A Statistical, Batch, Proxy-Side Web Prefetching Scheme for Efficient Internet Bandwidth Usage

Sook-Hyang Kim, Jae-Young Kim and James W. Hong
{shk, jay, jwkhong}@postech.ac.kr

Department of Computer Science and Engineering
Pohang University of Science and Technology
Pohang, Korea

Abstract - As the number of World Wide Web (Web) users grows, Web traffic continues to increase at an exponential rate and has become one of the major components of Internet traffic. One of the solutions to reduce Web traffic and speed up Web access is Web caching. However, standard Web caching has limitations in reflecting usage patterns that change on a daily basis, with fluctuating peak and off-peak periods. In this paper, we introduce a prefetching Web caching scheme for reducing bandwidth during peak periods by using bandwidth during off-peak periods. We propose a statistical, batch, and proxy-side prefetching scheme that improves cache hit rate with a small amount of additional storage space. We have simulated our scheme based on event logs of a real Web proxy server and the results indicate that the proposed prefetching scheme can reduce bandwidth during peak-periods. Also, the scheme can be adaptively applied to any Web usage patterns by changing prefetching parameters.

Keywords - Web caching, prefetching scheme, Web access pattern, adaptive bandwidth management

1. Introduction

As the number of World Wide Web (Web) users grows, Web traffic continues to increase at an exponential rate. Currently, Web traffic is one of the major components of Internet traffic. Figure 1 shows Web traffic bandwidth usage in student dormitories for 24 hours at POSTECH (a university in Korea). Figure 1 shows that high bandwidth is required during peak periods, while leaving bandwidth idle during off-peak periods. It is desirable to balance the bandwidth usage between peak periods and off-peak periods. This will reduce Web access time and make more efficient use of Internet links.

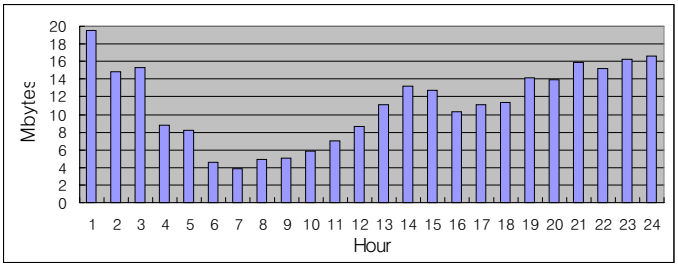

Figure 1. Bandwidth usage pattern of Web requests in a day

One of the solutions to reduce Web traffic and speed up Web access is through the use of Web caching [7, 10]. However, Web caching is limited to reducing network bandwidth usage during peak periods [4, 5, 11, 12]. In this paper, we focus on the use of prefetching, based on a caching server, for reducing bandwidth during peak periods using off-peak period bandwidth. We have developed a statistical, batch, proxy-side prefetching scheme that improves cache hit rate, while only requiring a small amount of additional storage space. This prefetching scheme reflects Web access patterns of users. In fact, this scheme may increase total bandwidth usage slightly in comparison with standard Web caching. However, the proposed scheme can efficiently reduce Web traffic bandwidth usage during peak periods by consuming unused bandwidth during off-peak periods.

The remainder of this paper is organized as follows. Section 2 examines current Web prefetching methods. In Section 3, we discuss the issues associated with the existing prefetching schemes, analyze Web traffic access patterns and present our prefetching scheme. Section 4 describes the design of our prefetching system. In Section 5, we describe the simulation framework used to test our system, and the results of the simulation are presented in Section 6. Finally, Section 7 summarizes our work and discusses directions for future research.

2. Prefetching Methods

Prefetching is a method to anticipate future Web object requests and prefetch the Web objects in a local cache. A prefetching approach can be categorized by following three aspects.

