On Bandwidth Smoothing[*]

Carlos Maltzahn and Kathy J. Richardson
Compaq Computer Corporation
Network Systems Laboratory
Palo Alto, CA
carlosm@cs.colorado.edu, kjr@pa.dec.com

Dirk Grunwald and James H. Martin
University of Colorado
Department of Computer Science
Boulder, CO
{grunwald,martin}@cs.colorado.edu

Abstract

The bandwidth usage due to HTTP traffic often varies considerably over the course of a day, requiring high network performance during peak periods while leaving network resources unused during off-peak periods. We show that using these extra network resources to prefetch web content during off-peak periods can significantly reduce peak bandwidth usage without compromising cache consistency. With large HTTP traffic variations it is therefore feasible to apply "bandwidth smoothing" to reduce the cost and the required capacity of a network infrastructure. In addition to reducing the peak network demand, bandwidth smoothing improves cache hit rates. We apply machine learning techniques to automatically develop prefetch strategies that have high accuracy. Our results are based on web proxy traces generated at a large corporate Internet exchange point and data collected from recent scans of popular web sites.

1 Introduction

The bandwidth usage due to web traffic can vary considerable over the course of a day. Figure 1 shows the web traffic bandwidth usage at the Palo Alto gateway of Digital Equipment Corporation. The figure shows that peak bandwidth can be significantly higher than the average bandwidth usage. Bandwidth usage varies dramatically each day but the fluctuations are similar each weekday with clearly discernible peak and off-peak periods.

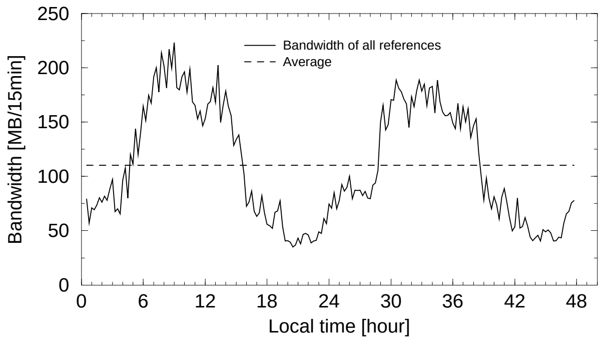


Figure 1: The typical bandwidth usage at the Palo Alto Gateway over the course of two weekdays. Each data point in the graph is the average Kbyte per second byte request rate of a 15 minute interval. Bandwidth usage varies dramatically each day but the fluctuations are similar each weekday with clearly discernible peak and off-peak periods. In this traffic profile the peak/off-peak boundaries lie at around 4:30 AM and 4:30 PM.

These diurnal access profiles are typical for enterprise-level web caches [22, 14, 17].

To perform well, a network needs to accommodate peak bandwidth usages. A reduction in peak bandwidth usage will therefore save network resources including lower demand for DNS, lower server and proxy loads, and smaller bandwidth design requirements.

A common approach to reducing bandwidth usage is to cache web objects as they are requested. Demand-based caching reduces both peak and off-peak bandwidth usage. The effectiveness of this form of caching is limited

[*]For a more detailed version of this paper see [18].

because of the high rate-of-change of large parts of the web content, the size of the web, the working set size, and the object reuse rates [10].

Peak bandwidth usage can also be reduced by shifting some bandwidth from peak periods to off-peak periods. We call this approach *bandwidth smoothing*. In contrast to caching, bandwidth smoothing does not necessarily reduce the daily average bandwidth usage – in fact, it will increase the total bandwidth usage. However, this approach uses unused resources during off-peak to reduce peak bandwidth usage.

Bandwidth smoothing can be accomplished by either appropriately changing user bandwidth usage behavior or by prefetching data during off-peak time. In this paper we will focus on the feasibility of the latter approach.

Web caching is performed by web proxies with caches (web caches) or by routers with attached caches (transparent caches). Both implementations are usually deployed at network traffic aggregation points and edge points between networks of multiple administrative domains. Aggregation points combine the web traffic from a large network user community. This larger user community increases the cache hit rate and reduces latency [11]. Edge points are an opportunity to reduce the bandwidth usage across domains because inter-domain bandwidth is frequently more expensive than bandwidth within domains. Corporate gateways are usually both aggregation points and edge points. We account for this common configuration by basing our feasibility study on data collected from web proxies which are installed at a major Internet gateway of a large international corporation.

Network resources such as bandwidth are frequently purchased in quanta (*e.g.*, a T1 or T3 line). Reducing peak bandwidth usage by less than a quantum may not result in any cost savings. However, small reductions of peak bandwidth usage in many locations of a large organization can aggregate to savings that span bandwidth purchase quanta and can therefore lead to real cost savings. Reducing peak bandwidth requirements also extends the lifetime of existing network resources by delaying the need to purchase the next bandwidth quantum. For new networks, peak bandwidth reduction reduces the bandwidth capacity requirements which allows the purchase of fewer or smaller bandwidth quanta.

The rest of the paper is organized as follows: in the next section we lay out a framework for bandwidth smoothing, analyze the prefetchable bandwidth of the Palo Alto gateway. The following section is on how to measure prefetch performance. In section 4 we demonstrate and discuss the performance of machine learning techniques to automatically develop prefetch strategies. Section 5 is an overview of related work. We conclude with a summary and future work.

2 Prefetchable Bandwidth

The goal of bandwidth smoothing is to shift some of the bandwidth usage from the peak usage periods to off-peak periods. Bandwidth smoothing is a technique that requires caching; prefetched items must be stored in the cache until they are referenced. Furthermore, we assume that items remain in the cache whether they are prefetched or demand fetched by a user. Obviously, cached items no longer need to be prefetched.

The effect of caching needs to be taken into account before smoothing techniques are applied to ensure the effects are additive. We are therefore only interested in "steady-state" bandwidth smoothing where we only study the effect of off-peak prefetching on the *directly following* peak period.

Figure 2 shows cache effects on bandwidth consumption. Caching somewhat smoothes the bandwidth consumption because it reduces the magnitude of the peak bandwidth usage more than the off-peak bandwidth usage. Our measurements also indicate that the hit rate during peak periods is higher than during off-peak periods.

