Performance Impact of Large File Transfer on Web Proxy Caching: A Case Study in a High Bandwidth Campus Network Environment

Hyun-Chul Kim, Dongman Lee, Kilnam Chon, Beakcheol Jang, Taekyoung Kwon, and Yanghee Choi

Abstract: **Since large objects consume substantial resources, web proxy caching incurs a fundamental trade-off between performance (i.e., hit-ratio and latency) and overhead (i.e., resource usage), in terms of caching and relaying large objects to users. This paper investigates how and to what extent the current dedicated-server based web proxy caching scheme is affected by large file transfers in a high bandwidth campus network environment. We use a series of trace-based performance analyses and profiling of various resource components in our experimental *squid* proxy cache server. Large file transfers often overwhelm our cache server. This causes a bottleneck in a web network, by saturating the network bandwidth of the cache server. Due to the requests for large objects, response times required for delivery of concurrently requested small objects increase, by a factor as high as a few million, in the worst cases. We argue that this cache bandwidth bottleneck problem is due to the fundamental limitations of the current centralized web proxy caching model that scales poorly when there are a limited amount of dedicated resources. This is a serious threat to the viability of the current web proxy caching model, particularly in a high bandwidth access network, since it leads to sporadic disconnections of the downstream access network from the global web network. We propose a peer-to-peer cooperative web caching scheme to address the cache bandwidth bottleneck problem. We show that it performs the task of caching and delivery of large objects in an efficient and cost-effective manner, without generating significant overheads for participating peers.**

Index Terms: **Peer-to-peer, performance measurement, web cache, workload characterization.**

I. INTRODUCTION

Web proxy caching is one of the most popular techniques to facilitate information sharing on the Internet. As with other forms of caching used at various levels of the memory hierarchy (e.g., hardware, operating systems, application), web proxy caching exploits the reference locality principle to improve the cost and performance of data access [1]. This approach has been especially effective for the Internet, where large geographic and topological distances separate the content producers and consumers.

From the outset, one of the challenges for web proxy caching has been efficient handling of variable-sized web objects [2] with only a limited amount of dedicated resource. The size of web files spans several orders of magnitude and its distribution is heavy-tailed. Explicit support for multimedia formats on the web has led to a significant increase in web file size, thereby increasing the tail weight of the size distribution [2]–[4]. The development of compression techniques, such as MPEGs, and gigantic archives of scientific data sets has made the challenge more difficult and urgent. As the network provides higher bandwidth than before[1] and content providers redesign their sites to support users with high-speed access, users are more likely to regularly download all kinds of very large objects [2], [5].

This widespread increase in the transfer of large objects has a significant impact on the web, as well as its underlying network infrastructure [2]. For instance, intense bursts of web traffic are more frequently caused by concurrent requests for large objects, rather than a large number of requests for small objects [6], placing even more load on web proxy caches. However, to the best of our knowledge, there has been no work that measures and quantifies the impact of large objects on the performance of web proxy caching, though the proliferation of large object downloads may make web proxy caching a bottleneck in a high bandwidth network.

This paper presents a case study questioning the viability of the current (centralized) web proxy caching model in a high bandwidth network environment, in which end users often download large web objects. We conduct a systematic study that quantifies the impact of large object transfer on the performance of our experimental *squid* [7] cache server.

Our contributions are two-fold: First, this is the first study that performs a thorough trace-based analysis on the impact of large object transfer on web caching performance in a high bandwidth network environment, in terms of a cache's bandwidth usage and response times. We use real-world web cache traces collected at two high bandwidth campus networks, in two countries with the highest broadband and fiber-to-the-home (FTTH) penetration in the world; Japan and Korea. Even a small number of concurrent requests for large objects often saturates our web proxy cache's

Manuscript received February 06, 2008; approved for publication by Hyunseung Choo, Division III Editor, August 05, 2009.

This work was financially supported by NAP of Korea Research Council of Fundamental Science and Technology and the grant from the IT R&D program of MKE/IITA [2007-F-038-03, Fundamental Technologies for the Future Internet].

H.-C. Kim, T. Kwon, and Y. Choi are with the School of Computer Science and Engineering, Seoul National University, Seoul, South Korea, email: {hkim, tk, yhchoi}@mmlab.snu.ac.kr.

D. Lee is with the Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, South Korea, email: dlee@cs.kaist.ac.kr.

K. Chon is with the Graduate School of Media and Governance, Keio University, Japan, email: chonkn@gmail.com.

B. Jang is with the Department of Computer Science, North Carolina State University, Raleigh, USA, email: bcjang@gmail.com.

[1]With the rapid growth of IP networking technologies, it is common for today's local/regional area networks, as well as domestic/international research networks, to have more than 100 Mb/s of user connectivity and 1 Gb/s-class backbone [8].

bandwidth resource and causes significant performance degradation in a high bandwidth network environment. Due to the delivery of large objects, comprising less than 0.1% of the total number of requests, approximately 30% of concurrent requests for small objects are delayed. During peak-times, the (worst case) response time required for delivery of small objects increases by a factor as high as a few million. We argue that the bandwidth bottleneck problem is due to the fundamental limitations of the current centralized web proxy caching model that scales poorly when there is a limited amount of dedicated resource. This is a serious threat to the viability of the current web proxy caching model, particularly in a high bandwidth access network, since it actually leads to sporadic disconnection of the downstream access network from the global web network.

Second, we address the bottleneck problem by proposing a peer-to-peer cooperative web caching scheme. We show that the proposed scheme performs the task of caching and delivery of large objects in an efficient and cost-effective manner, without generating significant overheads for participating peers.

The remainder of this paper is structured as follows: Section II explains our cache traces and performance evaluation methodologies. In Section III, we empirically show the extent to which large object transfers affect the performance of cache servers in handling requests for the other concurrent small objects. To address the cache bandwidth bottleneck problem, Section IV proposes a peer-to-peer cooperative web caching scheme and evaluates its performance. Section V discusses related work. We conclude this paper in Section VI.

II. EXPERIMENTAL METHODOLOGY

In this section, we discuss the details of the collected trace workloads, describe our experimental environment and detail the performance metrics used in the experimental evaluation.

A. Access Characteristics of Large Objects

We employed traces from *leaf* university caches, at KAIST campus network in South Korea and Waseda University's Nishi-waseda campus and Toyama campus network in Japan to generate the workload.[2] End users in KAIST and Waseda are provided with 10/100 Mb/s or 1 Gb/s switched Ethernet connectivity. Table 1 summarizes the trace information. All traces were collected over four days. Each consisted of tens of million requests, generated by a population of several thousand clients. It also shows the proportion of the number of requests for large objects, in terms of access counts and transferred bytes, per trace.[3] In both traces, the traffic volume by objects larger than 1 MB comprised about 30–50% of the total traffic, though the number of these requests was less than 1% in terms of the access count. The results are consistent with Jung *et al.* [4], except that the proportion of traffic generated by objects larger than 10 MB in the KAIST trace has increased by more than a factor of three over the past three years (from 10% to 30%). This is likely due to the recent proliferation of high quality video data and large compressed files on the Internet. It appears that the size of large objects has been increasing on the web, especially where broadband access networks are widely deployed. Though the Waseda trace contains a relatively smaller proportion of requests for large objects, the results are also consistent with Jung *et al.* [4].

