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

Arun Venkataramani Praveen Yalagandula Ravindranath Kokku Sadia Sharif Mike Dahlin
Department of Computer Sciences
University of Texas at Austin

Abstract

This paper examines the costs and potential benefits of long-term prefetching for content distribution. In traditional short-term prefetching, caches use recent access history to predict and prefetch objects likely to be referenced in the near future. In contrast, 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 algorithms for selecting objects for long-term prefetching. We find that although the Zipf-like popularity distribution of web objects makes it challenging to prefetch enough objects to significantly improve hit rates, systems can achieve significant benefits at modest costs by focusing 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 [7, 8, 15, 16, 25].

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 [7]. 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 [3].

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 Prefetching model

Prefetching schemes can be categorized as short-term and long-term. In the short-term model, objects that are likely to be referenced in the near future are prefetched based on the client's recent access history.In the long term model, objects are prefetched and updated based on long-term global access and update patterns.

We envision a hierarchical structure for a CDN, with caches at higher levels in the hierarchy using long-term prefetching to maintain a collection of objects, to maximize the global hit rates of the CDN for a given bandwidth cost. The caches at the lower levels in the hierarchy can further use short term prefetching to efficiently service client requests, but we do not consider this optimization here. In the rest of the paper, we focus on the long term prefetching model.

A content distribution system using long term prefetching, should address the following challenges:

Statistics tracking. It should keep track of various statistics such as access patterns and object lifetimes. Maintaining these estimates is a key challenge to deploying a long-term prefetching based system, and various strategies could be used to gather and maintain this information. 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 [19].

Object selection criteria. Based on the collected statistics, it should select a good set of objects for prefetching, that can result in significant improvements in hit rate at modest costs.

Distribution of updates. Updates must be distributed to all the caches that include the modified object in their collection of replicated objects. We assume a push-based system in which updates to replicas are sent immediately to caches that are "subscribed" to the object in question.

In this paper, we focus on object selection. We assume the presence of required statistics and a push-based update system.

Prefetching increases system resource demands in order to improve response time. Maintaining a collection of hundreds of gigabytes or several terabytes of useful web data incurs not just a space cost but also a continuous bandwidth cost: as objects in the collection change, the system must fetch new versions of changed objects and must fetch newly created objects. This places a constraint on our long term prefetching model in that the benefits of our prefetching model should justify the continued cost incurred in terms of bandwidth.

Prefetch algorithms often explicitly calculate the probability that a candidate for prefetching will be used in order to control the resource demands. 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.

Several factors support the position that aggressive prefetching can be justified even if it "wastes" system resources. (i) If bandwidth is cheap and human waiting time is expensive, prefetching can be justified even if it significantly increases bandwidth demands and only modestly improves response times [10, 15]. (ii) 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. (iii) prefetch requests may be less expensive to serve than demand requests for the same amount of data. For example, servers may schedule prefetch requests to be handled in the background [12].

We conclude that if aggressive prefetching can be shown to significantly improve response time, the required infrastructure can be built to accommodate it.

3 Prefetching algorithms

A naive popularity-based algorithm identifies the k most popular objects in the universe and maintains copies of them in the cache. Whenever any one of these objects is updated (or a new object joins the set of the most popular k objects), the system fetches the new object into the cache immediately.

This algorithm doesn't work well in presence of popular objects that change frequently. As we explain in [29], to increase hit rate by 10% the popularity algorithm might consume 1000 times more bandwidth than a demand cache. This paper does not consider the simple popularity-based algorithm any further.

3.1 Threshold

The Threshold algorithm that we focus on in this paper, balances object access frequency and object update frequency and only fetches objects whose probability of being accessed before being updated exceeds a specified threshold. In particular, for object i, given the object's expected lifetime $lifetime_i$, the probability that a request will access that object P_i, and the total request rate of demand requests to the cache *requestRate* and assuming that object references and updates are independent, the probability that a cache will access an object that it prefetches before that version of the object dies is

$$P_{goodFetch} = 1 - (1 - P_i)^{lifetime_i \cdot requestRate} \quad (1)$$

where $lifetime_i * requestRate$ is the total number of requests to the cache expected during the object lifetime, and $(1 - P_i)^{lifetime_i \cdot requestRate}$ is the probability that none of these requests access object i.

The Threshold algorithm prefetches the collection of objects whose $P_{goodFetch}$ exceeds a specified threshold. This definition of threshold is similar to that used by several short-term prefetching algorithms [15], and it provides a natural way to limit the bandwidth wasted by prefetching. For each object prefetched, we will consume at most $\frac{1}{threshold}$ times more bandwidth accessing that object than a demand system. Note that the total amount of wasted bandwidth typically remains significantly below this value because some objects will attain a higher useful prefetch fraction than enforced by this threshold.