One way to accomplish bandwidth smoothing is to predict peak period cache misses and prefetch the corresponding objects during the preceding off-peak period. The remainder of this section presents definitions and evaluates the prefetch potential and characteristics of prefetchable objects.

2.1 Definitions

We call an object *prefetchable* for a given peak period if it has the following properties: the object

- is referenced during the peak period and was not found in the cache,
- exists during the preceding off-peak time,
- is cacheable, and

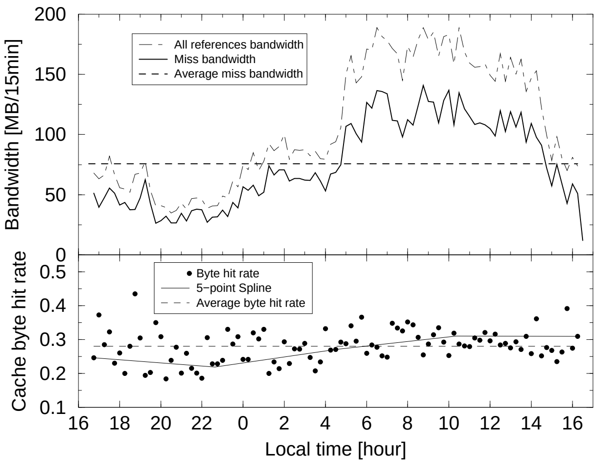


Figure 2: The smoothing effect of traditional caching on bandwidth usage. The cache byte hit rate is higher during the peak period because of more sharing opportunities of a larger network user community.

- is unaltered between the beginning of the preceding off-peak period and the time it is requested.

If an object fails to meet any of these conditions for a given peak period we call it *non-prefetchable* for that peak period. Because of the first condition a non-prefetchable object during one peak period can be a prefetchable object during another peak period. Non-prefetchable objects which meet all prefetchability conditions except the first are called *no-miss-prefetchable* objects.

The combined size of all prefetchable objects of a given peak period is the *prefetchable bandwidth*. There are three disjoint kinds of prefetchable objects, depending on the web access history of the aggregated network user community:

- *Seen prefetchable* objects have names which were previously referenced. These are either *revalidation misses* (caused by stale data) or *capacity misses* (caused by finite-capacity caches).
- *Seen-server prefetchable* objects are referenced for the first time, but they are served from previously accessed web servers. These are *compulsory misses* because they have not been seen before.
- The names and servers of *unseen-server prefetchable* objects were unknown prior to the current reference. Neither the object nor the server were previously accessed. These are also *compulsory misses* because the data has not been seen before

We distinguish *seen-server prefetchable* and *unseen-server prefetchable* objects because predicting the latter kind of objects is more difficult: the proxy access history offers no information about the existence of unseen servers.

2.2 Experimental Measurement and Evaluation Environment

In order to estimate the prefetchable bandwidth for bandwidth smoothing we analyzed the Digital WRL HTTP proxy request traces [15]. The traces cover the days from August 29th through September 22nd, 1996 and consist of over 22 million requests.

To identify prefetchable bandwidth in this trace we need to determine the trace's peak and off-peak periods, cache misses, and the cacheability and object age of each missed object. For the sake of simplicity we divided bandwidth usage into off-peak periods starting at 4:30 PM and ending at 4:30 AM each weekday, and peak periods starting at 4:30 AM and ending at 4:30 PM each weekday (see figure 1, all times in Pacific Daylight Savings Time). We analyzed the HTTP traffic to obtain object age information in order to identify prefetchable objects.

To identify misses in the Digital WRL trace (which was generated by a cacheless web proxy), we assumed (1) a cache of infinite size, (2) a cache hit is represented by a re-reference of an object with the same modification date as the previous reference[1], and (3) the cache never serves a stale object. From this cache model we determine the list of objects and the miss bandwidth for peak and off-peak periods.

To preclude any cache cold-start effects on our measurements we applied this model for at least two days worth of trace data before taking any bandwidth measurements.

2.3 Prefetchable Bandwidth Analysis

Figure 3 shows the composition of the miss bandwidth during a typical weekday peak period of the Digital trace

[1]The modification date must be non-zero. By convention a modification date of zero is used for dynamic and non-cacheable objects, *i.e.* these objects always miss in the cache. Some researchers use modification date and size to differentiate objects [19]. However, we are not aware of any web caches which do not determine object staleness solely based on object age.

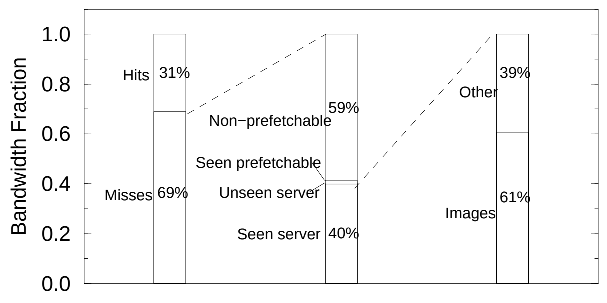


Figure 3: The components of the reference bandwidth. The miss bandwidth was measured for one peak period after a cache warm-up of over two days. The hit rate of the warm cache is 31%. The contribution of seen prefetchables and *unseen-server prefetchables* to the miss bandwidth is negligible.

out of an infinite cache. About 40% of the miss bandwidth is prefetchable.

Almost all the prefetchable bandwidth consists of *seen-server prefetchable* objects. The other prefetchable components (*seen prefetchable* and *unseen-server prefetchable* objects) are negligible. With a fixed size cache the number of *seen prefetchable* objects would increase through capacity misses. *Unseen-server prefetchable* objects are entirely workload dependent and independent of cache configurations. For a bandwidth smoothing prefetching strategy to work, it must rely on web server information in order to discover the names of *seen-server prefetchable* objects. Thus, prefetching involves two subproblems: predicting what to look for (the object selection problem) and predicting where to look (the server selection problem).

