g(p,l)} \\
&= al(1-p)^{al-1}\frac{1}{al} + \left\{-a\left[(1-p)^{al}\ln(1-p)\right]\right\}\frac{1}{ap} \\
&= (1-p)^{al}\left[\frac{1}{1-p} - \frac{1}{p}\ln(1-p)\right] \\
&= (1-p)^{al}\left[\sum_{i=0}^{\infty}p^i + \sum_{i=0}^{\infty}\frac{p^i}{i+1}\right], (0 \le p \le 1)
\end{aligned} \tag{20}$$

	Hite Rate	Bandwidth	H/B
Popularity [12]	1	4	4
APL [9]/(*Good Fetch)* [14]	2	3	2
Lifetime [9]	4	1	3
H/B-Greedy	3	2	1

Table 1. Performance comparison of algorithms in terms of various metrics. (Lower value denotes better performance.)

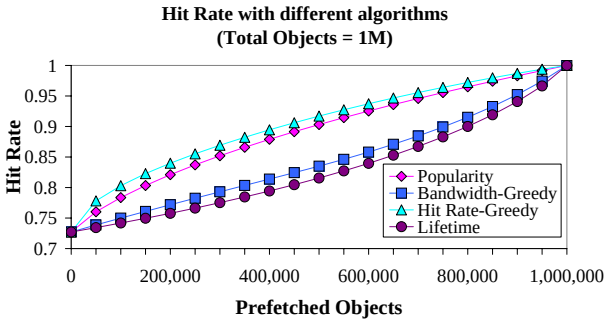

Figure 5. Performance of *Hit Rate-Greedy* algorithm.

It is easy to see that $\frac{df(p,l)}{dg(p,l)} > 0$ always holds for any possible values of p and l, that is, $f(p,l)$ monotonously increases with $g(p,l)$. This property tells us that ranking objects using *Good Fetch* and *APL* criteria results in the same order—they always choose the same group of objects to prefetch, and thus they are equivalent to each other. ∎

7 Conclusions

This paper surveyed several well-accepted web prefetching algorithms. It then proposed a family of algorithms intended to improve the respective prefetching performance metrics under consideration — the hit rate, the bandwidth, or the H/B ratio. Within the family of *Objective-Greedy* algorithms, the *Hit Rate-Greedy* and *Bandwidth-Greedy* prefetching achieve the highest hit rate and lowest bandwidth, respectively. Extensive simulations showed that the new algorithms perform better than all other algorithms in terms of various objective metrics. We also proved that the existing algorithms *Good Fetch* and *Prefetch by APL* are equivalent in terms of their choice of objects to prefetch.

We observe that the *H/B-Greedy* prefetching algorithm proposed in this paper can be modified to derive a good (or near optimal) algorithm with respect to the H^k/B metric [9], where k can be fine-tuned to emphasize either the hit rate or the bandwidth in this performance metric. A strict optimal algorithm may be obtained by a solution to the maximum weighted average problem with pre-selected items.

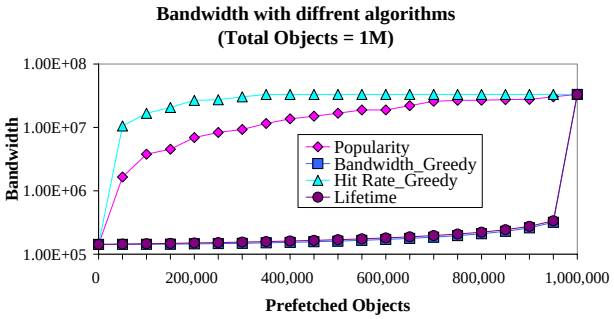

Figure 6. Performance of *Bandwidth-Greedy* algorithm.

References

[1] A. Bestavros, C.R. Cunha, M.E. Crovella, *Characteristics of WWW Client-based Traces*, Technical Report, Boston University, July 1995.

[2] L. Breslau, P. Cao, L. Fan, G. Philips, S. Shenker, Web Caching and Zipf-like Distributions: Evidence and Implications, *Proc. IEEE Infocom*, pp. 126-134, 1999.

[3] P. Cao, S. Irani, Cost-aware WWW proxy caching algorithms, *Proc. USENIX Symp. on Internet Technologies and Systems*, pp.193–206, 1997.

[4] M.E. Crovella, A. Bestavros, Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes, *IEEE/ACM Trans. on Networking*, 5(6):835-746, 1997.

[5] M. Crovella, P. Barford, The Network Effects of Prefetching, *Proc. IEEE Infocom*, pp.1232-1239, 1998.

[6] F. Douglis, A. Feldmann, B. Krishnamurthy, J. C. Mogul, Rate of Change and Other Metrics: A Live Study of the World Wide Web, *Proc. USENIX Symposium on Internet Technologies and Systems*, pp. 147-158, 1997.

[7] D. Eppstein, D.S. Hirschberg, Choosing Subsets with Maximum Weighted Average, *Journal of Algorithms*, 24(1):177-193, 1997.

[8] S. Glassman, A Caching Relay for the World Wide Web, *Computer Networks & ISDN Systems*, 27(2):165-173, 1994.

[9] Y. Jiang, M.Y. Wu, W. Shu, Web Prefetching: Cost, Benefits and Performance, *11^{th} World Wide Web Conference (WWW)*, 2002.

[10] T.M. Kroeger, D.E. Long, J. Mogul, Exploring the Bounds of Web Latency Reduction from Caching and Prefetching, *Proc. USENIX Symposium on Internet Technologies and Systems*, pp.13-22, 1997.

[11] C. Liu, P. Cao, Maintaining Strong Cache Consistency in the World-Wide Web, *Proc. IEEE ICDCS*, pp. 12-21, 1997.

[12] E.P. Markatos, C.E. Chironaki, A Top 10 Approach for Prefetching the Web, *Proc. INET'98: Internet Global Summit*, July 1998.

[13] N. Nishikawa, T. Hosokawa, Y. Mori, K. Yoshidab, H. Tsujia, Memory Based Architecture with Distributed WWW Caching Proxy, *Computer Networks*, 30(1-7):205-214, 1998.

[14] A. Venkataramani, P. Yalagandula, R. Kokku, S. Sharif, M. Dahlin, The Potential Costs and Benefits of Long-term Prefetching. *Computer Communications*, 25(4):367-375, 2002.