2.1 Server-Side, Client-Side and Proxy-Side Prefetching

Prefetching can be initiated by Web servers, by clients or by cache servers (also called proxies). When a client demands a Web object, a Web server may anticipate the next request and immediately preload the corresponding Web objects to the client [2, 14, 15]. A client can also initiate the prefetching of Web objects by changing the user's configuration. A client can also use a Web access pattern monitoring system, which observes the past access patterns for particular Web objects and prefetches Web objects on the behalf on the user [1, 2, 13]. Also, a proxy can initiate the prefetching of Web objects. A proxy can analyze the Web access patterns of clients, anticipate future requests and prefetch them [3, 17, 18].

2.2 Statistical and Deterministic Prefetching

The decision whether a Web object should be prefetched can be either statistical [5, 8, 9] or deterministic [1]. In statistical prefetching, the prefetching system uses the Web access patterns of users. In a deterministic scheme, prefetching can be configured statically by clients or by proxies. For example, a user may configure Web objects

such as the home page of a popular newspaper with the deterministic prefetching method. These Web objects are always prefetched everytime the prefetching is enabled.

2.3 Prediction and Batch Prefetching

The criteria for deciding when a Web object should be prefetched can either be by prediction or by batch. In the prediction scheme, when a client requests a Web object, a Web server, proxy server or client may predict the next request and immediately preload the predicted Web objects to the client [2, 5, 13, 16, 18]. The goal of most prediction prefetching is to reduce delay in Web access time. However, if the prediction is not accurate, network resources are wasted and network bottlenecks are generated during peak periods. In the batch scheme, Web objects that the user is likely to access in the near future are prefetched during off-peak periods [6].

3. Proposed Prefetching Scheme

In the previous section, we described several prefetching methods. In the server-side prefetching scheme, the client needs to be aware of the prefetching, and both client and Web server must change their configurations. Client-side prefetching cannot share the Web access patterns of all users in an Intranet, because this scheme relies on only one user's Web access pattern. Deterministic prefetching has limited expansion to all Web objects. In the prediction prefetching scheme, network traffic may be generated during peak periods so this scheme is not appropriate for reducing peak bandwidth usage. Therefore, we decided to develop a batch, statistical and proxy-side prefetching scheme.

3.1 Problems

There are several problems associated with a batch, statistical and proxy-side prefetching scheme.

- For the statistical prefetching scheme to work efficiently, Web access patterns of users should be accurately reflected in the prefetching. However, Web access patterns of users can change frequently.
- If a prefetched object is not requested, the network bandwidth for prefetching the object has been wasted. Thus, the hit rate of prefetchable object should be high.
- The traditional batch prefetching scheme demands a large amount of additional disk space. The total amount of disk space for prefetching should be adjustable in a flexible way.

3.2 Solutions

The followings are solutions for the above problems.

- To reflect Web access patterns in prefetching, the prefetching system needs to periodically analyze current Web object access patterns, based on the most recent "access log" of a proxy server, and then generates a list of Web objects that are to be prefetched.
- To increase hit rate of the prefetched objects, we need to select prefetchable objects that might be requested as more as possible. There is a tendency that a Web object that is requested a lot previously, is likely to be requested again in the near future. In order to select Web objects requested previously, a reference count for each Web object is calculated from the access log. Whenever a cached Web object is requested by a client, the reference count for that object is increased. If we salect Web objects that have significant reference count as prefetchable Web objects, the hit rate of the prefetched objects is surely improved.
- To reduce the total amount of disk space required for prefetching, the number of prefetched objects needs to be restricted. Prefetching every Web object will fill up available disk space very quickly. Reference count is a good metric for selecting Web objects. A Web object that has bigger reference count has priority over that of smaller reference count. By applying this method adaptively, the amount of disk space can be adjusted dynamically.

3.3 Prefetching Parameters

The parameters for operating our prefetching scheme are 1) time period for gathering input data (logs of HTTP request), 2) reference count of objects in the cache, 3) total bytes of prefetchable objects, and 4) prefetching start time.

First, the time period for gathering input data (logs) is intended to reflect the Web access patterns of users. The time period is determined by characteristics of network usage patterns in a given user community. Different user community can have different time period parameter (e.g., the day before, several days before, etc.). Web object requests in this time period are considered as input data for caculating a prefetchable object list.