Here, we show the potential count hit-ratio and byte hit-ratio due to caching objects larger than 1 MB per trace. We considered an object to be cacheable if it did not contain substrings such as 'cgi-bin' or '?' and if it did not have a file extension such as '.cgi,' as in [4] and *squid* [7]. All uncacheable objects were counted as cache misses, as with [4]. According to our analysis, some large multimedia objects are extremely popular. A few video files were accessed thousands of times, and tens of multimedia objects were accessed hundreds of times during the trace collection periods. For each KAIST and Waseda trace, potentially 42.1% and 24.8% of the byte hit-ratio and 53.6% and 25.5% of the count hit-ratio, respectively, could have been due to caching of objects larger than 1 MB. This indicates (i) the locality of large multimedia objects is significantly high in both traces and (ii) even a single cache-miss for an extremely large object may require a large volume of traffic to be fetched from its distant original web server [9]. Therefore, large objects should be cached, to reduce network bandwidth usage significantly. In Section IV, we investigate how and to what extent caching and delivery of large objects directly affects the performance of web caching.

Table 1. Characteristics of trace workload.

Trace	KAIST	Waseda
Duration	2002.6.7–10 (96 hr.)	2003.12.9–12 (96 hr.)
# of requests	39.4 M	24.7 M
Transferred bytes	375 GB	193 GB
# of unique clients	8,798	3,147
% of requests > 1 MB (count/byte ratio)	0.06/47.3	0.04/28.2
% of requests > 10 MB (count/byte ratio)	0.006/32.8	0.004/13.9
% of requests > 100 MB (count/byte ratio)	0.0009/17.7	0.0001/2.3
% of requests > 1 GB (count/byte ratio)	None	None
Size of the largest object	681 MB	652 MB

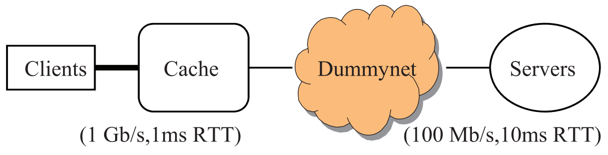


Fig. 1. Experimental environment.

B. Experimental Environment

Fig. 1 illustrates the experimental environment. We used popular open source codes, *proxycizer* [10], *squid* [7], and *dummynet* [11]. One of the most popular open-source cache server software, *squid* is known to comprise at least 80% of the caching

[2]These traces are the most recent *leaf* (university) cache traces available to us, since the cache servers had been dismantled in mid-2000s. There were no publicly available *leaf* cache traces collected in 2004–2009 either. We admit that our traces, collected in 2002 and 2003, may look old-fashioned, however, even if we used more recent traces, it would not affect what we find in this paper; even a small number of large file transfers often saturates our cache servers' bandwidth resource and significantly degrade its performance, since the number of requests for large objects and transfers are obviously increasing as time goes by [2], [5].

[3]Since the size of multimedia objects such as video, audio, and high-definition image data typically ranges from a few Megabytes to a few Gigabytes, this paper assumed that objects larger than 1 MB are large.

proxy market in Europe, and about 70% in the US [12]. The *proxycizer* package, a suite of simulation tools that can be used in simulating and/or driving web caches developed at Duke university, was used to perform trace-driven workload generation. Among various tools included in the package, we used *simclient* to stream requests to the *squid* web cache. It generates a real-time request stream given in a real-world trace. For the server side, *webulator*, a non-blocking *http* server that returns objects on request was used. It returns objects based on the information from a database generated from a real-world trace (i.e., using the length field to determine the length of a requested object). *Squid* was used as a web caching component. We used *dummynet* to emulate the communication delay and bandwidth limitations. In *dummynet*, the features of an emulated network are generally controlled and configured via a command line interface.

We conducted experiments on three PCs. One was an Intel PC server implementing the *squid* cache server, with 2 GHz Pentium 4 CPU, 2 GB RAM, and 1 Gb/s Ethernet NIC. The other two implemented *simclient* and the emulated web server, *webulator*, respectively. They had 2 GHz Pentium 4 CPU, 1 GB RAM, and 1 Gb/s Ethernet NIC. *Squid*'s cache space was configured to 100 GB. The cache server's bandwidth was set to 1 b/s, as in the KAIST and Waseda web caches.[4]

We set the client-cache round trip time to 1 millisecond, based on the round trip time measurement results obtained at the KAIST campus network. We set the upstream bandwidth of the cache server to 100 Mb/s, based on the reports generated from the FlowScan+ system that monitors the flow information at the KAIST campus network border router. We set the cache-server round trip time to 10 milliseconds, since 70–80% of HTTP requests for large objects logged at the KAIST trace workloads were directed to domestic web servers located in South Korea. The Waseda trace showed a similar pattern; around 60–70% of requests were directed to domestic web servers for which the country code top level domain (ccTLD) is *jp*.

We ignored packet losses and delay variability. These assumptions are acceptable (previous studies on the performance of web caches have also made similar assumptions [5], [13]), since we were interested in a web cache server's characteristics in a high bandwidth access network where large objects are frequently requested, rather than the effects of various Internet dynamics on these workloads.

C. Data Reduction and Performance Metrics

We created a smaller, more compact log due to the extremely large access logs created by caches (nearly 10 GB of data in total). This enabled us to complete our workload re-generation and analyses in real-time. To this end, we used one day's workload per trace, instead of all trace workloads. Two criteria were used for the selection of the trace: First, we selected a workload in which both the number of requests for large objects and the volume of traffic generated by these requests was greatest among the collected traces. This enabled us to determine to what extent requests for large objects can generate overhead for the cache server. Second, we selected a workload for which the total number of requests was sufficiently small to ensure that loads generated by small objects delivery were as low as possible. By this means, we aim to show the impact of large object delivery on the performance of web caching more clearly.

[4]Japan, Korea, Hong Kong, and much of Europe already lead the United States in FTTH deployments, with a number of other nations now accelerating their own FTTH efforts. A 100 Mb/s connection capable of supporting convergence of voice, video, and very high speed bi-directional data is now common in Japan, Korea, and Hong Kong. It will soon be available in a number of other countries [14].

Table 2. Characteristics of the selected trace workload.

Trace	KAIST	Waseda
Duration	2002.6.9 (24 hr.)	2003.12.11 (24 hr.)
# of requests	7.38 M	5.61 M
Transferred bytes	79 GB	42 GB
# of unique clients	5,633	2,186
% of requests > 1 MB (count/byte ratio)	0.08/55.5	0.05/32.3
% of requests > 10 MB (count/byte ratio)	0.008/41.1	0.004/20.4
% of requests > 100 MB (count/byte ratio)	0.001/24.6	0.0009/6.2
% of requests > 1 GB (count/byte ratio)	None	None
Size of the large object	681 MB	231 MB

Table 2 shows the characteristics of the selected traces. In both the KAIST and Waseda traces, the proportion of the number of requests and the volume of traffic for large objects is larger than those of the four day average, shown in Table 1.

