The Potential Costs and Benefits of Long-term Prefetching for Content Distribution

Arun Venkataramani   Praveen Yalagandula   Ravindranath Kokku   Sadia Sharif   Mike Dahlin

Technical Report TR-01-13
Department of Computer Sciences
University of Texas at Austin
TAY 2.124, Austin, TX 78712-1188, USA

{{arun,ypraveen, rkoku,dahlin}@cs}, sharif@ece}.utexas.edu

June 30, 2001

Abstract

This paper examines the costs and potential benefits of long-term prefetching for content distribution. In contrast with traditional short-term prefetching, in which caches use recent access history to predict and prefetch objects likely to be referenced in the near future, long-term prefetching uses long-term steady-state object access rates and update frequencies to identify objects to replicate to content distribution locations. Compared to demand caching, long-term prefetching increases network bandwidth and disk space costs but may benefit a system by improving hit rates. Using analytic models and trace-based simulations, we examine several algorithms for selecting objects for long-term prefetching. We find that although the web's Zipf-like object popularities makes it challenging to prefetch enough objects to significantly improve hit rates, systems can achieve significant benefits at modest costs by focusing their attention on long-lived objects.

1 Introduction

In spite of advances in web proxy caching techniques in the past few years, proxy cache hit rates have not improved much. Even with unlimited cache space, passive caching suffers from uncacheable data, consistency misses for cached data and compulsory misses for new data. Prefetching attempts to overcome these limitations of passive caching by proactively fetching content without waiting for client requests. Traditional short-term prefetching at clients uses recent access history to predict and prefetch objects likely to be referenced in the near future and can considerably improve hit rates [8, 9, 18, 21, 32].

In this paper we examine a technique more appropriate for large proxies and content distribution networks (CDNs), namely *long-term prefetching*. Rather than basing prefetching decisions on the recent history of individual clients, long term prefetching seeks to increase hit rates by using global object access patterns to identify a collection of valuable objects to replicate to caches and content distribution servers.

As hardware costs fall, more aggressive prefetching becomes attractive making it possible to store an enormous collection of data at a large content distribution site. For example, in March 2001 an 80GB disk drive cost about $250 [1]. However, maintaining a collection of hundreds of gigabytes or several terabytes of useful web data incurs not just a space cost but also a bandwidth cost: as objects in the collection change, the system must fetch their new versions. It must also fetch newly created objects that meet its selection criteria. Due to the Web's Zipf-like access patterns, a large number of objects must be actively prefetched to improve hit rates significantly [8]. Maintaining such a collection appears to be challenging. In particular, bandwidth expenditure will be the primary constraint in a long term prefetching strategy. For example, in May 2001 a 1.5 Mbps T1 connection cost about $1000 per month [2].

In this paper, we present a model for understanding steady-state cache behavior in a bandwidth-constrained prefetching environment. Our hypothesis is that by prefetching objects that are both long-lived and popular, we can significantly improve hit rates for moderate bandwidth costs. The key contribution of our work is a threshold algorithm for long term prefetching that balances object access frequency and object update frequency and that only fetches objects whose probability of being accessed before being updated exceeds a specified threshold.

Using synthetic and real proxy trace based simulations we establish that our algorithm provides significant hit rate improvements at moderate storage and bandwidth costs. For example for a modest-size cache that receives 10 demand requests per second, long-term prefetching can improve steady state hit rates for cacheable data from about 62% (for an infinite demand-only cache) to above 75% while increasing the bandwidth demands of the system by less than a factor of 2. More generally, we quantify the trade-offs involved in choosing a reasonable prefetch threshold for a given object access rate. Based on our trace based simulation, we conclude that the key challenge to deploying such algorithms is developing good predictors of global access patterns. Although we leave development of such predictors as future work, we provide initial evidence that even simple predictors may work well.

The rest of the paper is organized as follows. Section 2 provides some background information about prefetching and our prefetching model. Section 3 presents the algorithms that we consider for long-term prefetching. Section 4 discusses the methodology we use to evaluate long-term prefetching. Section 5 discusses the results of our simulations and provides insights about how long-term prefetching works. Section 6 discusses related work. Section 7 summarizes our conclusions.