The second parameter is the reference count of objects in the cache. To reduce the unnecessary use of network bandwidth and increase the hit rate of prefetched Web objects, we must set a limit on the minimum reference count of prefetchable objects. In this way, we can obtain a set of highly-referenced Web objects.

The third parameter involves the total bytes of the Web objects that will be prefetched. This can be determined by taking the reference count into consideration. Since the prefetching disk space has a limitation with this parameter, the Web object with a high reference count needs to have a high priority for prefetch selection.

The fourth parameter is the prefetching start time. Since our goal is to use the unused bandwidth during off-peak periods, the prefetching should start and end within the off-peak periods. For example, we decide that the off-peak period is the period with a bandwidth usage lower than 80% of average bandwidth usage in a day. This is an engineering decision and can be defined differently in different situations. Consequently, from Figure 1, the off-peak period is from 04:00 to 13:00 (9 hours) and the peak period is from 13:00 to 04:00 (15 hours).

3.4 A Real Web Traffic Trace

We have monitored Web traffic from sixteen IP C class subnets that are used by student dormitories at POSTECH for two weeks, from October 15, 1999 to October 28, 1999. The total number of requests from 447 different clients was 4.07 millions and the total amount of Web objects sent

back to clients was 50.2 Gbytes. The average object hit rate was 54.96% and the average byte hit rate was 31.42%. Daily average number of requests was 291,236 and daily average amount of Web objects was 3.6 Gbytes.

When considering the reference count of each Web object, the numbers are changed as in Table 1 because many Web objects were accessed repeatedly. 291,236 HTTP requests were made for only 121,883 uniquely identifiable Web objects. Also the disk space for storing the Web objects was about 2.1 Gbytes.

Table 1. An example Web traffic categorized by reference count

Reference count	Daily average # of objects (%)	Daily average amount of total objects (Mbytes) (%)
1	86,227 (70.75%)	1411.2 (68.12%)
2	16,690 (13.69%)	296.39 (14.31%)
3	6,462 (5.3%)	112.71 (5.44%)
4	3,451 (2.83%)	70.74 (3.41%)
over 5	9,053 (7.43%)	180.74 (8.72%)
Total	121,883 (100%)	2071.62 (100%)

Among the cached Web objects, 70.75% of them were never requested afterwards, which had a reference count of 1. Only 29.25% of them had a reference count of 2 or more. The total amount of Web objects that were accessed only once in a day was 68.12% of the total amount. Only 31.88% of the disk space will be needed when we store Web objects with a reference count of more than 1 only.

4. Design of a Prefetching System

Based on the prefetching scheme discussed in the previous section, we have designed a prefetching system. The design architecture of our system cooperating with a caching server is illustrated in Figure 2.

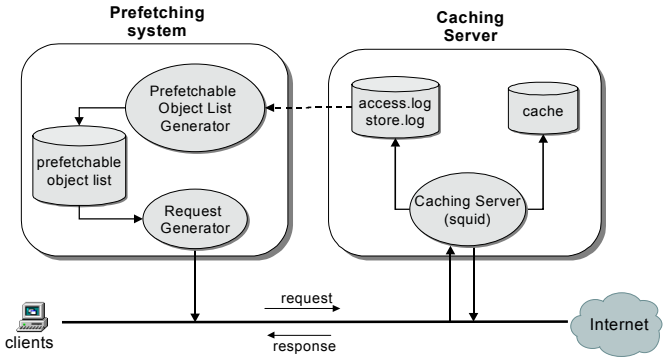

Figure 2. Design architecture of our prefetching system

The overall system consists of two parts: a prefetch list generator and a request generator. The prefetching system works together with a public domain caching server (Squid [7]). Squid can be configured to create log files recording Web caching and access events. Our prefetching system imports two log files, named “access.log” and “store.log” from Squid cache server. And then the prefetch list generator produces a list of prefetchable object and the request generator actually sends HTTP requests for Web objects in the list.

4.1 Prefetch List Generator

The prefetch list generator is used to decide whether a Web object should be prefetched. It uses the access log that reports the access information of HTTP requests from clients, and the store log that reports the information of Web objects in the cache. Figure 3 and Figure 4 show the entry format of access log and store log respectively.