In [29], we show that the above selection criterion is indeed optimal to within a constant factor of approximation. More formally, given a set of n objects, with a popularity distribution P_i and a lifetime distribution $\lambda_i, 1 \leq i \leq n$, a total bandwidth constraint B and an infinite demand cache, the problem is to select a suitable subset of objects S to keep updated locally in order to maximize hit rate. We show that this problem can be reduced to the Knapsack problem and that the threshold strategy is equivalent to the value-density heuristic for the Knapsack problem.

4 Methodology

We performed a cost-benefit analysis of the Threshold algorithm in terms of improvement in hit rate, bandwidth consumption and cache size through two sets of experiments - (i) an analytical evaluation based on a synthetic workload of a billion objects and (ii) a trace based simulation that compares the performance of different predictors for assessing object popularities. A key benefit of using a synthetic workload is that it allows us to model global object popularities with a known distribution. Thus popularities for all objects are known, even for those which are not seen during a particular simulation run. On the other hand the proxy trace based experiments analyze the performance of adaptive versions of the Threshold algorithm that use various popularity predictors on smaller but realistic workloads. Such workloads exhibit temporal locality between accesses to the same object, have representative object size distributions, and exhibit burstiness in request traffic as observed in real proxy workloads.

4.1 Analytic Model

In this section we develop expressions for calculating the steady state hit rate for a demand cache as well as steady state hit rate, cache size and bandwidth consumption for a prefetch cache with a specified threshold. Due to space constraints we show only the first two derivations. We use a workload model based on a Zipf-like popularity distribution for object popularities where $p_i = \frac{C}{i^\alpha}$ represents the popularity of the ith object and C is a normalization constant [5, 7, 13, 17]. We assume that object lifetimes follow an exponential distribution [8] with l_i representing the average lifetime of object i. Each object i is assumed to have a fixed size s_i. For simplicity, we assume the demand request arrivals follow a Poisson distribution with a mean of a requests per second.

Steady state demand hit rate

Let $P_{A_i}(t)$ represent the probability that the most recent access to object *i* was *t* time units back, and $P_{B_i}(t)$, the probability that no updates were done to object *i* since its last access *t* time units in the past. Now, the probability of a hit on a request to a demand cache is

$$P_{hit_d} \;=\; \sum_i p_i \int_0^\infty P_{A_i}(t) P_{B_i}(t)\,dt, \quad (2)$$

Suppose $P_{(a,t)}(k)$ is the probability that *k* requests occur in any interval of length *t* given an access arrival rate of a. With the assumption of request arrivals following Poisson distribution, the probability of *k* arrivals occurring in *t* seconds is $P_{(a,t)}(k) = e^{-at}.\frac{(at)^k}{k!}$. The probability of no accesses to object *i* in this time *t* is

$$P(0 \text{ accesses to object } i \text{ in } t \text{ time}) \quad =$$

$$
\begin{aligned}
&= \sum_{k=0}^{\infty} P(k \text{ requests in } t) P(\text{none of } k \text{ requests is for } i) \\
&= \sum_{k=0}^{\infty} \left(e^{-at} \frac{(at)^k}{k!} \right) (1-p_i)^k \\
&= e^{-(ap_i)t} \; ,
\end{aligned}
$$

The above equation implies that the inter arrival times to object *i* follow an exponential distribution with mean $(1/ap_i)$. Hence, the probability of an access to an object *i* occurring *t* time units after the most recent access to it is $(ap_i)e^{-(ap_i)t}$, which by definition is $P_{A_i}(t)$. Hence,

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

Given that the lifetimes of an object *i* are exponentially distributed with an average l_i, the probability of no updates to that object happen in time *t* is

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

From Equations 2, 3 and 4, the probability of hit on an access will be

$$
\begin{aligned}
P_{hit_d} &= \sum_i p_i \int_0^{\infty} \left((ap_i)e^{-(ap_i)t} \right) \left(e^{-t/l_i} \right) dt \\
&= \sum_i p_i (ap_i) \left(\left. \frac{e^{-(ap_i+1/l_i)t}}{-(ap_i+1/l_i)} \right|_0^{\infty} \right) \\
&= \sum_i p_i \left(\frac{ap_i l_i}{ap_i l_i + 1} \right) \qquad (5)
\end{aligned}
$$

The term $\frac{ap_i l_i}{ap_i l_i+1}$ represents the fraction of hits among accesses to the object *i*. We call this the *freshness factor* of object *i* and denote it by $ff(i)$.