2 Background

This section describes five key parameters of web workloads that determine the effectiveness of caching and prefetching: prefetching models, object popularities, object sizes, object update patterns and lifetimes, and the availability of spare bandwidth for prefetching.

2.1 Prefetching models

We categorize prefetching schemes into two groups: short-term and long-term. In the short-term model, a cache's recent requests are observed and likely near-term future requests are predicted. Based on these predictions, the objects are prefetched. Considerable research has been performed on this type of model [18, 21, 32], most of which are based on variations of a Prediction-by-Partial-Matching (PPM) strategy [16].

In the long-term model of prefetching on which we focus, we assume that a cache or content distribution site maintains a collection of replicated objects based on global access pattern statistics such as object popularity and update rates. We envision a hierarchical structure for content distribution with lower level caches (proxy caches) primarily focusing on servicing client requests and the higher level caches (content distribution servers) on effective content distribution using long-term prefetching. Proxy caches can use short term prefetching to improve hit rates further. Content servers maintain a collection of popular objects and update these objects as they change. New objects are added to the collection based on server assistance and user access.

The content distribution system requires four components:

1. *Statistics tracking.* Our selection algorithm uses as input: (i) estimates of object lifetimes and (ii) estimates of access frequency to objects. Maintaining these estimates is a key challenge to deploying a long-term prefetching based system, and we do not address this problem in detail.

   If content servers are trusted by the content distribution system, they may be able to provide good estimates. Otherwise, the system itself must gather access probability reports from clients or caches and track object update rates. For example, a distributed federation of caches and content distribution nodes could gather local object access distributions and report these statistics to a central aggregation site which would distribute the aggregate statistics to the caches and nodes. There is some evidence that relatively short windows of time can provide good access estimates [25].

2. *Selection criteria.* Based on the statistics, the selection criteria module determines which objects should be included in the replica's collection. The rest of this paper discusses this issue in detail.

3. *Data and update distribution.* The system must distribute objects and updates to objects to caches that include the objects in their collection of replicated objects. We model a push-based system in which updates to replicas are sent immediately to caches that have "subscribed" to the object in question. We leave the details of constructing such a system as future work.

4. *Request redirection.* In order to enable clients to transparently access a nearby copy of a replicated object, an effective redirection scheme is needed. A number of experimental [20, 38] and commercial systems [3, 4] address this issue.

2.2 Popularity distributions

A key parameter for understanding long-term prefetching is the distribution of requests across objects. Several studies [6, 15, 22, 36] have found that the relative distribution with which Web pages are accessed follows a Zipf-like distribution. Zipf's law states that the relative probability of a request for the *i*'th most popular object page is inversely proportional to *i*. Cunha et al. [15] found that the request probability for a Web cache trace, when fitted with a curve of the form $1/i^\alpha$, yields a curve with exponent of $\alpha = 0.982$ which we will use as a default parameter. Other researchers have reached similar conclusions [8].

According to this model, given an universe of N Web pages, the relative probability of *i*th most popular page is

$$p_i \quad = \quad \frac{C}{i^\alpha}, \text{ where } C = \frac{1}{\sum_{k=0}^{N}\left(\frac{1}{k^\alpha}\right)} \tag{1}$$

For our synthetic workload, we will use this model of accesses with $N = 10^9$, $\alpha = .982$, and $C = 0.0389$.

2.3 Object sizes

Studies by Barford and Crovella [14] show that web object sizes exhibit a distribution that is a hybrid of a log-normal and a heavy tailed Pareto distribution. The average size of a web object has been shown to be around 13KB. Work by Breslau et al. [8] suggests that there is little or no correlation between object sizes and their popularity. However, an earlier study by Crovella et al. [15] claims an inverse relationship between object sizes and popularities, *i.e.* users *prefer* small documents. They show a weak Zipf correlation between popularity and size with a zipf parameter -0.33. However we did not observe an appreciable correlation between object sizes and popularity in the Squid traces we analyzed. Hence, for our simulations we do not assume any correlation. It must, however, be emphasized that if the inverse correlation were to be assumed, a prefetching strategy based on maintaining popular objects will perform better with respect to both bandwidth and cache size.