Before analyzing the server selection problem we introduce a few definitions that proved to be convenient: We call the prefetchable bandwidth served by a server the *prefetchable service* of this server. A *top prefetchable service group* is a subset of all servers such that the subset consists of servers where each server serves a higher amount of prefetchable bandwidth than any server in the subset's complement. Servers in a *prefetchable service order* are ordered by their prefetchable service.

The server selection problem is simplified by the fact that a small number of servers provide most of the prefetchable items. According to our data, 10% of all servers serve 70% of all prefetchable bandwidth. Figure 4 shows the cumulative prefetchable service distribution over servers in reverse prefetchable service order. These results, however, only show the existence of top prefetchable service groups for a given day. The diurnal bandwidth usage suggests there may be a day-to-day stability of top prefetchable service groups. To verify this conjecture, we tested the following simple heuristic: *if* a server serves prefetchable bandwidth during a given peak period *then* the same server serves prefetchable bandwidth during the following peak period.

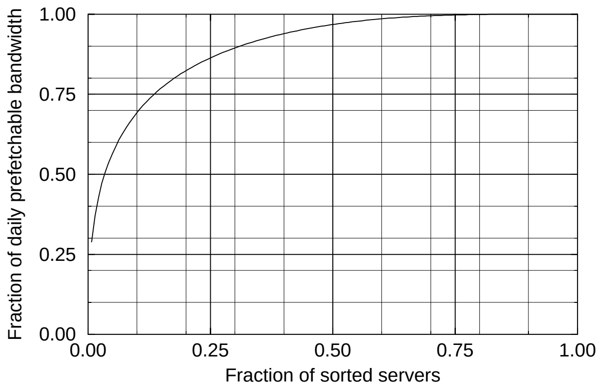


Figure 4: The distribution of prefetchable bandwidth over all servers that served a miss during a single peak period, in reverse prefetchable service order. The measurement was taken after a cache warm-up of over two days. The fact that about 4% of servers serve 50% of all prefetchables simplifies the server selection problem.

We found that unless the top prefetchable service group consists only of a few hundred servers, this simple heuristic fails [18]. There is a large spread of predictability in larger groups which suggests that keeping track of individual server behavior is beneficial. However, tracking prefetchability profiles of a large number of servers is impractical if done manually. In section 4 we investigate the performance of prefetching strategies that are automatically derived from using standard machine learning techniques.

3 Prefetch Performance

In order to exploit prefetchable bandwidth we need to actually prefetch objects from servers according to a prefetch strategy. The performance of a prefetch strategy depends on how closely the set of prefetched objects matches the set of prefetchable objects. In practice, a certain percentage of the prefetched bandwidth consists of objects fetched because of false positive predictions and a certain amount of prefetchable bandwidth is not prefetched because of false negative predictions. Prefetch performance is defined by three measures:

Accuracy $P_a = B_{++}/B_+$ where B_{++} is the prefetchable component of the prefetched bandwidth and B_+ is the entire prefetched bandwidth. P_a represents the fraction of prefetched bandwidth that is prefetchable (also called *prefetch hit rate*).

Coverage $P_c = B_{++}/B_{pre}$ represents the fraction of the prefetchable bandwidth B_{pre} that has been prefetched.

Timeliness P_t represents the fraction of the prefetched prefetchable bandwidth that will not be modified before its miss time. Timeliness is part of our definition of the prefetchability of an object and therefore an implied property of prefetchable bandwidth.

For clarification of the meaning of accuracy and coverage we draw a 2×2 matrix listing the frequency for each outcome of a prefetch prediction:

$$\begin{array}{cc|c|c|} & & \multicolumn{2}{c}{\text{Prefetched}} \\ & & \bar{F} & F \\ \cline{3-4} \text{Prefetchable} & \bar{P} & A & B \\ \cline{3-4} & P & C & D \\ \cline{3-4} \end{array}$$

where P is "prefetchable", $\bar{P}$ "no-miss-prefetchable", F "prefetched", and $\bar{F}$ "not prefetched"; thus $A = \bar{P} \cap \bar{F}$, $B = \bar{P} \cap F$, $C = P \cap \bar{F}$, and $D = P \cap F$. Expressing accuracy and coverage in terms of this frequency matrix we get $P_a = B_{++}/B_+ = D/(B + D)$ and $P_c = B_{++}/B_{pre} = D/(C + D)$.

We now show how these performance metrics relate to bandwidth smoothing. The goal of bandwidth smoothing is to always keep bandwidth consumption below a "target level" which ideally is significantly lower than the peak level without bandwidth smoothing. The lowest possible target level for a particular bandwidth profile depends on the total amount of prefetchable bandwidth, the extra bandwidth available for prefetching during off-peak time, and the prefetch performance.

Figure 5 shows that the extra bandwidth available for prefetching is bounded by the target level. Thus, the definition of the lowest possible target level is recursive: the lower the target level the less extra bandwidth is available for prefetching, which in turn requires higher prefetching performance to maintain the target level. If the target level increases, more extra bandwidth becomes available and less prefetching performance is required to maintain the target level. Because of this counterbalance of target level reduction and extra bandwidth availability, the lowest possible target level is always defined.

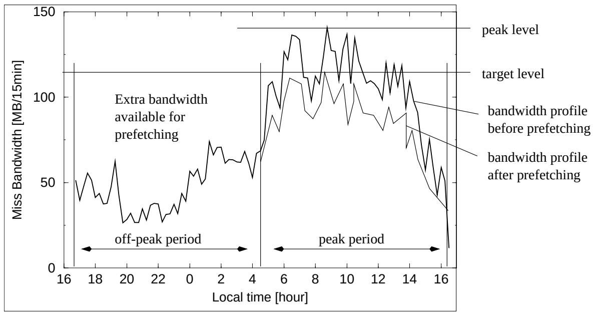


Figure 5: The target level is the peak bandwidth consumption after prefetching (peak of bandwdith profile after prefetching). The extra bandwidth available for prefetching during off-peak times is bounded by the target level. The lowest achievable target level depends (recursively) on the extra bandwidth available for prefetching during off-peak times and the performance of prefetching.