There are no object replacement or removal operations during the experiments, since the total volume of transferred bytes is less than the cache storage space (100 GB) in both traces. This is acceptable for our experiments, since we focus on the direct impact of large object transfers—particularly through our experimental web cache—on the delivery performance of other small objects concurrently transferred to users, rather than the impact of replacement or removal operations related to large objects that have been cached in the cache server, on the performance of web caching.

We made three different versions of the selected traces and compared the results obtained for each to measure and quantify the impact of large object delivery on the performance of our *squid* web proxy cache. The first is the real-world trace that contains all requests for large objects (> 1 MB). The second contains no such requests. The third is an intermediate one, where requests for objects larger than 10 MB are eliminated from the original trace. We used two different metrics, cache bandwidth usage and response times required for delivery of small objects for the comparison. The former is defined as the volume of data that passes through a web cache per unit time (one second). The latter is defined to be the time required for the cache server to service the request. From the experimental results, we intended to answer the following questions. This allows us to determine the load on our experimental web cache server due to concurrent requests for large objects:

1. To what extent do the requests for large objects contribute to generating peak web traffic?
2. Does the traffic generated by large objects affect and delay concurrently requested small objects? If so, to what extent?
3. What is the threshold size of large objects that generate significant overheads for the experimental 1 Gb/s web cache?
4. Why are small file transfers affected by concurrent large file transfers? Which is the bottleneck resource?

The following section addresses these questions.

III. IMPACT OF LARGE FILE TRANSFERS ON WEB PROXY CACHING PERFORMANCE

This section presents the impact of large file transfers on our *squid* web proxy cache, by answering the aforementioned four questions.

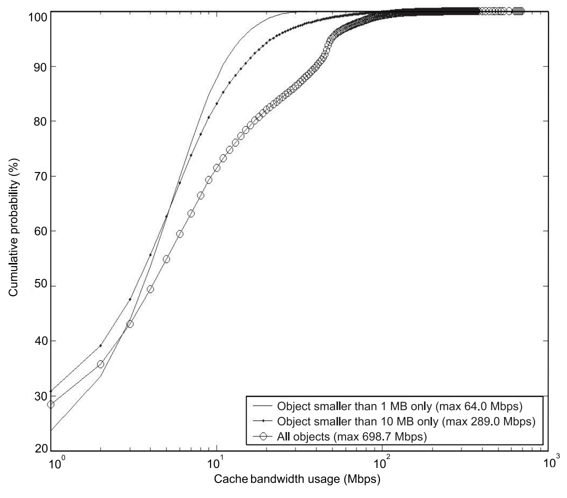
A. Impact on Cache Bandwidth Usage

In this subsection, we examine the impact of large object delivery on cache bandwidth usage. To this end, we calculated the average transfer rate of all transferred objects per second, based on the cache log data, for instance, the object size, request arrival time, and response times, recorded in the *squid* log file. Then, we obtained the cache bandwidth usage per second, by summing the transfer rates of all objects transferred at the time. Fig. 2 plots the cumulative distribution function of the cache bandwidth usage per selected trace, for KAIST and Waseda University, respectively.

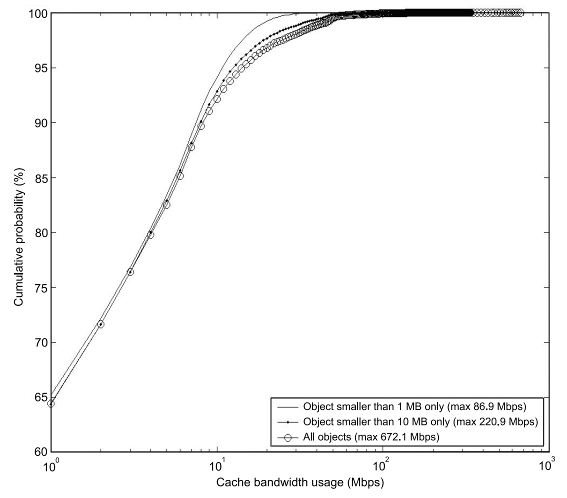
The figures show that the cache bandwidth usage exhibits heavier tailed characteristics with requests for large objects. In Fig. 2(a), without requests for objects larger than 1 MB, the 90th percentile, 99th percentile, and peak of the cache bandwidth usage are 11.4 Mb/s, 22.7 Mb/s, and 64.0 Mb/s, respectively. The average bandwidth usage is 5.7 Mb/s. It is clear that the cache's 1 Gb/s bandwidth resource is under-utilized throughout the experiment, with 7.38 million requests for small objects.

However, as the trace includes an additional few thousand requests for objects for which the size is between 1 MB and 10 MB, the 90th percentile, 99th percentile, and peak of the cache bandwidth usage increases to 15.2 Mb/s (1.4x), 53.1 Mb/s (2.4x), and 289.0 Mb/s (4.5x), respectively. The average bandwidth usage increases to 7.8 Mb/s (1.3x). In the next step, as the trace includes more requests for objects larger than 10 MB for which the access count comprises only 0.008% of the trace, the 90th percentile, 99th percentile, and peak of the cache bandwidth usage increases to 40.3 Mb/s (3.5x), 99.5 Mb/s (4.4x), and 698.7 Mb/s (10.1x), respectively. The average bandwidth usage increases to 14.7 Mb/s (2.5x).

We obtained similar results with the Waseda trace, as shown in Fig. 2(b). The volume of increased bandwidth usage due to requests for large objects was relatively smaller than that of the KAIST trace. Without requests for objects larger than 1 MB, the 90th percentile, 99th percentile, and peak of the cache bandwidth usage were 7.5 Mb/s, 18.8 Mb/s, and 86.9 Mb/s, respectively. The average bandwidth usage was 3.1 Mb/s. As with the aforementioned case, the cache's bandwidth resource is under-utilized throughout the experiment. As the trace includes additional requests for objects for which the size is between 1 MB and 10 MB, the 90th percentile, 99th percentile, and peak of the cache bandwidth usage increases to 7.9 Mb/s (1.1x), 32.3 Mb/s (1.7x), and 220.9 Mb/s (2.5x) respectively. The average bandwidth usage was 3.7 Mb/s (1.2x). In the next step, as the trace includes more requests for objects larger than 10 MB for which the access count comprises only 0.004% of the Waseda trace, the 90th percentile, 99th percentile, and peak of the cache bandwidth usage increase to 8.2 Mb/s (1.1x), 43.0 Mb/s (2.3x), and 672.1 Mb/s (7.7x), respectively. The average bandwidth usage increases to 4.1 Mb/s (1.3x). Table 3 summarizes the results.



(a)



(b)

Fig. 2. CDF of cache bandwidth usage with and without requests for large objects: (a) KAIST trace and (b) Waseda trace.

To summarize, the number of requests for large objects substantially increases, and governs the volume of the peak cache bandwidth usage (i.e., peak web traffic) in a high bandwidth access network, though it comprises less than 0.1% of the total access count. Without requests for objects larger than 1 MB, the peak volume of traffic that passes through our experimental web cache server decreases significantly, by 76–86%.