Time	Elapsed	Remotehost	Code/Status	Byte	Method	URL

Figure 3. Access log entry format

Time	Action	Status	OBJ_DATE	OBJ_LASTMOD	Expires	Type	Len	Method	URL

Figure 4. Store log entry format

The access log keeps records of Web cache requests. For each requested Web object, time of request, size of the object, and URL of the object are extracted to understand access patterns of Web objects in the cache.

The store log keeps records of status of cached Web objects. OBJ_DATE is a value of the HTTP date [19] denoted by the date and time at which the message originated. OBJ_LASTMOD is a value of the HTTP Last-Modified, indicating the date and time when the sender believes the resource was last modified. Expires from the HTTP reply header gives the date and time after which the entity should be considered stale.

By analyzing the above values in the log files, the prefetch list generator makes a list of prefetchable objects with calculated reference counts. Figure 5 shows the format of a prefetch list entry.

Reference Count	URL	Byte

Figure 5. Prefetch list format

4.2 Cache Refreshing Algorithm of Squid

Web objects need to be removed from the cache when they expire. When a cached Web object is requested, Squid checks the freshness of the object, and then records the event in access log and store log. If an object is ‘FRESH,’ it is available to clients. If an object is ‘STALE,’ it needs to be fetched from the original server.

Figure 6 shows how Squid decides when to refresh a cached object [7]. The refresh algorithm is checked in the order listed here and the first matching entry is used. If none of the entries matches, then the default will be used.

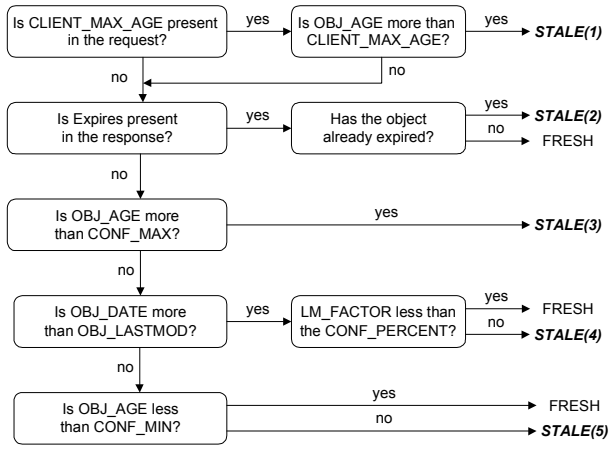

Figure 6. Flow chart of cache refreshing algorithm of Squid

The CLIENT_MAX_AGE is the maximum object age the client will accept as taken from the HTTP/1.1 Cache-Control request header [20]. The OBJ_AGE is how much

the object has aged since it was retrieved. The LM_AGE is how old the object was when it was retrieved. The LM_FACTOR is the ratio of OBJ_AGE to LM_AGE. The NOW is current time. AGE is how much the object has aged since it was retrieved: OBJ_AGE = NOW - OBJ_DATE. LM_AGE is how old the object was when it was retrieved: LM_AGE = OBJ_DATE - OBJ_LASTMOD. LM_FACTOR is the ratio of AGE to LM_AGE: LM_FACTOR = OBJ_AGE / LM_AGE. MAX_AGE, MIN_AGE and CONF_PERCENT are compared with the parameters of the refresh pattern rules. These values are used for the default value of Expires.

4.3 Freshness Decision for Our Prefetching System

Our prefetching system also needs to have a similar way of deciding freshness of prefetchable Web objects already stored in the cache server. After a prefetch list is made, every prefetchable object in the list should be tested whether it is stale or not, before it is actually requested. It is unnecessary to request Web objects already in fresh state. We need to distinguish only stale objects from the prefetch list.

One very important thing to consider here is that prefetched objects are not used at the time of fetching, but are expected to be used at sometime in the future. Therefore, when checking the freshness of a Web object, we have to make decisions as if we were at some time later from the current time. The time interval into the future can be adjustable for different usage patterns. We have chosen 24 hours (1 day) for this time interval. Thus, at the time of prefetching we are to prefetch two kinds of Web objects: Web objects that are already stale at that time and Web objects that will be stale 24 hours later.