Steady state prefetch hit rate

In the Threshold algorithm, if an object qualifies for prefetching, *i.e.* its $P_{goodFetch}$ as calculated in Equation 1 is greater than the chosen threshold value *T*. then every access to it results in a hit. For other objects the fractional hit rate remains same as calculated before and is $ff(i)$. Hence, the steady state hit rate in a prefetch based scheme with threshold value *T* is

$$P_{hit_p}(i,T) \;=\; \sum_i p_i h_i, \text{where} \tag{6}$$

$$h_i = \begin{cases} 1 & \text{if } P_{goodFetch}(i) > \text{T} \\ \frac{ap_i l_i}{ap_i l_i + 1} & \text{otherwise} \end{cases}$$

Steady state cache sizes

Proceeding as above, it is straightforward to show that the steady state demand cache size $Csize_{ss_d}$ is given by

$$Csize_{ss_d} \;=\; \sum_i s_i \frac{ap_i l}{ap_i l + 1} \tag{7}$$

When using the Threshold algorithm for prefetching with threshold value T, the estimated total (prefetch + demand) cache size is calculated as

$$Csize_{ss_p} \;=\; \sum_i s_i * f_i, \text{where} \tag{8}$$

$$f_i = \begin{cases} 1 & \text{if } P_{goodPrefetch}(i) > \text{T} \\ \frac{ap_i l}{ap_i l + 1} & \text{otherwise} \end{cases}$$

Steady State Bandwidth

The estimated steady state bandwidth consumed by just demand fetches is

$$BW_{ss_d} \;=\; \sum_i s_i (1 - ff(i)) ap_i \tag{9}$$

For the Threshold algorithm based prefetch strategy, the steady state bandwidth consumed by both prefetch and demand fetches is

$$BW_{ss_p} \;=\; \sum_i s_i * h_i, \text{where} \tag{10}$$

$$h_i = \begin{cases} 1/l_i & \text{if } P_{goodFetch}(i) > \text{T} \\ ap_i(1 - ff(i)) & \text{otherwise} \end{cases}$$

The detailed proofs for all of the above derivations may be found in [29].

4.2 Analytic model parameters

Our analytic results assume a set of 1 billion objects that exhibit a Zipf-like popularity distribution with the Zipf parameter $\alpha = -0.982$ [7, 13]. The sizes of the objects are assumed to follow a distribution given by a *log-normal* [1] body and a *pareto* [2] tail as explained in [6]. We assume that there is no correlation between object sizes and their popularities. The distribution of object lifetimes was taken from [14], again assuming no correlation with popularity or size. This distribution shows a mean lifetime of about 1.8 months and a median of 12.8 days for HTML files, and a mean of 3.8 months and a median of 63.9 days for image files. The analytic results are obtained by numerically evaluating the expressions derived in section 4.1, given these parameters.

All data is considered to be cacheable. This assumption is justified because we are concerned only with the improvement in hit rate due to prefetching, and uncacheable data affects both demand and long term prefetch caching alike. We also speculate that efforts such as active caches [9], active names [27], and ICAP [26] will support execution of

[1] $p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(lnx-\mu)^2/2\sigma^2}$, $\mu = 9.357; \sigma = 1.318$

[2] $p(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$, $k = 133K; \alpha = 1.1$

server programs at caches and content distribution nodes. Moreover, efforts to improve cache consistency semantics [11, 30, 20, 23] will enable caching of much currently uncacheable data.

Our analysis has several limitations. It ignores burstiness of the request traffic and approximates it by a fixed average arrival rate. This assumption directly affects the number of consistency misses as seen by a demand cache. Burstiness of request traffic can also hurt prefetching at a proxy since there may not be any bandwidth available for prefetching at the peak points of demand traffic. On the other hand prefetching can also help by smoothing out demand [12] network traffic. Ignoring temporal locality in the synthetically generated trace underestimates the hit rate seen by the demand cache.

4.3 Proxy trace simulation

The trace based simulator uses a 12 day trace logged by the Squid proxy cache [4] at NCSA, Urbana Champaign between Feb 27 and Mar 10 2001. The trace consists of about 10 million records and accesses to 4.2 million unique objects. We simulate an LRU based demand cache and a prefetch cache implementing the Threshold algorithm using two simple predictors for assessing object popularities. Query URLs (with a ”?” in them) are considered as both uncacheable and unprefetchable.