Here, we investigate what objects significantly contribute to peak traffic and to what extent. The aim of this investigation is to determine the threshold size of large objects that generates significant bandwidth overhead for a 1 Gb/s web cache.

To this end, all requested objects were categorized in four groups according to their size; less than 1 MB, $\mathcal{O}(1)$ MB, $\mathcal{O}(10)$ MB, and $\mathcal{O}(100)$ MB. Each group was further divided into two subgroups; cache-hit and cache-miss. Then, for each group of objects, we calculated and summed the transfer rate of concurrently delivered objects, per second. Finally, we obtained the volume of the greatest peak traffic generated by each group

Table 3. Cache bandwidth usage with and without request for large objects.

Trace contains requests for	Cache bandwidth usage (Mb/s)							
	KAIST trace				Waseda trace			
	90th kv.	99th kv.	Peak	Avg.	90th kv.	99th kv.	Peak	Avg.
Objects smaller than 1 MB only	11.4	22.7	64.0	5.7	7.5	18.8	86.9	3.1
Objects smaller than 10 MB only	15.2	53.1	289.0	7.8	7.9	32.3	220.9	3.7
All objects (the original traces)	40.3	99.5	698.7	14.7	8.2	43.0	672.1	4.1

Table 4. The volume of peak traffic generated by each group of large objects, categorized according to object size.

		Size of requested objects			
		Smaller than 1 MB	Between 1 MB and 10 MB	Between 10 MB and 100 MB	Larger than 100 MB
KAIST	Cache hit	39.1 Mb/s(143 req.)	197.5 Mb/s (3 req.)	332.2 Mb/s (1 req.)	652.3 Mb/s (1 req.)
	Cache miss	51.0 Mb/s (20 req.)	109.8 Mb/s (3 req.)	168.3 Mb/s (6 req.)	80.2 Mb/s (3 req.)
Waseda	Cache hit	39.0 Mb/s (116 req.)	219.3 Mb/s (5 req.)	355.2 Mb/s (1 req.)	668.1 Mb/s (1 req.)
	Cache miss	85.6 Mb/s (15 req.)	122.5 Mb/s (3 req.)	99.1 Mb/s (2 req.)	35.3 Mb/s (1 req.)

Table 5. Average and max latency taken to deliver small objects with and without requests for large objects.

Trace contains requests for	KAIST	Waseda
Objects smaller than 1 MB	14.3 ms / 12.9 sec	21.7 ms / 900.0 sec
Objects smaller than 10 MB	21.3 ms / 19.2 sec	32.4 ms / 1681.9 sec
All objects (real trace)	572.8 ms / 716.2 sec	1,503 ms / 2,591 sec

of objects. Table 4 summarizes the results.

For each entry of peak traffic in the table, the number of requests for similar-sized objects is shown in parentheses. As shown in the table, the volume of traffic generated by concurrent requests for small objects is at most 39.0–85.6 Mb/s, whereas only a few concurrent requests for large objects often generates 200 Mb/s or more of traffic. This begins to saturate the cache server's bandwidth resource. As expected, the missed requests for large objects do not generate many traffic surges, compared with the hit ones.

On a high bandwidth access network, even a single hit request for an object for which the size is tens or hundreds of MB often causes a traffic spike, and starts to saturate the bandwidth resource of a 1 Gb/s cache server, while missed requests for similar-sized objects do not. Only a few concurrent hit requests for objects for which the size is between 1 MB and 10 MB, such as MP3 audio files, also generate relatively high traffic for a 1 Gb/s cache server. The greater the number of downstream clients a cache server has, the higher the probability that they make concurrent requests for large objects. That is, the greater the user population serviced by a cache server, the greater the extent to which a cache server is affected by the bandwidth saturation problem.

B. Impact on Response Times

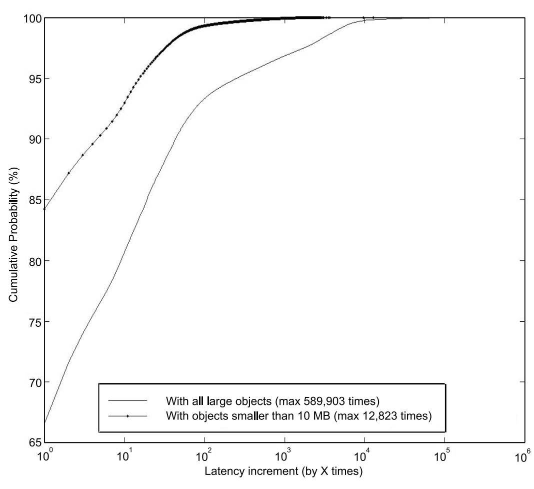
In this subsection, we examine the impact of large object delivery on the performance of web caching. To this end, we computed and compared the response time required for our cache server to deliver small objects for which the size is less than 1 MB to clients, with and without concurrent requests for large objects, respectively. Table 5 shows the average and worse case response times required for delivery of a small object, in each case.

When we eliminated the requests for objects larger than 1 MB from the KAIST and Waseda traces, the average response times required for delivery of small objects were 14.3 ms and 21.7 ms, respectively. The average and maximum response time of the Waseda trace are both higher than those of the KAIST trace. This is due to the concentration of the numerous requests in the Waseda trace in the afternoon; whereas, requests in the KAIST trace are more evenly distributed across the day and night. As a result, during peak times, requests in the Waseda trace generate more significant overheads than those in the KAIST trace.

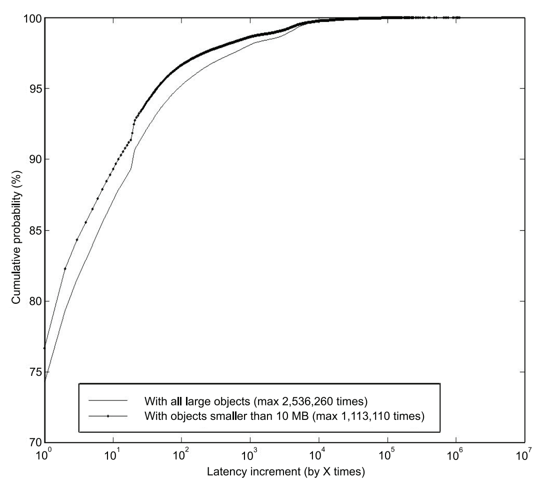
When requests for objects for which the size is between 1 MB and 10 MB are included, the average response times increase to 21.3 ms (1.5x) and 32.4 ms (1.5x). As the traces include more requests for objects larger than 10 MB, the average response times per trace significantly increase, to 572.8 ms (40.1x) and 1,503 ms (69.2x). This clearly shows that delivery of small objects is delayed by the extremely small number of requests for large objects, for which the access count comprises less than 0.1%.