Figure 7 illustrates the operations executed between each prefetching time interval. The current date and time is 4:00 am, October 16. During the previous 24 hours the prefetching system gathers input log data for creating the prefetch list. At the current prefetching time every object in the prefetch list is checked for the freshness decision as if a user requests the object at 4:00 am, October 17.

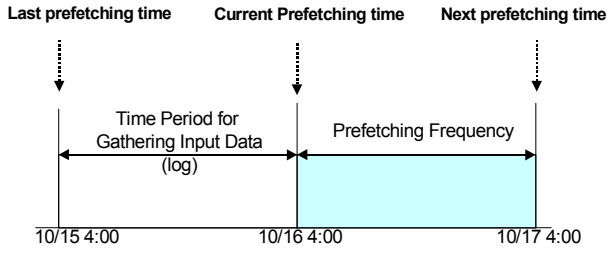

Figure 7. Prefetching frequency

We have used a modified version of the cache refreshing algorithm of Squid for the freshness decision in our prefetching scheme. The modification is quite simple. The parameter "NOW" in Section 4.2 should be extended to reflect the time interval into the future. The new value for NOW is calculated by adding prefetching time interval to the current time. In this way our prefetching system is able to make requests for Web objects that will expire in the future.

4.4 Request Generator

The request generator operates as a Web client. The request generator will generate HTTP requests from the prefetch list and send the HTTP requests to the Web server during off-peak periods. We have developed the request generator using "wget," which is a command-line Web client that supports HTTP [21]. Implementation of the request generator is straightforward.

5. Simulation

We have designed a simulation model to estimate the performance of our prefetching scheme. The simulation is trace-driven and based on log files of a real cache server (Squid), such as access log and store log, are employed to emulate the requests of the clients and to access the operational behavior of the prefetching system. We have also verified our scheme with the simulation system illustrated in Figure 8.

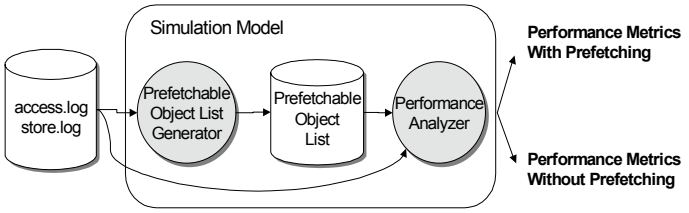

Figure 8. Simulation system

The simulation procedures are as follows:

- The prefetch list generator extracts the prefetch list from the access log using the prefeching parameters determined in Section 5.1.
- If a Web object in the prefetch list exists in the access log of the next day, the performance of prefetching becomes higher.
- The performance of the cache server with prefetching is compared to the performance of the cache server without prefetching. The factors of comparison are performance metrics of proposed prefetching in Section 5.2.

5.1 Prefetchable Web Objects

Using the prefetching parameters described in Section 3 and the results from the Web traffic trace, we have selected the conditions of prefetchable objects for simulation.

The period for gathering input data (log files) is 24 hours. Analyzing the results of the web traffic traces at POSTECH, we found that the access pattern variation is repeated every day.

To evaluate the performance of our prefetching approach per reference count, we chose the prefetchable Web objects that are referenced more than once, more than twice, more than three times, more than four times, and more than five times. In this section, we did not select particular reference count for prefetching because we should decide appropriate reference count for prefetching from the results of simulation.

The total size of prefetchable objects is less than the surplus bandwidth of off-peak periods and the prefetching start time is from 04:00 to 13:00 (9 hours).