The sizes of the objects are used as logged in the trace. However, as in the analytic model, lifetimes are generated synthetically from the distribution given in [14], because the traces do not contain object update information. Since our prefetch strategy is sensitive to object lifetimes, this distribution could directly and significantly affect our results. Hence we also perform a sensitivity analysis of the performance of the Threshold algorithm with respect to median object lifetime by shifting the probability distribution curve of the lifetimes by several orders of magnitude along the lifetime axis.

We use a simple statistical model to estimate the popularities of the objects. The simulator maintains a running count of the number of accesses to each object and computes object popularity by dividing this number by the total number of accesses seen. At any access the simulator computes the $P_{goodFetch}$ of the object as defined in Equation 1 and checks if it exceeds the threshold, in which case the access is considered to be a prefetch hit. We call this object popularity predictor as *Predictor1*. *Predictor1* is aggressive, as it uses the current access to update the corresponding object's popularity before computing its $P_{goodFetch}$. This predictor can potentially save compulsory misses to an object since it knows an object's popularity even before the first access to it. Such a predictor is feasible in a scenario where popularity information is aggressively pushed out by the server, or is widely shared across several cooperating caches. We also simulate a more conservative predictor known as *Predictor-2* that can not prevent compulsory misses. An object has to be seen at least once before before its popularity can be used to consider it for prefetching by this predictor. For comparison purposes, we also simulate an oracle *Ever-Fresh* that gets rid of *all* consistency misses, *i.e.*, an access is a hit if the corresponding object has been seen before. The hit rate so obtained is the maximum attainable by a cache based on local prefetching alone. Even the oracle cannot prevent compulsory misses or achieve hits to uncacheable objects. For all of the experiments, the cache was allowed to warm up for 8 days. The long period of warm up will prevent inflated estimates of popularities corresponding to accesses in the beginning of the trace.

As part of future work we intend to measure the performance of more aggressive predictors that gather popularity information from several cooperating caches. In such a scenario, it might be possible to considerably reduce compulsory as well as consistency misses.

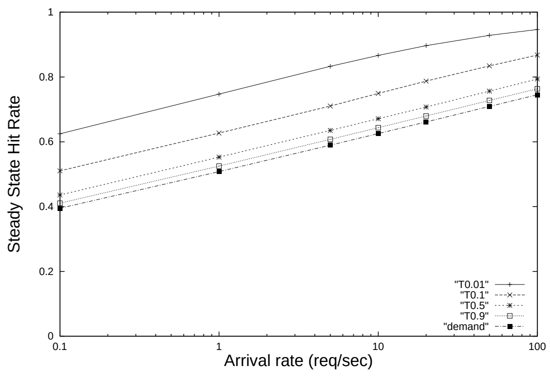
5 Results

5.1 Analytic Evaluation

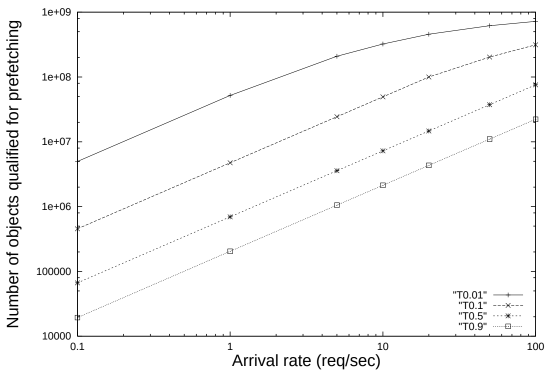
Figure 1(a) plots the hit rates obtained by our prefetching policy for various prefetch thresholds with increasing request arrival rate. Note that demand request arrival rate is a key parameter describing the scale of the content distribution node under consideration and an increase in demand arrival rate increases the set of objects that meet a given threshold. The steady state hit rates for demand cache serve as a baseline for comparing the hit rates obtained by our prefetch policy. The graphs show that the threshold policy improves hit rates. For example, at an arrival rate of 1 req/sec, for threshold 0.5, we get an overall hit rate of 55% compared to a hit rate of 50% obtained by an infinite demand cache in steady state. For an arrival rate of 10 req/sec, for threshold 0.1, we get an overall hit rate of 75% compared to a hit rate of 62% obtained by the demand cache.

Note that significant improvements are achievable across a broad range of CDN scales. Although lower arrival rates reduce the collection of objects that meet the prefetch threshold criteria, lower arrival rates also reduce the steady state hit rates achieved by a demand cache.