To further investigate how many requests for small objects are directly affected by concurrently delivered large objects and to what extent; for every request for small objects, we compared its two response times in two different experiments; with and without concurrent requests for large objects. Fig. 3 shows the results in CDF. As shown in Fig. 3(a), as the KAIST trace includes requests for objects for which the size is between 1 MB and 10 MB, about 26% of requests for other small objects are delayed due to concurrently delivered large objects. For about 8% of the requests, the response time increases by more than a factor of ten. For about 0.7% and 0.06% of the requests, the response time increases by more than a factor of one hundred and one thousand, respectively. In the worst case, it increases by a factor of 12,823. When the KAIST trace includes all requests for objects larger than 10 MB, there is a much steeper increase in response times. In this case, a greater number of requests for small objects are even more severely delayed; around 34% of requests for small objects incur delays. For about 20% of the requests, the response times increase by more than a factor of ten. For about 6.8% and 3.2% of requests, the response times increase by more than a factor of one hundred and one thousand, respectively. In the worst case, it increases by a factor as high as half a million. Around 0.3% of requests failed.

We also obtained similar results using the Waseda trace, while the proportion of the number of delayed requests due to concurrently delivered large objects is smaller than that of KAIST. As shown in Fig. 3(b), as the Waseda trace includes requests for objects for which the size is between 1 MB and 10 MB, about 23%

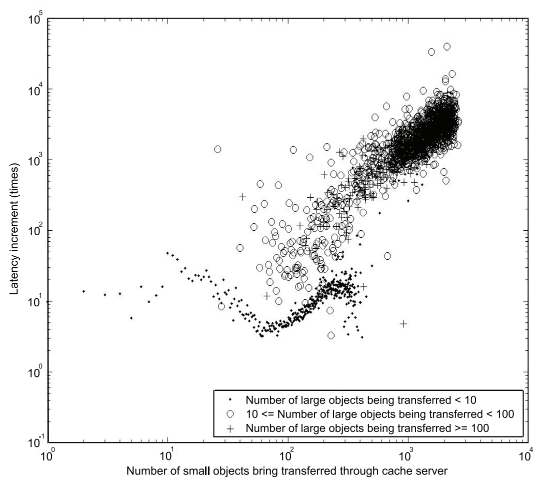


(a)

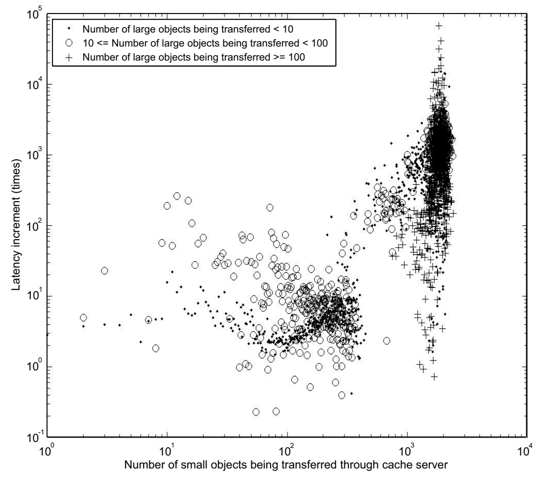


(b)

Fig. 3. Latency increase with concurrent requests for large objects: (a) KAIST trace and (b) Waseda trace.



(a)



(b)

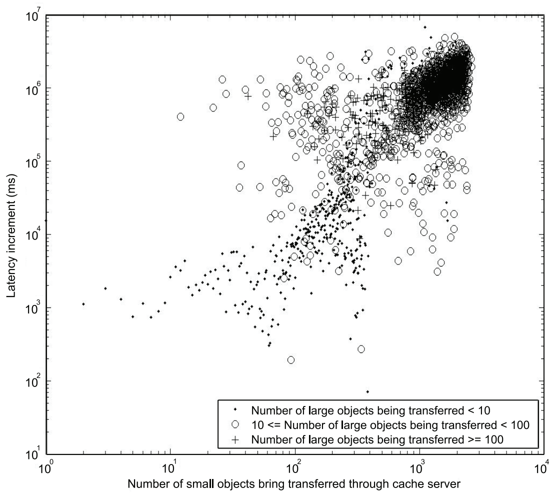
Fig. 4. Latency increase (in *X* times) based on the number of concurrent requests for small and large objects: (a) KAIST trace and (b) Waseda trace.

of requests for other small objects are delayed. For about 10.6%, 3.3%, and 1.3% of the requests for small objects, the response times increase by more than a factor of ten, one hundred, and one thousand, respectively.

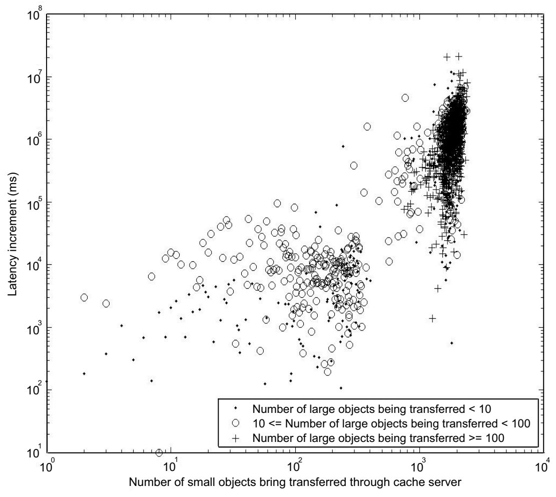
The worst case figure is even worse than that of the KAIST case; it increases by a factor of 1.1 million. When the Waseda trace includes all requests for objects larger than 10 MB, there is a relatively steeper increase in response time. As for the KAIST trace experiment, a greater number of requests for small objects are even more severely delayed; around 26% of requests for small objects incur delays. For about 12.7% of the requests, the response time increases by more than a factor of ten. For about 4.7% and 1.9% of the requests, the response time increases by more than a factor of one hundred and one thousand, respectively. In the worst case, it increases by a factor as high as 2.5 million. The request failure rate is 4.2%, a factor of fourteen higher than that of the KAIST trace.

Figs. 4 and 5 show the variation of response times for small objects, according to the number of concurrent requests for small and large objects. Fig. 4 depicts the average latency increase of small objects, particularly when there are n concurrent requests for small objects (as in the x-axis) and m concurrent requests for large objects (as in the legend). Fig. 5 shows the real delays (in milliseconds) that small objects incurred due to large objects. As shown in Fig. 4(a), when the number of large objects transferred is less than 10, and the number of small objects delivered is less than 100, the average latency increase is less than a factor of hundred in most cases. As the number of small objects exceeds 100 and the load on the cache server increases, the average response time increases by a factor of a few to tens of thousands. Fig. 4(b) of the Waseda trace, whilst it is not identical, shows a similar pattern. Usually, the greater the number of large objects the cache server processes, the greater the delay incurred by small objects.