2.4 Update patterns and lifetimes

Web objects have two sources of change - (i) updates to objects that are already present, (ii) introduction of new objects. The work by Douglis et al. [17] shows that (mean) lifetimes of web objects are distributed with an overall mean of about 1.8 months for html files and 3.8 months for image files. Though they analyzed lifetimes for objects in varying popularity classes, little correlation is observed between lifetime and popularity. The work by Breslau et al. [8] further strengthens the case for lack of strong correlation between lifetime, popularity and size of objects.

The lifetime distribution for a single object over time is found to be exponential in [9]. They consider the Internet as an exponentially growing universe of objects with each object changing at time intervals determined by an exponential distribution. Their analysis shows that the age distribution of an exponentially growing population of objects with (identical) exponential age distributions remains exponential with the parameter given by the sum of the population growth and object update rate constants. They then show with respect to the cost of maintaining a collection of fresh popular objects that the introduction of new objects on the Internet is equivalent to changing objects in a static universe of objects with a different rate parameter.

In our simulations we use the data for lifetime distribution presented in [17]. We assume no correlation with popularity or size. Our criterion for selecting an object is based on its current popularity and mean lifetime and is independent of its past and of other objects. We therefore describe our algorithm in terms of the bandwidth cost to update a fixed collection of objects as they are updated. But following analysis done by Brewington et al. [9], our algorithm and analysis also apply to the case of maintaining a changing collection of objects that meet the system's replication selection criteria. We explain this assumption in greater detail in section 4.

2.5 Spare prefetch resources

Prefetching increases system resource demands in order to improve response time. This increase arises because not all objects prefetched end up being used. Resources consumed by prefetching include server CPU cycles, server disk I/O's, and network bandwidth. Therefore, a key issue in understanding prefetching is to determine an appropriate balance between increased resource consumption and improved response time.

Unfortunately, system resources and response time are not directly comparable quantities, and the appropriate balance depends on the value of improved response time and on the amount of "spare" system resources that can be consumed by prefetching without interfering with demand requests. For example, if a system has ample spare bandwidth, it may be justified to, say, quadruple bandwidth demands to improve response time by, say, 20%, but in other circumstances such a trade-off would be unwise.

Often, prefetch algorithms explicitly calculate the probability that a candidate for prefetching will be used. For such algorithms, it is natural to specify a *prefetch threshold* and to prefetch objects whose probability of use exceeds the prefetch threshold. This approach limits the excess resources consumed by prefetching to a factor of at most $\frac{1}{threshold}$ times more resources than a demand system. Note that the total amount of resource expansion may remain significantly below this upper bound because systems may not prefetch all objects and because some objects may attain a higher useful prefetch fractions than enforced by this threshold.

Although determining an appropriate prefetch threshold for a system is challenging, several factors support the position that aggressive prefetching can be justified even if it "wastes" system resources.

- *User time is valuable.* If bandwidth is cheap and human waiting time expensive, prefetching can be justified even if it significantly increases bandwidth demands and only modestly improves response times. For example, Duchamp argues for a prefetch threshold of 0.25 in his hyperlink prefetching system [18], and Chandra et al. [12] argue that thresholds as low as 0.01 may be justified given current WAN network transfer costs and human waiting time values.

- *Technology trends favor increased prefetching in the future.* The prices of computing, storage, and communications fall rapidly over time, while the value of human waiting time remains approximately constant.

- *Prefetch requests may be less expensive to serve than demand requests for the same amount of data.* Servers may schedule prefetch requests to be handled in the background, and networks may benefit from the reduced burstiness of prefetch traffic [14]. Furthermore, techniques such as multicast, digital fountain [10], delta-encoding [30], and satellite links appear well suited to long-term prefetching for content distribution and allow data transmission at a much lower cost than traditional network transmission.

Overall, we conclude that if aggressive prefetching is shown to significantly improve response time, the infrastructure can and will be built to accommodate it.