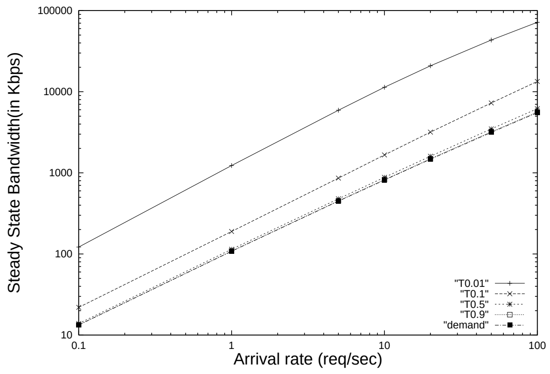
Figure 1(b) plots the total number of objects (from our simulated universe of one billion objects) that qualify to



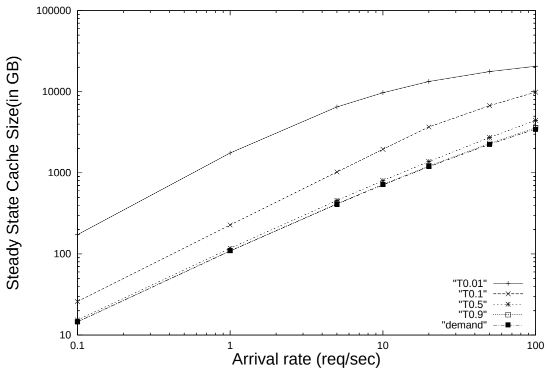
(a) Steady state hit rate



(b) Number of objects prefetched



(c) Steady state bandwidth consumed



(d) Steady state cache size

Figure 1: Prefetching popular long lived objects. Effect of increasing request arrival rates for various thresholds

be prefetched at various threshold values with increasing arrival rates. Figures 1(c) and 1(d) plot the amount of prefetch bandwidth and prefetch cache needed to maintain the prefetched objects.

As seen from the graphs, the choice of a threshold value affects the observed hit rate and the overhead. A high threshold implies a better chance of use of the prefetched object but a decreased hit rate since fewer objects will qualify for prefetching. At the same time it implies a reduced bandwidth and prefetch cache size overhead.

Figure 1(c) suggests that as the arrival rate increases, even low thresholds would incur a modest bandwidth overhead as compared to demand bandwidth. For example, for an arrival rate of 10 req/sec, a threshold of 0.1 would incur bandwidth costs of 1.6Mbps over the demand bandwidth of 800Kbps. From Figure 1(a), this would give us a hit rate of 75%.

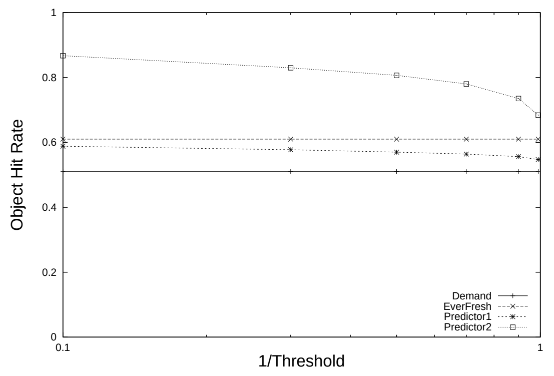
Figure 1(d) helps us in analyzing a typical cache size budget needed at a real world proxy given its request rate. For example, a 10req/sec request rate, with a threshold of 0.1 would correspond to a 2TB prefetch cache size. Given today's disk costs, it would cost around $6400 to add a 2TB disk. From Figure 1(a), a threshold of 0.1 at an arrival rate of 10 req/sec would provide us with a 75% hit rate.

Given falling hardware costs, and the potential to use system-level techniques to decrease cost of handling prefetch requests, even more aggressive prefetching such as T=0.01 might be contemplated in the future.From the graphs, at the cost of 10TB disk and 10Mbps bandwidth, we could achieve 87% hit rate which effectively means a decrease in missrate from 45% to 13%.

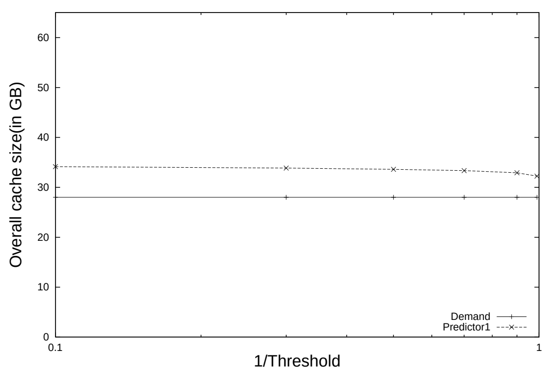
5.2 Trace based simulations

A simple analyis of the trace showed that out of the 10.9 million requests that the proxy received, approximately 10% were consistency check messages. This implies that even an ideal prefetching strategy that prevents all of the consistency misses cannot give an improvement of more than 10% in hit rate over that of an infinite demand cache, unless, its statistics tracing spans multiple caches, or it allows servers to supply popularity estimates when objects are created.