Figure 6 shows minimum target levels of the bandwidth usage profile depicted in figure 2 depending on the prefetchable bandwidth and different accuracy and coverage levels (see [18] for a detailed description on how to calculate minimum target levels). This figure quantifies our earlier qualitative measures: higher accuracy reduces the needed bandwidth for prefetching during off-peak periods while higher coverage increases the bandwidth savings during peak periods. Interestingly, 25% accuracy achieves already over 40% of the maximum peak level reduction. This is due to the aforementioned mutual relationship between target level and extra bandwidth available for prefetching. Our later experiments will show that we can automatically develop prefetch strategies with high accuracy and medium coverage. We developed these tests using a machine learning tool.

4 The Use of Machine Learning for Finding Prefetch Strategies

The content as well as the traffic characteristics of the World-Wide Web is complex, heterogeneous and changes over time as new services become available and new protocol standards replace old ones. To perform well, prefetch algorithms have to be able to adapt to these changing patterns. Other areas of computer systems research with similar problem complexities and similar requirements for adaptability have successfully

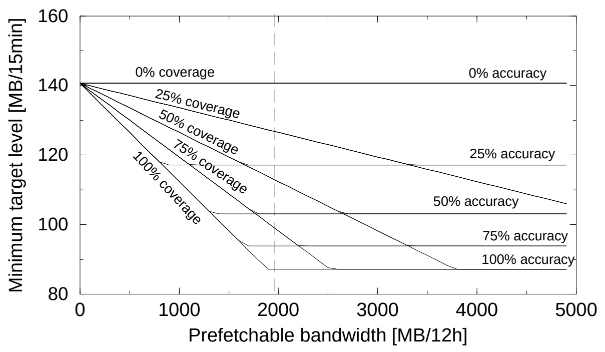


Figure 6: The relationship between prefetchable bandwidth, prefetch performance, and the lowest achievable target level (see [18] on how to calculate minimum target levels). A prefetch performance of zero coverage and zero accuracy means that none of the prefetched objects are prefetchable. For the ideal prefetch performance of 100% coverage and accuracy the target level is at first inversely proportional to the amount of prefetchable bandwidth until the extra bandwidth available for prefetching is exhausted. At that point additional prefetchable bandwidth does not decrease the target level. Generally, coverage determines the fraction of prefetched bandwidth that actually lowers the target level and accuracy determines the maximum amount of prefetchable bandwidth that can be prefetched until the extra bandwidth available for prefetching is exhausted.

applied machine learning techniques (see for example [3]). We therefore investigate the use of machine learning techniques to automatically generate prefetching strategies. Automatic generation allows us to adapt prefetch strategies as often as every day.

The goal of a bandwidth smoothing prefetch strategy is to predict prefetchable objects. As we have shown in section 2.2 almost all prefetchable objects are *seen-server-prefetchable*. It would be very difficult to come up with prefetch strategies that predict the names of prefetchable objects without knowing of their existence. Instead we assume that the prefetch strategy has somehow access to the names of potentially prefetchable objects and is trained to pick from these objects based on their age, MIME-type, size, server name, and top level domain. Clearly, knowing the names of all objects on the Internet is an unrealistic assumption, especially considering the goals of bandwidth smoothing. We have however shown in section 2.2 that a relatively small number of servers serve the majority of prefetchable bandwidth. Thus, we can significantly reduce the number of considered web sites without losing much coverage. Additionally, new services such as [2] allow the fast and compact retrieval of web server content information. We think that our approach becomes more realistic as these services become more available.

4.1 Machine Learning

There are many approaches to machine learning. We used an approach called supervised learning, where the learning algorithm is given a set of input-output pairs (labeled data). The input describes the information available for making the decision and the output describes the correct decision [8]. As we demonstrate below, trace data of web proxy servers and content on the web provide a wealth of labeled data.

The result of a learning task is a classification model (also called a *classifier*) which allows the classification of unseen data below a certain error rate. For generating prefetch strategies we use a tool called RIPPER [4] which efficiently produces and evaluates a classifier in form of a propositional rule set and which is freely available for non-commercial use at [5].

To illustrate what RIPPER does we use a simple learning task from the golf playing domain. First we need to declare the values of attributes and classes:

```
Play, 'No Play'.
outlook: sunny, overcast, rain.
temperature: continuous.
humidity: continuous.
windy: true, false.
```

The first line defines the possible class values (either "Play" or "No Play"). The last four lines define the name and possible values of each attribute that characterizes each training case. Each line in the following training data shows an example of when to play or not to play golf. The first four values correspond to the four attribute definition above, and the last value marks the classification of the example.

```
sunny, 85, 85, false, 'No Play'.
sunny, 80, 90, true, 'No Play'.
overcast, 83, 78, false, Play.
rain, 70, 96, false, Play.
rain, 68, 80, false, Play.
rain, 65, 70, true, 'No Play'.
overcast, 64, 65, true, Play.
sunny, 72, 95, false, 'No Play'.
```

```
sunny, 69, 70, false, Play.
rain, 75, 80, false, Play.
sunny, 75, 70, true, Play.
overcast, 72, 90, true, Play.
overcast, 81, 75, false, Play.
rain, 71, 80, true, 'No Play'.
```

From this data RIPPER constructs a classifier in the form of three rules: two rules on when not to play and one rule that defines the default value as "Play". The first rule means,"if it is windy and rain is to be expected, do not play golf," and the second, "if the humidity is 85% or higher and the outlook is sunny, do not play golf."

```
'No Play' :- windy=true, out-
look=rain (2/0).
'No Play' :- humidity>=85, out-
look=sunny (3/0).
default Play (9/0).
```

The golf data set is a *coherent* data set, *i.e.* none of the examples contradict each other. This is reflected in the parenthesized values after each rule which indicate the number of examples which support the rule and the number of examples which do not match the rule. In a coherent data set the latter is always zero. The real strength of RIPPER is its ability to deal with incoherent data sets with contradicting examples. In this case RIPPER attempts to identify a classifier with a low error rate.