5.2 Performance Metrics

Four quantities, expressed in percentages, are used to evaluate performance of the prefetching system:

- Request Saving: The number of prefetched Web objects that the users requested for the first time divided by the total number of requested pages. When the request saving increases, the cache object hit rate also increases.
- Bandwidth Saving: The amount of bytes of prefetched Web objects that the users requested for the first time divided by the total amount of bytes of requested Web objects. The bandwidth saving prompts the cache bytes hit rate to increase.
- Accuracy: It consists of the prefetched object hit rate and the prefetched byte hit rate.
  - ✓ prefetched object hit rate: The number of prefetched Web objects that the users requested divided by the total number of prefetched Web objects.
  - ✓ prefetched byte hit rate: The amount of bytes of prefetched Web objects that the users requested divided by the total amount of bytes of prefetched Web objects.
- Wasted Bandwidth: The amount of bytes of Web objects that are prefetched but are not accessed by the client.

6. Results

In this section, we discuss the results obtained from our simulation experiments based on performance metrics.

6.1 Request Saving

Figure 9 shows the performance gain of our prefetching scheme with respect to the cache object hit rate. The cache object hit rate is increased by maximum 4.3% compared with the non-prefetching case. When the reference count increases, the request saving decreases.

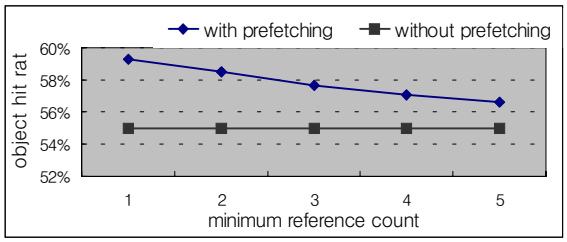

Figure 9. Object hit rate according to minimum reference count

6.2 Bandwidth Saving

Figure 10 reports the cache byte hit rate with and without the use of prefetching. The cache byte hit rate is raised by maximum 5% higher on average than the non-prefetching system When the reference count increases, the bandwidth saving decreases.

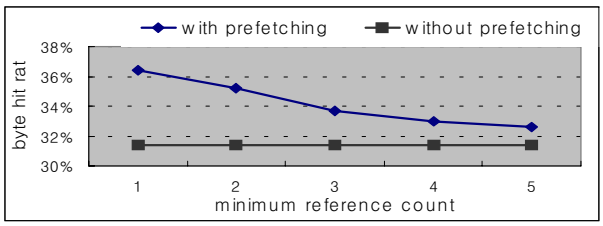

Figure 10. Byte hit rate according to minumum reference count

6.3 Accuracy

Figure 11 illustrate the accuracy of prefetching. In Table 4, when the reference count increases, the prefetched object hit rate and prefetched byte hit rate also increase.

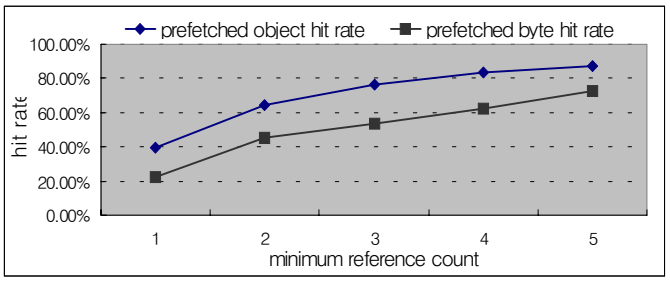

Figure 11. Accuracy per reference count

6.4 Wasted Bandwidth

Figure 12 illustrates a rate of increase about bandwidth usage during off-peak periods. When the reference count increases, wasted bandwidth decreases.

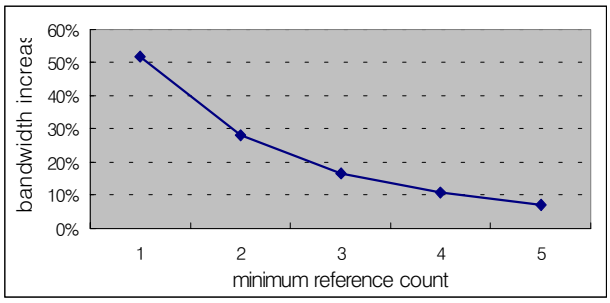

Figure 12. Banwidth increase during off-peak periods per minimum reference count

6.5 Summary of Results

Table 2 summarizes performance metrics of prefetching per reference count. If the reference count is higher, then the request saving, the bandwidth saving, and the overall accuracy improves and the wasted bandwidth decreases.