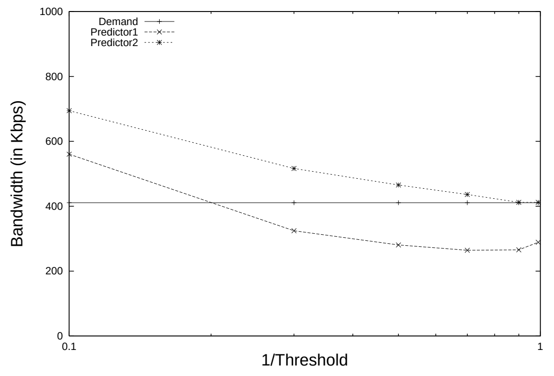
In our experiments we assume a demand cache of size 28GB. We allow the cache to be warmed for 8 days and then gather performance measurements over the remaining 4 days.



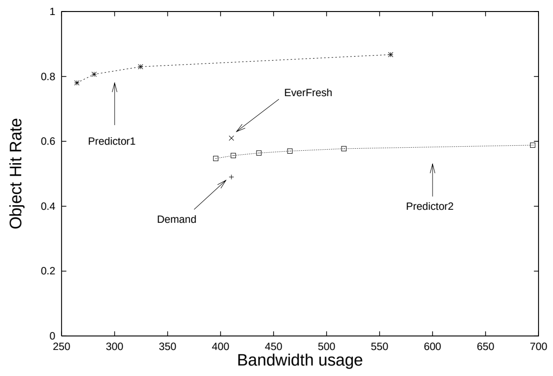
(a) Object Hit rate



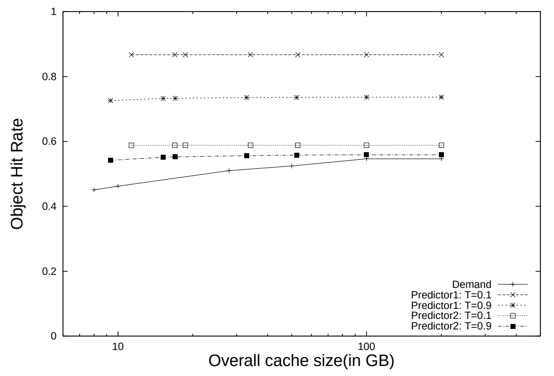
(b) Overall cache size



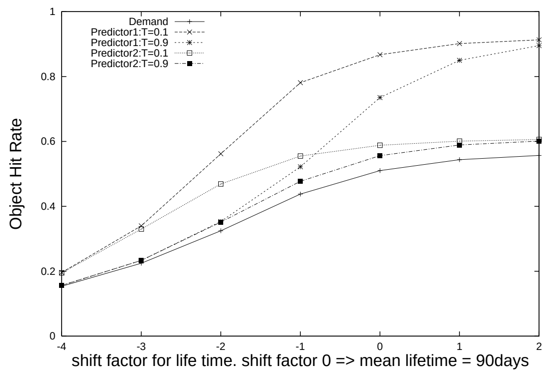
(c) Total Bandwidth



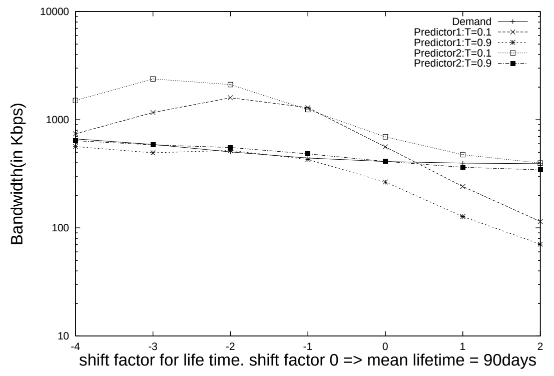
(d) Hit rate vs. Total Bandwidth



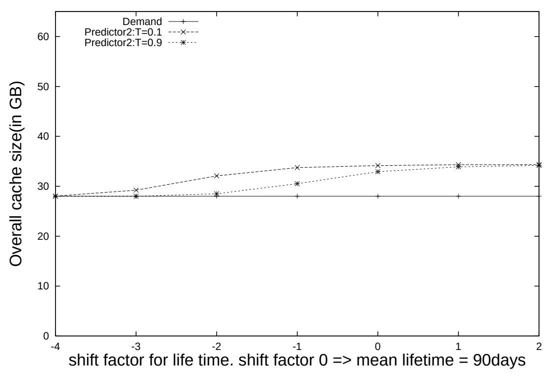
(e) Hit rate vs. Total cache size



(f) Object Hit rate senstivity



(g) Bandwidth sensitivity



(h) Total cache size sensitivity

Figure 2: (a)-(e) Effects of varying the Threshold value on hit rate and bandwidth; (f)-(h) Sensitivity analysis of results