RIPPER also supports the construction of *ensembles of classifiers*. An ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way, usually by weighted voting (see [9] for an overview). Our results include performance data using an ensemble construction algorithm called ADABOOST [13, 12]. The technique is also called *boosting* and works roughly like this: Each classifier is constructed using a "weak learner" such as RIPPER. The difference between the individual classifiers is that they are trained on increasingly more difficult learning problems. The first classifier is learned by the original training data. The next learning problem is constructed by adding weight to the examples which are misclassified by the first classifier. This more difficult learning problem is used to train the next classifier. The examples misclassified by the second classifier receive additional weight in the next learning problem, and so on.

4.2 Training

In order to find a prefetch strategy, one has to find training data consisting of *positive evidence*, *i.e.* examples of objects that should be prefetched, and *negative evidence*, *i.e.* examples of objects that should not be prefetched.

Access log data from web proxy servers provides positive evidence because it includes sufficient information to identify prefetchable objects (in the strict sense of our definition of prefetchable objects). Negative evidence consists of objects that do not meet all prefetchability conditions. Access log data provides some negative evidence as it includes all objects that are missed during a peak period but either didn't exist during the preceding off-peak time, were not cacheable, or were modified between the beginning of the preceding off-peak period and the time it was requested. However, access log data does not include any evidence of *no-miss-prefetchables*, objects that meet all prefetchability conditions except they are not missed during a peak period. This information is only available from a content summary of web servers which include the name and other attributes about each potentially prefetchable object.

There are multiple ways to acquire this information which differ in their impact on bandwidth consumption and their requirements on local services. For example, if the local site also runs a large search engine, the negative evidence can be derived from the search engine index as long as the search index contains information about textual as well as non-textual objects (see for example [2]). This approach has very little impact on bandwidth consumption. If no local search engine is available, a remote search engine could offer a service that allows querying for a very compact representation of the names and attributes of objects with certain properties. Finally, servers themselves could provide such a querying service in some well-known manner.

4.3 Training and Testing Methodology

We collected access log data of 14 days of full gateway traffic at Digital WRL (Monday, 5/18/1998 - Sunday, 5/31/1998). Near the end of the 14 day period, during Friday, 5/29/1998, we identified the top 320 prefetchable service group which serves about 45% of the total prefetchable bandwidth during the 11 day period prior to Friday. Having none of the above services available for efficiently acquiring web server content information, we used a "web robot" to scan these 320 servers during the weekend (5/30-31/1998). The resulting scan contains information on 1,935,086 objects. Limited resources prevented us from scanning more than 320 servers and because of time constraints we were unable to completely scan these 320 servers. To estimate the relative size of the scan data sample, we assume that prefetchable objects are uniformly distributed in any scan data. Since we know the amount of prefetchable bandwidth from the access log data we can then approximate the scan data sample size by the fraction of the prefetchable bandwidth contained in the scan data. According to this approximation our scan data sample size is about 22% of the size of the entire content of the scanned servers.

For the positive evidence we identified and encoded each prefetchable object in the access log by six attributes: (1) age, (2) MIME Type, (3) size, (4) server name, (5) top level domain, (6) the label that marks this entry as prefetchable, and (7) a weight proportional to the size. The first three attributes train object selection, the fourth and fifth attribute train server selection, and the weight represents the relative significance of an entry to the overall bandwidth consumption.

For the negative evidence we collected each day's no-miss-prefetchable objects from the scan data. Recall that no-miss-prefetchable objects are objects that meet all prefetchability conditions as described in section 2.1 except the first condition, *i.e.* the object is not missed during the peak period of the current day. Each no-miss-prefetchable object is encoded and weighted in the same way as prefetchable objects except that the entry is labeled as non-prefetchable.

The resulting training data consists mostly of negative evidence (see Figure 7). The figure also shows that the average size of objects in positive examples is 20,320 Bytes while the average size of in negative examples is 103,183 Bytes.

4.4 Experiments

We conducted two experiments: In the first experiment we applied RIPPER to the training data of the days 5/20/1998 - 5/30/1998. This produced eleven sets of rules. We tested each rule set against the training data of the next day (5/21/1998 - 5/31/1998). This results in prefetch performance data for 11 consecutive days, using a different prefetch strategy each day.

In the second experiment we used boosting to construct a 10 classifier ensemble using RIPPER as a weak learner. Since boosting takes significantly longer than the generation of a single classifier, we only generated one prefetch strategy based on the data of 5/20/1998 and tested its performance on the training data of the 11 remaining days.

To better distinguish between the results of these experiments we refer to the 11 prefetch strategies created by the first experiment as the "non-boosted prefetch strategies", and we call the prefetch strategy from the second experiment the "boosted prefetch strategy".

4.5 Results

Figure 8 shows the performance of machine learned prefetching strategies in terms of bandwidth and in terms of accuracy and coverage. For non-boosted prefetch strategies accuracy is particularly high. This means that the bandwidth used for prefetching is well invested. However coverage is generally low. One reason for this is that the prefetch strategies pick out smaller objects since the average size of positive examples is smaller than the size of negative examples: if the performance is measured based on number of examples instead of number of Bytes, the coverage is significantly higher.

Notice that the first weekend is Memorial Day weekend. HTTP traffic pattern during weekends and holidays are different from traffic patterns during work days. Therefore, Friday provides poor training data for Saturday, and Memorial Monday provides poor training data for Tuesday. This is reflected by relatively low accuracy on Saturday and Tuesday. A better strategy would be to use the previous Sunday as training day for a Saturday, and to use previous Friday as a training day for the first week day, in this case Tuesday.

The performance of the boosted prefetch strategy is particularly high on the first Thursday and Friday because of the proximity to the day of training. In terms of saved bandwidth the boosted prefetch strategy out-performs all non-boosted strategies but shows lower accuracy, particularly during the weekends. Since we only use one strategy which was learned on a week day the boosted strategy performs much better on Tuesday than the corresponding non-boosted strategy which was learned on a holiday.