Table 2. Summary of performance metrics per reference count

Minimum reference count	Request saving	Bandwidth saving	Wasted bandwidth	Accuracy	
				Prefetched object hit rate	Prefetched byte hit rate
1	4.30 %	5.01 %	17.8 %	39.4 %	22 %
2	3.53 %	3.78 %	4.6 %	64.3 %	45.1%
3	2.72 %	2.27 %	2.0 %	76.5 %	53.5 %
4	2.10 %	1.59 %	1.0 %	83.3 %	62.6 %
5	1.62 %	1.19 %	0.5 %	87.1 %	72.6 %

When the amount of reduced bandwidth usage by prefetching increases in volume and the amount of wasted bandwidth by prefetching decreases, the effectiveness of prefetching increases. Accordingly, we define the effectiveness of prefetching in the following equation:

$E_p = B_s / B_w$

Where E_p: Effectiveness of prefetching
B_s: Amount of saved bytes (Mbyte)
B_w: Amount of wasted bytes (Mbyte)

Table 3 shows the effectiveness of prefetching and the total rate of decrease of bandwidth per reference count by prefetching.

Table 3. Effectiveness of prefetching per reference count

Minimum reference count	B_s (Mbyte)	B_w (Mbyte)	E_p	Bandwidth Saving
1	189.16	670.72	0.28	5.01 %
2	141.82	172.65	0.82	3.78 %
3	85.65	74.47	1.15	2.27 %
4	60.08	35.91	1.67	1.59 %
5	45.06	16.97	2.66	1.19 %

In the previous section, we do not select a particular reference count for prefetching because the appropriate reference count for prefetching must be determinated by the results of simulation. To select an appropriate minimum reference count for prefetching, we should consider the effectiveness of prefetching and the amount of bandwidth saving per the minimum reference count. When the effectiveness of prefetching increases, the amount of bandwidth saving decreases. Therefore, we should choose a minimum reference count to guarantee E_p more than 0.5 with a bandwidth saving of greater than 3%. Consequently, the most appropriate minimum reference count is 2.

When the minimum reference count is 2, the average number of bytes of prefetched objects is 201,649,369 bytes. Since the peak period is 64,800 seconds (19 hours), the amount of bandwidth saving can be calculated as

Bandwidth saving

= (141824895 bytes * 8) / 54000 sec = 20.52 Kbps

Thus, the amount of bandwidth saved by prefetching when the minumu reference count of 2 is 20.52 Kbps.

7. Summary and Future Work

We have presented a prefetching scheme for Web traffic with the goal of reducing peak bandwidth usage. Our results indicate that statistical, batch, and proxy-side prefetching can lead to a significant reduction of peak bandwidth usage by using extra network resources to prefetch Web objects during off-peak periods. The amount of bandwidth saving by the proposed prefetching scheme applied to POSTECH was 20.52 Kbps.

We can increase the cache object hit rate and byte hit rate using our scheme. Also, our prefetching scheme results in high accuracy with the increased hit rate of prefetched objects, ranging from 22% to 73%. Consequently, with our prefetching scheme, a cache server can use network bandwidth effectively.

The work presented in this paper investigated the use of the Web in an academic environment, which does not have a strong usage pattern. We believe that we could achieve similar or better results in an industrial environment where the off-peak periods are typically larger (e.g., from 20:00 to 08:00) and the usage pattern is much more apparent.

For future work, the prefetching system needs to have a feature of dynamic adaptation of prefetching parameters for handling frequent change in usage patterns more effectively. We also consider applying the prefetching scheme to streaming Web objects that are consuming a lot of bandwidth currently. And easy-to-use management and monitoring interfaces for controlling our prefetching system are under development.

References

[1] Z. Wang and J. Crowcroft, “Prefetching in World Wide Web,” IEEE Globecom’96, http://www.cs.url.ac.uk/staff/zwang/papers/.