Figures 2(a), (b) and (c) show the change in hit rate, overall bandwidth consumption (demand+prefetch) and overall cache size (demand+prefetch) with varying thresholds. In figure 2(a), the *EverFresh* algorithm gives a hit rate of about 61%. At a threshold of 0.1 *Predictor2* almost matches this ”optimal” achievable hit rate. *Predictor1*, as expected, gives much higher hit rates since it can avoid compulsary misses as well. Figure 2(c) shows that for a threshold of 0.1, the bandwidth blow-up as compared to the demand bandwidth is less than 2X. *Predictor 1* shows lesser bandwidth costs because of the *smoothing* out of prefetched bytes over a larger duration (a typical prefetched object’s lifetime) as opposed to the post-warming period of 4 days for *Demand* and *Predictor2*. Figure 2(b) shows that the increase in cache size due to prefetching is nominal as well. Thus, we conclude that we can obtain significant improvements in hit rate at modest bandwidth and cache space costs.

Figures 2 (d) and (e) show the attainable hit rate with respect to the bandwidth and cache space costs respectively. These graphs have also been generated from the data obtained from experiments performed by varying the threshold and have been included for ease of reference.

To test the sensitivity of our results to our assumptions about lifetimes, we vary the mean life time of objects and study its effect. Figures 2(f)- 2(h) show our results. The x-axis shows a shift factor s which denotes the horizontal displacement along the lifetime axis (on a logscale) of the probability density function corresponding to the CDF taken from [14]. This graph varies the mean lifetime of the objects across several orders of magnitude, with each unit representing a change in the average lifetimes by a factor of 10. The graphs show that when life times are small, we get less hit rate improvement but at the same time use less prefetch cache size and bandwidth. This result is expected because our algorithm does not select short lived objects. At bigger lifetime values, we achieve higher hit rates at a reduced cost of bandwidth and prefetch cache size. The hit rate graphs at high threshold values flatten as the number of unique objects are limited, and almost all the objects would have already qualified for prefetch. This observation holds for a universe of a fixed set of objects; if we are already caching all the ojects, then increase in life times or arrival rates would not alter hit rates. But the bandwidth required to keep the objects refreshed reduces proportionally as the lifetimes increase.

In summary, trace based simulation results show that our prefetch algorithm indeed provides significant hit rate improvements. One limitation of our trace based study is that we chose a medium-sized trace. But given the results we obtained, increasing the trace length would only benefit our results rather than hurt them.

6 Related Work

The idea of prefetching in the web has been widely studied by many researchers recently. Most of the research has concentrated on short-term prefetching based on recent access patterns of clients. Duchamp [15] provides a survey of various research contributions relevant to short-term prefetching in the web: software systems, algorithms, simulations and prototypes [24], and papers that establish bounds [21]. Duchamp [15] proposes to prefetch hyperlinks of web pages based on aggregate access patterns of clients as observed by the server. This approach gives priority to popular URLs and hence is similar to our popularity algorithm. Our (threshold based) approach is different from the above approaches in that we consider long-term prefetching suitable to CDNs [2] and busy proxies by prefetching objects that are both popular and long lived so that we get long-term benefits.

Gwertzman et al. [18] discuss push caching which makes use of a server’s global knowledge of usage patterns and network topology to distribute data to cooperating servers. A number of research efforts support multicast delivery for web content distribution. Li et al. [22] investigate multicast invalidation and delivery of popular, frequently updated objects to web cache proxies. These techniques are complementary to our work and can be used to realize the distribution of updates that we assume in our CDN model.

Implementations of cooperating caches are becoming increasingly commonplace. Sharing of popularity information across caches enables not only better estimates of popularity but can also prevent some compulsory misses. Venkataramani et al. [28] develop a provably near optimal algorithm for placing objects in a set of cooperating caches that are constrained by update bandwidth. Given the network distances between the cooperating caches and the predictions of access rates from each cache to a set of objects, the placement algorithms determine where to place each object in order to minimize the average access cost. The algorithm could be used to extend long-term prefetching to a collection of cooperating caches.