Note that when the number of concurrent requests for small objects exceeds a few hundred to a thousand, for instance, during peak-times, even a few concurrent requests for large objects

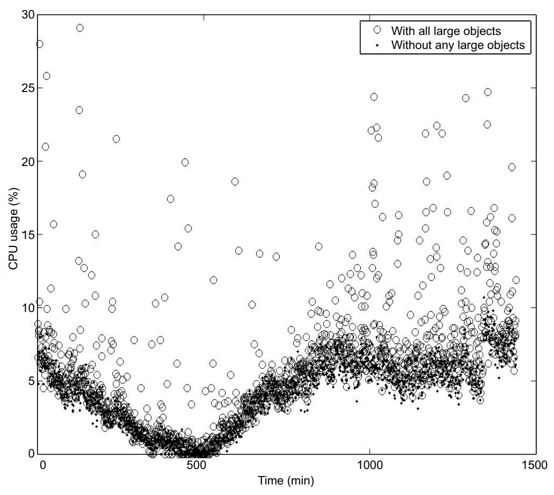


(a)

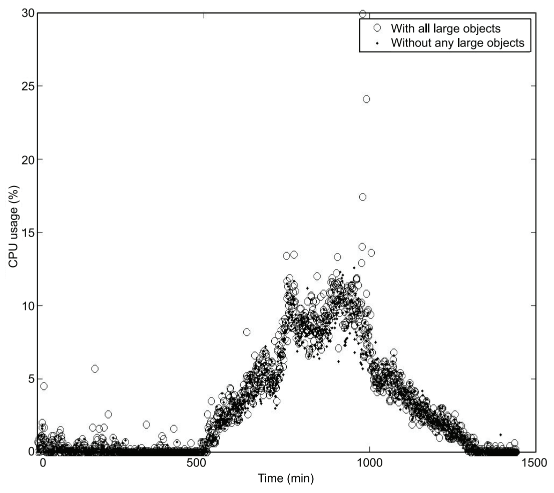


(b)

Fig. 5. Latency increase (in milliseconds) based on the number of concurrent requests for small and large objects: (a) KAIST trace and (b) Waseda trace.



(a)



(b)

Fig. 6. CPU usage with and without requests for large objects: (a) KAIST trace and (b) Waseda trace.

often causes significant performance degradation at the cache server, thereby causing huge delays for the delivery of small objects. Here, we argue that the frequent peak-time performance degradation bottleneck at cache servers is largely likely due to concurrent delivery of large objects, regardless of their number.

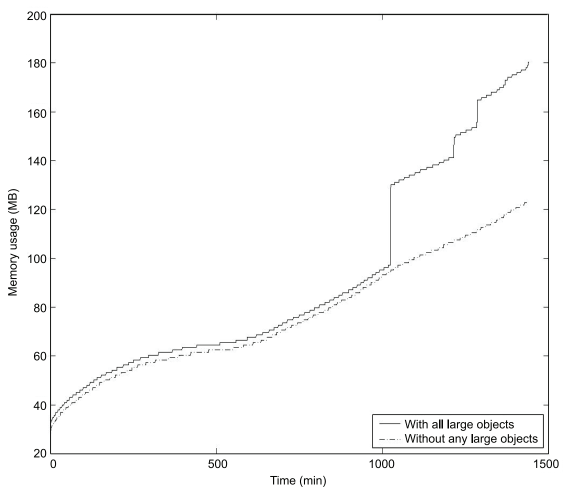
C. *Impact on CPU Usage, Main Memory Usage, and Disk I/O Operations*

Thus far, we have examined the impacts of large file transfers in terms of the delivery performance of concurrent small file transfers on our experimental *squid* web proxy cache server. The purpose of this subsection is to provide more insight into where the performance bottleneck arises, via performance profiling of various internal components other than bandwidth resource, i.e., CPU usage, main memory usage, and number of disk I/O operations, in the cache proxy. To this end, we use a popular system monitoring tool, viz., *bsdsar* [15].

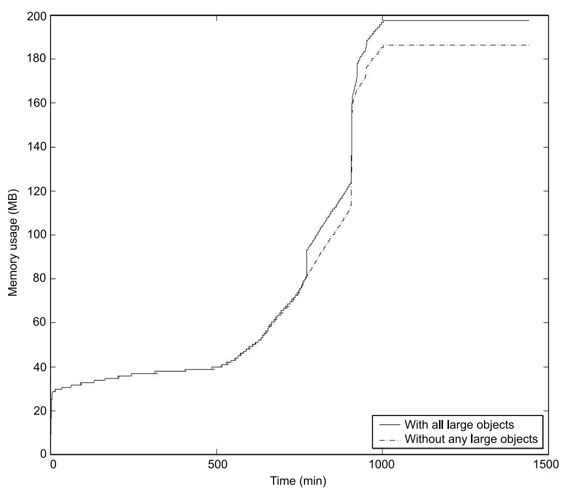
The CPU usage, main memory usage, and number of disk I/O operations of the cache box with and without requests for large objects are illustrated in Figs. 6, 7, and 8, respectively. As shown in these figures, none of these resources in the cache box are overwhelmed by requests for large objects. With requests for large objects, the CPU usage comprises at most 30%, and main memory usage is no more than 200 MB (only 10% of 2 GB). Additional disk overheads, in terms of the number of I/O operations due to requests for large objects, are also negligible (Fig. 8). The results we have presented thus far are congruent with the previous work; this states that the performance bottleneck of Internet servers consisting primarily of static content is due to the limited network bandwidth allocated to the server [16].

D. *Implications for Web Caching*

We found that requests for large objects from high bandwidth clients often generates sudden spikes, and governs the volume of peak loads in web traffic. We observed that requests for large

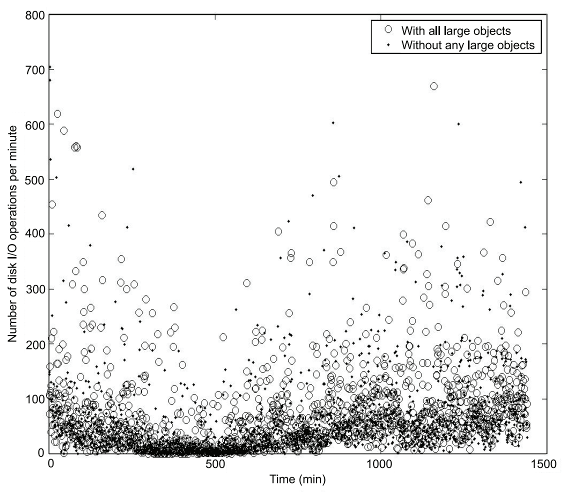


(a)

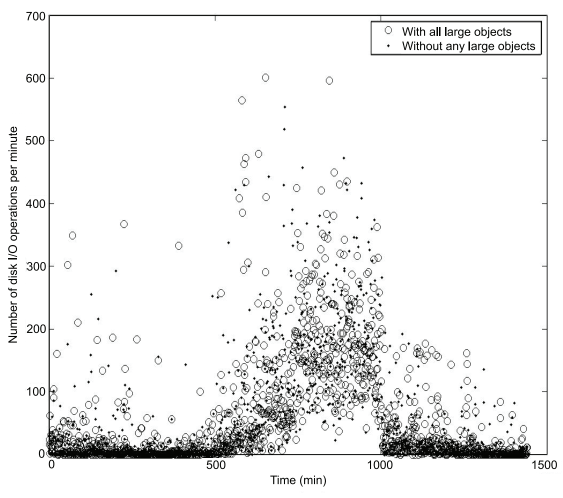


(b)

Fig. 7. Main memory usage with and without requests for large objects: (a) KAIST trace and (b) Waseda trace.