Figure 9 shows the bandwidth smoothing effect of the learned prefetching strategies relative to the cacheable service of the top 320 servers on Thursday and Friday (for clarity we left out the other days but the results are

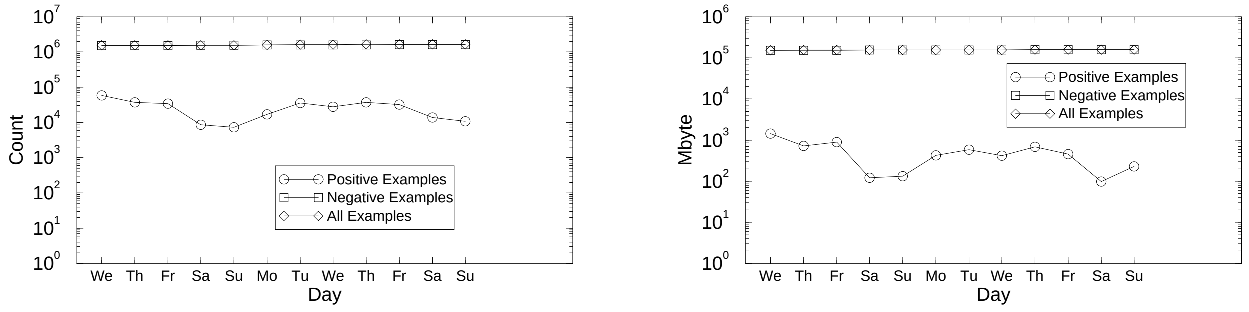


Figure 7: The positive and negative evidence components of each day's training data measured in number of examples and number of Bytes. The y-axis is log-scaled. The vast majority of training examples are negative. Over time the number and size of negative evidence remains almost unchanged. This indicates that most of the no-miss-prefetchable objects are older than the measurement period.

similar). To simulate the effect of prefetching during off-peak hours we raised the off-peak bandwidth usage to a minimum level at which the difference between miss bandwidth and the raised level equals the prefetched bandwidth. The resulting increase in bandwidth usage is represented by white areas below the black curve. During peak periods the bandwidth savings are shown by grey areas above the black curve. No prefetching has been done for the left-most period of the graph (peak period of Wednesday).

The smoothing of peak bandwidth usage varies. At the beginning of the peak period of Thursday, two miss bandwidth peaks are completely cut off. The highest peak on Thursday is reduced by 10.5% using a non-boosted strategy (left graph) and nearly 15% using the boosted strategy (right graph). The peak bandwidth of misses to all servers during Thursday is around 200M Byte/15 mins. Thus the reduction of the overall peak bandwidth usage level is around 3% to 4.5%. While the overall performance does not seem very impressive, the peak load reduction for the selected 320 servers is significant.

4.6 Discussion

The results show (a) and that there is considerable "prefetchable" data available for bandwidth smoothing and (b) that automated machine learning is a promising method to rapidly and automatically develop prefetch strategies (see [18] for an analysis of the machine-learned prefetching strategies).

Our results assume a *loss ratio* of one. The loss ratio is defined as the cost of false positives over the cost of false negatives. A loss ratio > 1.0 means that accuracy is valued higher than coverage, and a loss ratio < 1.0 means that coverage is valued higher than accuracy. The generated prefetch strategies tend to have high accuracy and low coverage. Furthermore, figure 9 shows that our current prefetch strategies do not fully utilize the extra bandwidth available during off-peak peak periods. We are currently investigating results with smaller loss ratios to increase coverage at the expense of accuracy.

The training of these strategies, require information obtained directly from web sites. Web site scanning consumes considerable bandwidth since at least the entire HTML content has to be down-loaded plus header information of images and other objects. Clearly, this time and resource consuming process would make our approach infeasible. However, a well-known query interface that allows clients to issue a query to a server or a search engine for information about objects that match certain properties would significantly reduce the overhead (see for example [2]). There are also a number of promising projects in the Web metadata community [20] which are interested in a compact representation of web content. Finally, we expect that future search engines will provide detailed profiles of web server content.

5 Related Work

We are not aware of any work that investigated real web traffic work loads in terms of shifting peak bandwidth usage to off-peak periods through prefetching; most work on prefetching focuses on immediate prefetching to reduce interaction latency.

In [16] Kroeger *et al.* examine the potential of prefetching for latency reduction. Their study is based on the same traces as our analysis in section 2.3. They found