7 Conclusion

In this paper, we focussed on a technique called long-term prefetching, which is beneficial for web proxies and content distribution networks. In this prefetching model, we have explored an algorithm for object selection based on the popularity and lifetime of objects. We evaluated the performance of a long-term prefetching algorithm whose aggressiveness can be tuned by varying a prefetch threshold. For example, our analytical evaluation showed that, for a cache that receives 10 demand requests per second, our prefetching algorithm can improve hit rates by up to 13 percent, while increasing the bandwidth requirements by just over a factor of 2.

The work in this study brings up three key issues which make interesting topics of future research.

1. Exploring the performance of a two level prefetcher composed of a long-term prefetching module that is useful at the entry level while accessing web pages, and a short-term prefetcher that can prefetch hyperlinks there onwards.

2. Exploring mechanisms to estimate the long term effects of prefetching on the underlying network, and the increased opportunities it offers for traffic shaping. resulting from the smoothening of demand network traffic.

3. Extending the threshold algorithm to a set of cooperating caches that can share aggregated popularity information.

An in depth understanding of these issues will form the basis for the desgin of efficient content distribution systems in the future.

References

[1] http://cs.utexas.edu/d̃ahlin/techtrends/data/diskprices/data.

[2] http://www.akamai.com.

[3] http://www.netdoor.com/info/tiert3pricing.html.

[4] http://www.squid-cache.org.

[5] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing Reference Locality in the WWW. In *Proceedings of PDIS'96*.

[6] P. Barford and M. Crovella. Generating representative workloads for network and server performance evaluation, 1998.

[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] B. E. Brewington and G. Cybenko. How dynamic is the web? *WWW9 / Computer Networks*, 33(1-6):257–276, 2000.

[9] P. Cao, J. Zhang, and K. Beach. Active cache: Caching dynamic contents on the web. Technical Report CS-TR-1998-1363, 1998.

[10] B. Chandra, M. Dahlin, L. Gao, A. Khoja, A. Razzaq, and A. Sewani. Resource management for scalable disconnected access to web services. In *WWW10*, May 2001.

[11] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters. In *SIGCOMM98*.

[12] M. Crovella and P. Barford. The network effects of prefetching. In *Proceedings of IEEE Infocom*, 1998.

[13] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of WWW Client-based Traces. Technical Report TR-95-010, Boston University, CS Department, April 1995.

[14] 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 *USITS'97*.

[15] D. Duchamp. Prefetching Hyperlinks. In *USITS99*.

[16] L. Fan, P. Cao, W. Lin, and Q. Jacobson. Web prefetching between low-bandwidth clients and proxies: Potential and performance, 1999.

[17] S. Glassman. A caching relay for the World Wide Web. *Computer Networks and ISDN Systems*, 27(2):165–173, 1994.

[18] J. Gwertzman and M. Seltzer. An analysis of geographical push-caching, 1997.

[19] G Karakostas and D. Serpanos. Practical LFU implementation for Web Caching . Technical Report TR-622-00, Department of Computer Science, Princeton University, 2000.

[20] B. Krishnamurthy and C. Wills. Piggyback Server Invalidation for Proxy Cache Coherency. In *WWW7*, 1998.

[21] T. M. Kroeger, D. E. Long, and J. C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

[22] D. Li and D. R. Cheriton. Scalable web caching of frequently updated objects using reliable multicast. In *Proceedings of USITS'99*.

[23] C. Liu and P. Cao. Maintaining Strong Cache Consistency in the World-Wide Web. In *Proceedings of ICDCS 1997*, May 1997.

[24] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve World-Wide Web latency. In *Proceedings of the SIGCOMM'96*.

[25] T. Palpanas. Web prefetching using partial match prediction, 1998.

[26] The ICAP Protocol Group. Icap the internet content adaptation protocol. Technical Report draft-opes-icap-00.txt, IETF, December 2000.

[27] A. Vahdat, M. Dahlin, T. Anderson, and A. Aggarwal. Active names: Flexible location and transport of wide-area resources. In *USENIX Symposium on Internet Technologies and Systems*, 1999.

[28] A. Venkataramani, M. Dahlin, and P. Weidmann. Bandwidth constrained placement in a WAN. In *To appear in ACM Principles of Distributed Computing*, Aug 2001.

[29] Arun Venkataramani, Praveen Yalagandula, Ravindranath Kokku, Sadia Sharif, and Mike Dahlin. The potential costs and benefits of long term prefetching for content distribution. Technical Report TR-01-13, UT, Austin, 2001.

[30] J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Using Leases to Support Server-Driven Consistency in Large-Scale Systems. In *Proceedings of ICDCS 1998*, May 1998.