(a)



(b)

Fig. 8. Number of disk I/O operations with and without requests for large objects: (a) KAIST trace and (b) Waseda trace.

objects for which the access count comprises less than 0.1% of the total, often overwhelm the cache server and make the cache server a bottleneck in a web network. They delay other concurrently requested small objects.

Our cache server incurred significant performance degradation, due to delivery of large objects. Around 30% of requests for small objects were delayed. During non peak-times, they were delayed by a factor of a few to tens.

During peak-times, even a few large objects delayed them by a factor of hundreds of thousands to a few million. Around 0.3% and 4.2% of requests for small objects failed, due to concurrently requested large objects, for the KAIST and Waseda trace experiment, respectively.

We argue that the bandwidth bottleneck is due to the fundamental limitations of the current centralized web proxy caching model that scales poorly when there is a limited amount of dedicated resource. This is a serious threat to the viability of the current web proxy caching model, particularly in a high bandwidth access network, since it leads to sporadic disconnections of the downstream access network from the global web network.

As a result, some users have to wait for more than a few seconds to obtain a small text document, which was previously downloaded within just a few milliseconds, due to a few large objects concurrently requested by other users. Some business-oriented sites, where the so-called eight second rule is applied, may lose visitors and money, because users accessing the sites are restricted at their local cache server.

That is, the *squid* cache proxy, in world-wide deployment for the last ten years,[5] is very vulnerable to attacks from malicious clients that repeatedly request very large objects through their high bandwidth pipes with malicious intent. Whilst we have only examined the *squid* proxy cache in this paper, we postulate that many other *squid*-based open-source cache servers, as well as commercial cache servers, are also vulnerable to this prob-

[5] *Squid* is known to capture at least 80% of the caching proxy market in Europe and about 70% in the US [12].

lem in a high bandwidth network, unless they have an additional mechanism for handling large-sized objects.

We postulate that the frequent peak-time performance degradation bottlenecks at cache servers are likely to be due to concurrent delivery of large objects. This is likely to be the reason why users in a high bandwidth network often complain about web cache servers installed on their access network. We also postulate that the transmission failure of web objects reported in [4] is not only due to the problem of wide-area Internet dynamics. Rather, at present, it is more likely to be due to the local cache bottlenecks.

Objects larger than $\mathcal{O}(10)$ MB had a relatively significant impact on the experimental cache. More than tens of concurrent requests for objects for which the size is between 1 MB and 10 MB affected its performance to a significant extent. This threshold size varies according to the downstream user population and number of concurrent user requests. In a larger scale access network where there are so many downstream users that more than hundreds or thousands of users often request MP3 objects concurrently, the threshold size is a few MB for a multi-Gb/s cache. In such an environment, even clients with 10/100 Mb/s connectivity will often cause the same problem for their upstream web proxy cache. In the same context, we postulate that even when a web proxy cache has 10 Gb/s connectivity, many downstream clients concurrently downloading $\mathcal{O}(10)$ MB objects with 100 Mb/s–1 Gb/s connectivity will often cause the same problem, in a large scale access network.

We postulate that web proxy caching continues to be one of the most popular information provisioning techniques on the Internet in the era of multimedia networking. However, our negative results naturally lead to reevaluation of the current web caching scheme.

IV. PEER-TO-PEER COOPERATIVE WEB CACHING

Thus far, we have studied the impact of large file transfers on the performance of web proxy caching in a high bandwidth network. This section proposes a peer-to-peer cooperative web caching scheme to solve the cache bottleneck problem. We evaluate the proposed scheme.

A. Why Peer-to-Peer?

The main idea of our scheme is based on the following; (1) many large web objects were previously downloaded and cached in a desktop machine's local web cache space. Why not share large objects among them directly, as many peer-to-peer applications do, rather than via a cache server, which often becomes a bandwidth bottleneck when handling large objects? (2) According to our previous study [17], the default size of the local web cache space in a desktop machine is generally configured to 3% or more of its total disk size. This comprises approximately 1 GB of the 33 GB average disk space installed on each machine. In most cases, 1 GB should be sufficient to cache and share large objects that were previously downloaded, for a few days to a week, since 98.8% of clients logged in the KAIST and Waseda traces had previously downloaded less than 100 MB of large objects over 96 hours, as shown in Table 6.

Table 6. Proportion of clients according to the total volume of downloaded objects (for 96 hours).

Trace	< 10 MB	> 10 and < 100 MB	> 100 MB and < 1 GB	> 1 GB
KAIST	69.3%	29.5%	1.2%	0.03%
Waseda	78.4%	20.5%	1.1%	0%

This summarizes the volume of large objects that had previously been downloaded and stored in each machine's disk space. We obtained the same results for both traces; of the machines that had downloaded at least one large object during the four days of the collection period, 99% had downloaded less than 100 MB of large objects. Only a few machines, 1%, had downloaded more than 100 MB of large objects.

The disk size doubles every 12 months [18] and the average disk usage is around 50–66% [17]. Therefore, it seems both feasible and cost effective to exploit each peer's local web cache space as a part of a large virtual cooperative web caching system to share large objects in their disk storage space, particularly on a high bandwidth network where every desktop machine has more than 100 Mb/s connectivity.

(3) We avoided modifying the conventional web caching model, to maximize the performance, backward compatibility and deployability of the proposed system. Small objects are generally delivered to users within a few milliseconds, unless they are concurrently transferring large objects through the same cache server. Accordingly, our scheme adopts a hybrid web caching model, where large objects are cached and delivered using desktop machines' resources, while small objects are still cached and transferred through a cache server, as in the conventional web caching scheme.

As in the conventional web caching system, our model lets HTTP requests from clients be directed to a local web cache server. Then, the cache first checks if the requested object has been cached. If a cache-miss occurs, it fetches the requested object from the original web server, and relays it to the requesting client. If a cache-hit occurs, the size of the requested object is checked. If it is less than the pre-configured threshold, the object is delivered from the cache server to the requesting client, as usual. Otherwise, if it is larger than the pre-configured threshold, the request is redirected by the cache server to other desktop machines holding the object in their local cache space.

We adopt the redirection mechanism proposed in [19], for redirection operations. This is a generalization of the well-known HTTP redirection 3xx codes. Our proxy cache server only needs to maintain a table of addresses of the desktop machines that have recently requested large objects. Considering that an IP address is four bytes and the number of unique large objects that have been requested over four days is less than ten thousand in both the KAIST and Waseda traces, only a few megabytes of memory is sufficient to maintain 10–100 addresses of desktop machines per requested large object. This server-based request redirection enables a client to locate the requested large object within just two application-level hops—the first to the cache server and the second to peers. The additional overhead required to redirect each request is negligible compared to the transmission delay required for delivery of an entire large object, since redirection is performed to a peer node within less

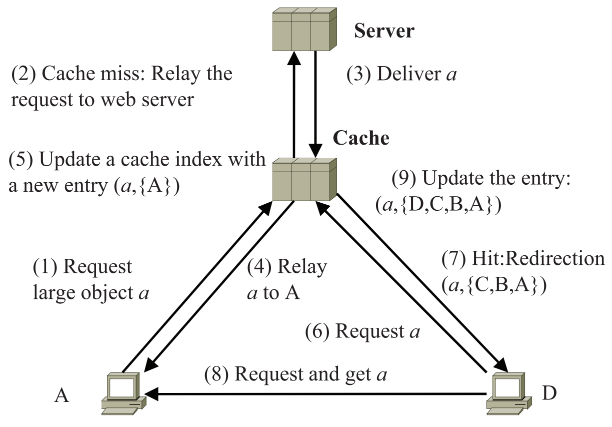

Server
(2) Cache miss: Relay the request to web server
(3) Deliver *a*
Cache
(5) Update a cache index with a new entry (*a*,{A})
(9) Update the entry: (*a*,{D,C,B,A})
(1) Request large object *a*
(4) Relay *a* to A
(7) Hit:Redirection (*a*,{C,B,A})
(6) Request *a*
(8) Request and get *a*
A
D

Fig. 9. Proposed model.

than a few milliseconds on the same campus network, and involves transferring just tens to hundreds of bytes of data.