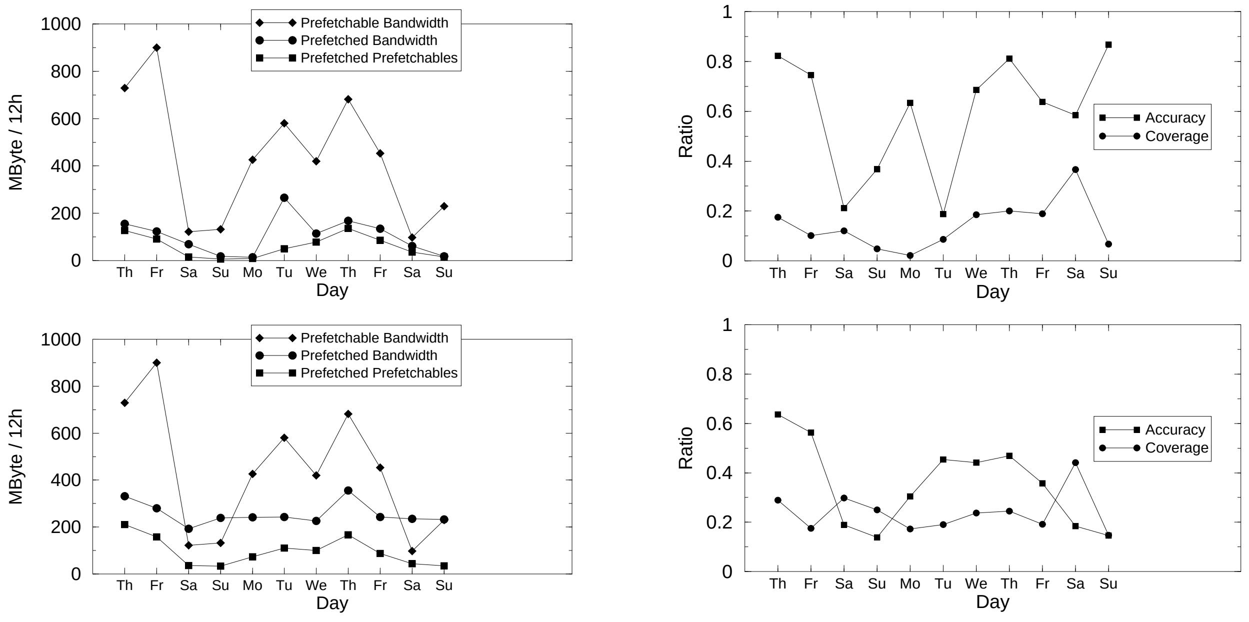


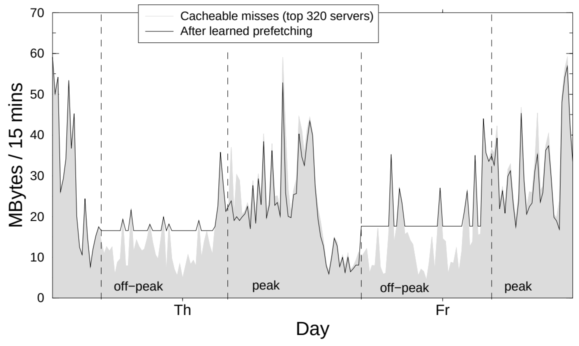
Figure 8: The upper two graphs show the performance of non-boosted prefetch strategies, each one trained on the day previous to the indicated day. The lower two graphs show the performance of the boosted prefetch strategy which was learned on the Wednesday prior to the first Thursday. “Prefetchable bandwidth” represents the total prefetchable bandwidth during the peak period of the indicated day. “Prefetched bandwidth” shows the total bandwidth prefetched during the off-peak period prior to the peak period of the indicated day. “Prefetched prefetchables” shows the total amount of bandwidth saved due to prefetching. The performance on Saturday and Tuesday (Monday was Memorial Day) is weak because of the different traffic pattern of week days and weekend/holidays. In terms of saved bandwidth the boosted prefetch strategy out-performs all non-boosted strategies but shows lower accuracy, particularly during the weekends.

that a combined caching and prefetching approach can at best reduce latency by 60%. Furthermore, the potential latency reduction depends on how far in advance an object can be prefetched. For prefetch lead times below 100 seconds, the latency reduction is significantly lower. In bandwidth smoothing we assume a diurnal bandwidth profile and prefetch lead times of up to twelve hours. The author’s stress that their results are based on a particular environment and assume off-line algorithms with full knowledge of the future. Our results show that machine learning is a promising approach to approximate these algorithms.

In [6] Crovella and Barford show that bandwidth smoothing can lead to an overall reduction of queuing delays in a network and therefore to an improvement of network latency.

A server-initiated prefetching approach based on the structure of web objects and user access heuristics as well as statistical data is presented in [23]. Padmanabhan and Mogul present an evaluation of a server-initiated approach in which the server sends replies to clients together with “hints” indicating which objects are likely to be referenced next [21]. Their trace-driven simulation showed that their technique could reduce significantly latency at the cost of an increase in bandwidth consumption by a similar fraction. Padmanabhan and Mogul’s approach is based on small extensions to the existing HTTP protocol. A similar study but with an idealized protocol was performed by Bestavros [1] in which the author proposes “speculative service” in which a server sends replies to clients together with a number of entire objects. This method achieved up to 50% reduction in perceived network latency.

The greatest challenge in prefetching is to achieve efficient prefetching. In [24], Wang and Crowcroft define prefetching efficiency as the ratio of prefetch hit rate (the probability of a correct prefetch) and the relative increase of bandwidth consumption to achieve that hit rate. Assuming a simplifying queuing model (M/M/1) they show an exponential relationship between the bandwidth utilization of the network link and the required prefetching efficiency to ensure network latency reduction. Crovella and Barford propose “rate controlled



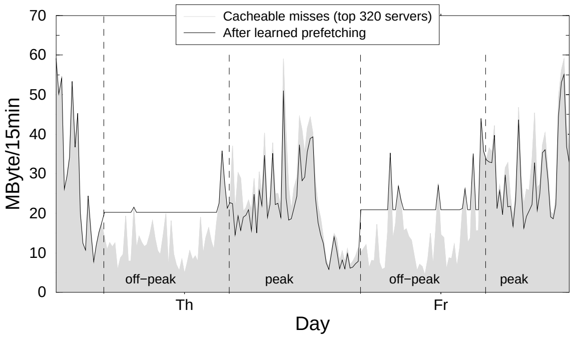


Figure 9: The impact of the learned prefetch strategies on bandwidth consumption (left without boosting, right with boosting). The x-axis marks the beginning of each day. The gray curve shows the bandwidth consumption profile of the top 320 servers. The black curve shows bandwidth consumption after prefetching. To show the impact of prefetching during off-peak hours we raised the off-peak bandwidth usage to a minimum level at which the difference between miss bandwidth and the raised level equals the prefetched bandwidth. White areas below the black curve show the extra bandwidth used for prefetching during off-peak periods. Grey areas above the black curve show the saved bandwidth during peak periods. The very left section of each graph is Wednesday's peak period which was only used for training and not for testing (*i.e.* no prefetching for Wednesday).

prefetching" in which traffic due to prefetching is treated as a lower priority than traffic due to actual client requests [7].

6 Summary and Future Work

We showed that using extra network resources to prefetch web content during off-peak periods can significantly reduce peak bandwidth usage and that these effects are additive to effects of traditional demand-based caching. We presented a mathematical model on how to calculate the benefit of bandwidth-smoothing a particular bandwidth usage profile.

We also showed that machine learning is a promising method to automatically generate prefetch strategies. These strategies were able to prefetch up to 40% of the prefetchable bandwidth and do so without wasting significant bandwidth.