[2] V. Padmanabhan and J. Mogul, “Using Predictive Prefetching to Improve World Wide Web Latency,” Computer Communication Review, 26(3):22-36, July 1996.

[3] Ken-ichi Chinen and Suguru Yanaguchi, “An Interactive Prefetching Proxy Server for Improvement of WWW Latency,” INET’97, 1997, http://www.isco.org/INET97/procceding/A1/A1_3.HTM.

[4] Arthur Goldberg, Ilya Pevzner and Robert Buff, “Caching Characteristic of Internet and Intranet Web proxy Traces,” In Computer Measurement Group Conference (CMG’98), Anaheim, CA, December 1998, http://www.cs.nyu.edu/artg.

[5] Carlos Maltzahn and Kathy J. Richardson, “On Bandwidth Smoothing,” In Web Caching Workshop WCW’99, 1999, http://www.ircache.net/Cache/ Workshop99/program.html.

[6] Themistoklis Palpanas and Alberto Mendelzon, “ Web Prefetching Using Partial Match Prediction,” In Web Caching Workshop WCW’99, 1999, http://www.ircache.net/Cache/Workshop99/program.html.

[7] Squid Internet Object Cache, http://squid.nlanr.net/ Squid/.

[8] Gihan V.Dias, Graham Cope and Ravi Wijayaratne, “A Smart Internet Caching System,” INET’96 Conference, 1996, http://www.isoc.org/ isoc/whatis/conferences/inet/96/proceedings /a4/a4_3.htm.

[9] Katsuo Doi, “WWW Access by Proactively Controlled Caching Proxy, ” Sharp Technical Journal, No. 66, December 1996.

[10] 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, http://www.cs.ubc.ca/spider/marwood/Projects/SPA/Report/Report.html.

[11] Marc Abrams, C.R.Standridge, G.Abdulla, S.Williams, and E.A.Fox, “Caching Proxies: Limitations and Potentials,” In Proceedings of the Fourth International WWW Conference, 1995, http://ei.cs.vt.edu/ ~succeed/WWW4/ WWW4.html.

[12] Anawat Chankhunthod et al, “A Hierarchical Internet Object Cache, ” Technical Conference, Usenix 1996, http://excalibur.usc.edu/ cache-html/cache.html.

[13] James Griffioen and Randy Appleton. “Reducing File System Latency using a Predictive Approach,” Proceddings of the 1994 Summer USENIX Technical Conference, Boston, Massachusetts, USA, 1994, http://usenix.org/publications/library/proceedings/ bos94/griffioen.html.

[14] Azer Bestavros, “Speculative Data Dissemination and Service to Reduce Server Load,” Network Traffic and Service Tome in Distributed Information System, In International Conference on Data Engineering, pages 180-189, New Orleans, LO, February 1996.

[15] Tomas M. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. “Exploring the bounds of web latency reduction from caching and prefetching,” In Proceedings of USENIX Symposium on Internet Technology and Systems, December 1997, http://www.usenix.org/ publications/library/proceedings/ usits97/kroeger.html.

[16] Evangelos P. Margatos and Catherine E. Chronaki, “A top-10 Approach to Prefetching on the Web,” Technical report, In Proceedings of INET' 98 (The Internet Summit), Geneva, Switzerland, July 1998, http://www.ics.forth.gr/ proj/arch-vlsi/OS/www.html.

[17] Wcol Group, “WWW Collector – the prefetching proxy server for WWW,” 1997, http://shika.aist-nara.ac.jp/products/wcol/wcol.html.

[18] Li Fan, Quinn Jacobson, Pei Cao, and Wei Lin, ”Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance”, In Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '99), Atlanta, GA, May 1999, http://www.cs.wisc.edu/~cao/.

[19] T. Berners-Lee, R. Fielding, and H. Frystyk, “Hypertext Transfer Protocol – HTTP/1.0,” RFC 1945, May, 1996.

[20] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee, “Hypertext Transfer Protocol - HTTP/1.1,” RFC 2616, June 1999.

[21] GNU wget Homepage, http://www.gnu.org/software/wget/wget.html.