3 Model and algorithms

In contrast with traditional caching, where space is the primary limiting factor, for long-term prefetching bandwidth is likely to be the primary limiting factor. In this section, we first describe an equilibrium model useful for understanding the dynamics of bandwidth-constrained long-term prefetching. We then describe our algorithms.

3.1 Bandwidth equilibrium

A long-term prefetching system attempts to maintain a collection of object replicas as these objects change and new objects are introduced into the system.

Figure 1 illustrates the forces that drive the collection of fresh objects stored in a cache towards equilibrium. New objects are inserted into the cache by demand requests that miss in the cache and by prefetches. Objects are removed from the set of fresh objects in the cache when servers update cached objects, invalidating the cached copy.[1]

[1] For simplicity, we describe a system in which servers invalidate clients' cached objects when they are updated [13, 26, 29, 37]. Client-polling consistency would yield essentially the same model: in that case, objects that expire are removed from the set of objects that may be accessed without contacting the server.

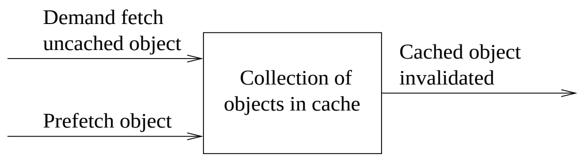

Figure 1: Equilibrium in bandwidth-constrained cache.

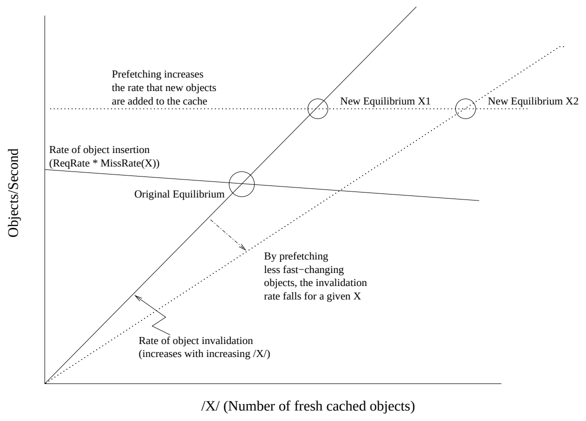

Figure 2: Equilibrium in bandwidth-constrained cache.

The solid lines in Figure 2 illustrates how this equilibrium is attained for a demand-only cache with no prefetching. Let X be the set of fresh objects in the cache at a given moment, and let $|X|$ denote the number of objects in this set. If the number of requests per second being sent to the cache is $ReqRate$, then the rate of object insertion into this set is $ReqRate \cdot MissRate(X)$. For a given request rate, the miss rate typically falls slowly as $|X|$ increases [19, 23], and the rate of insertion falls with it. At the same time the rate of invalidations (or expirations) of cached objects increases as $|X|$ increases. As the figure illustrates, these factors combine to yield an equilibrium collection of objects that can be maintained in the cache.

The dotted lines in Figure 2 illustrate how prefetching can change this equilibrium. First, as the horizontal line illustrates, prefetching increases the rate at which new objects are added to X. If the collection of objects prefetched have similar lifetimes to the collection of objects fetched on demand, then invalidation rates will behave in a similar fashion, and a new equilibrium with a larger set X will be attained as shown by the point labeled *New Equilibrium* X_1.

A prefetching system, however, has another degree of freedom: it can choose what objects to prefetch. If a prefetching system chooses to prefetch relatively long-lived objects, its invalidation rate for a given number of prefetched objects $|X|$ may be smaller than the invalidation rate for the same number of demand fetched objects. This change has the effect of shifting the invalidation rate line down, and yields a new equilibrium, *New Equilibrium* X_2, with $|X_2| > |X_1|$.

A potential disadvantage of preferentially prefetching long-lived objects is that the system may thereby reduce the number of frequently-referenced objects it prefetches. In particular, although $|X_2| > |X_1|$, if the objects in X_1 are more popular than the objects in X_2, the hit rate for equilibrium X_1 may exceed the hit rate for equilibrium X_2.