We are in the process of verifying and refining the presented models. We are also experimenting with various machine learning techniques to increase learning performance as well as reduce the duration of the learning process. We are currently working on a technique that increases prefetch performance by finding the optimal balance between accuracy and coverage. Our results above show that even with prefetching there is still a large amount of extra bandwidth available during off-peak periods. Until this extra bandwidth is used we can accept lower accuracy to increase coverage. The above results are based on a machine learning algorithm that assumes equal penalty for false positives and false negatives (the *loss ratio* is 1). By increasing the penalty for false negatives and reducing the penalty for false positives one reduces the accuracy in favor of coverage. We are therefore interested in the relationship between the loss ratio and prefetch performance.

While non-boosted learning of one prefetch strategy on a dedicated workstation takes only a couple of hours, boosted learning took in our case seven days (generating 10 rule sets). One way to speed up the learning process is to increase the abstraction of training data without loosing relevant pattern information. For example, size and age of our training examples are real values. Grouping these values in a well chosen manner can significantly speed up learning.

The current framework assumes *a priori* definition of peak and off-peak periods. Figure 8 and 9 show that the actual bandwidth profile does not always match these fixed periods and that a more flexible notion of peak and off-peak periods would increase the amount of extra bandwidth available for prefetching.

References

[1] Azer Bestavros. Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time for Distributed Information Systems. In *ICDE'96: The 1996 International Conference on Data Engineering*, New Orleans, Louisiana, March 1996.

[2] Krishna Bharat, Andrei Broder, Monika Henzinger, Puneet Kumar, and Suresh Venkatasubramanian. The connectivity server: fast access to linkage information on the web. *Computer Networks and ISDN Systems*, 30(1-7), April 1998. Special Issue on Seventh International World-Wide Web Conference, April 14-18, 1998, Brisbane, Australia.

[3] Brad Calder, Dirk Grunwald, Michael Jones, Donald Lindsay, James Martin, Michael Mozer, and Benjamin Zorn. Evidence-based static branch prediction using machine learning. *ACM Transactions on Programming Languages and Systems*, 19(1):188–223, January 1997.

[4] William W. Cohen. Fast Effective Rule Induction. In *Machine Learning: Proceedings of the Twelfth International Conference (ML95)*, 1995.

[5] William W. Cohen. The RIPPER Rule Learner. Available on the World-Wide Web at http://www.research.att.com/ wcohen/ripperd.html , November 25 1996.

[6] Mark Crovella and Paul Barford. The Network Effects of Prefetching. Technical Report 97-002, Boston University, CS Dept., February 7 1997.

[7] Mark Crovella and Paul Barford. The Network Effects of Prefetching. In *IEEE INFOCOM'98*, San Francisco, CA, 29 March - 2 April 1998. IEEE.

[8] Thomas G. Dietterich. Machine Learning. *ACM Computing Surveys*, 28(4es), December 1996. Available on the World-Wide Web at http://www.acm.org/pubs/citations/journals/surveys/1996-28-4es/a3-dietterich/ .

[9] Thomas G. Dietterich. Machine-Learning Research: Four Current Directions. *AI Magazine*, 18(4):97–136, Winter 1997.

[10] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Mogul. Rate of Change and other Metrics: a Live Study of the World-Wide Web. In *Usenix Symposium on Internet Technologies and Systems (USITS)*, Monterey, CA, USA, December 8-11 1997. Usenix.

[11] Brad Duska, David Marwood, and Michael J. Feeley. The Measured Access Characteristics of World-Wide Web Client Proxy Caches. In *Usenix Symposium on Internet Technologies and Systems (USITS)*, Monterey, CA, USA, December 8-11 1997. Usenix.

[12] Y. Freund and R. E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, Berlin, 1995. Springer Verlag.

[13] Y. Freund and R. E. Schapire. Experiments with a New Boosting Algorithm. In L. Saitta, editor, *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156, San Francisco, CA, 1996. Morgan Kaufmann.

[14] Steven D. Gribble and Eric A. Brewer. System design issues for Internet Middleware Services: Deductions from a large client trace. In *Usenix Symposium on Internet Technologies and Systems (USITS)*, Monterey, CA, USA, December 8-11 1997. Usenix.

[15] Thomas Kroeger, Jeffrey Mogul, and Carlos Maltzahn. Digital's Web Proxy Traces. Available on the World-Wide Web at ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html , October 1996.

[16] Thomas M. Kroeger, Darrel D. E. Long, and Jeffrey C. Mogul. Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. In *Usenix Symposium on Internet Technologies and Systems (USITS)*, Monterey, CA, USA, December 8-11 1997. Usenix.

[17] Carlos Maltzahn, Kathy Richardson, and Dirk Grunwald. Performance Issues of Enterprise Level Proxies. In *SIGMETRICS '97*, pages 13–23, Seattle, WA, June 15-18 1997. ACM.

[18] Carlos Maltzahn, Kathy Richardson, Dirk Grunwald, and James Martin. On Bandwidth Smoothing. In *4th International Web Caching Workshop (WCW'99)*, San Diego, CA, March 30 - April 2 1999.

[19] Jeffrey C. Mogul, Fred Douglis, Anja Feldman, and Balachander Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In *SIGCOMM '97*, Cannes, France, September 1997. ACM.

[20] International Federation of Library Associations and Institutions (IFLA). Digital Libraries: Metadata resources page. Available on the World Wide Web at http://www.nlc-bnc.ca/ifla/II/metadata.htm , August 31 1998.

[21] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using Predictive Prefetching to Improve World-Wide Web Latency. In *SIGCOMM'96*, pages 22–36. ACM, July 1996.

[22] Alex Rousskov and Valery Soloviev. A Performance Study of the Squid Proxy on HTTP/1.0. *to appear in World-Wide Web Journal*, Special Edition on WWW Characterization and Performance Evaluation, 1999.

[23] Stuart Wachsberg, Thomas Kunz, and Johnny Wong. Fast World-Wide Web Browsing Over Low-Bandwidth Links. Available on the World-Wide Web at http://ccnga.uwaterloo.ca/ sbwachsb/paper.html , 1996.

[24] Zheng Wang and Jon Crowcroft. Prefetching in World-Wide Web. Available on the World-Wide Web at http://www.cs.ucl.ac.uk/staff/zwang/papers/prefetch/Overview.html , January 30 1996.