Fig. 9 illustrates the proposed model. It consists of two components: A modified web cache server and a set of modified downstream desktop machines that also serve as peer caches for large objects. Let us assume that clients A, B, and C have recently requested large object *a*. Client D now requests the same object. The cache server responds to the request by sending a list of addresses, {C, B, A}, to D. Then, client D selects the peer (or peers, if it is designed to download from multiple sources) from which it will download the object.

Since (1) we assume that all peers are located close to one another on a single high bandwidth campus network, and (2) the number of requests for large objects and number of peers that have requested large objects over a few days has roughly the same order of magnitude, the proposed caching system does not need a complex algorithm for the peer selection process. Any peer selection algorithms that consider: (i) If the requested object is currently available in the peer, (ii) the maximum bandwidth the peer allocated to serve other clients (this is configured by its owner), (iii) the number of requests the peer currently serves to others, and (iv) the current CPU usage of the peer, should be sufficient for the proposed system.

When an error occurs during object transmission due to the selected peer's sudden departure from the system, the client contacts the next good peer and invokes partial fetching to fetch the rest of the data and construct a complete one, as in [4]. If the client does not find any available peers where the requested object is stored, and the object is still available in the cache server, the request is satisfied by the cache server. This simple fall-back mechanism enables the proposed system to switch back to a conventional web caching system at the cost of a few additional application-level operations. The following subsection presents our preliminary results from the trace-driven experiment that evaluates the feasibility and performance of the proposed web caching scheme.

B. Simulation Environment

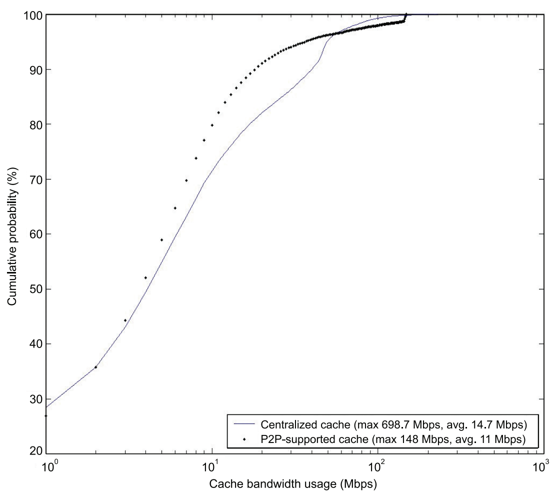
We used the emulated high bandwidth network environment shown in Fig. 1 for the evaluation. We used *simclient* to generate a stream of real-time requests based on an input trace, a modified version of *squid* for our modified cache server, *dummynet* to set the delay and bandwidth, and *webulator* to emulate web servers. The communications between the modified cache server and peers used in our experiment are based on Fig. 8. After the first request for a large object is served through the cache server, the following requests for the same large object are directed to our own simple simulator, viz., *simpeer.*

At this stage, the *simpeer* simulator logs the request arrival time, object URL, IP address of the requesting client, and size of the requested object. After all requests in the re-generation process conducted by *simclient* are completed, *simpeer* independently simulates peer characteristics based on the input log file and parameters, such as the volume of the network bandwidth and local cache space allocated by each peer. In *simpeer*, a new peer requesting a previously requested large object is redirected to the peer that most recently downloaded the same object. For simplicity, we assume that all peers allocate the same volume of upstream network bandwidth. The effects of the heterogeneous network bandwidth environment on the performance of the proposed system are left as our future work. We varied the upstream network bandwidth of all peers between 500 kb/s and 100 Mb/s to determine what volume of peer upstream bandwidth was required to make the proposed system operate effectively, without significant performance degradation. The download speed of each large object from a peer was calculated as the quotient of the maximum upstream bandwidth of the peer / the number of requests it is processing. All peers that had downloaded any large objects were assumed available over the whole simulation period, 24 hours.

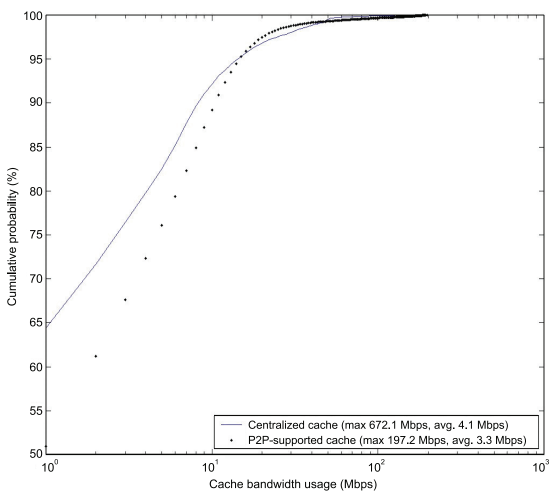
More detailed parameters on dynamic peer characteristics, such as the peers' uptime, downtime and CPU usage of each peer can also affect the transmission rate among peer nodes; however, these are not considered, since the purpose of this simulation is generally: (1) To show if our idea of directing requests for large objects from the cache server to desktop machines deals with the cache server bottleneck, (2) to determine how much overhead the proposed scheme generates for each peer in terms of additional network bandwidth consumption. As we observed in the previous section, other resources such as CPU and disk storage space are not primary bottlenecks when transferring large objects. In this paper, we assume that objects larger than 1 MB are large ones for this experiment. The size of each redirection message, in which multiple addresses of peers are contained, was configured to 30 bytes. All the other parameters, such as the cache server's bandwidth and round trip times, are shown in Fig. 1. We used the selected KAIST and Waseda traces described in Table 2.

C. Simulation Results

Fig. 10(a) plots the cumulative probability distribution of the cache server bandwidth usage for the KAIST trace, for two cases. The first is the conventional web caching scheme, and the other is for the proposed caching scheme. We found that the proposed scheme reduces the cache server bandwidth usage significantly. With the proposed scheme, the volume of peak traffic at the cache server was reduced by 78.8%, from 698.7 Mb/s to 148 Mb/s. Average traffic volume decreases from 14.7 Mb/s to 11 Mb/s. The volume of the additional traffic generated in

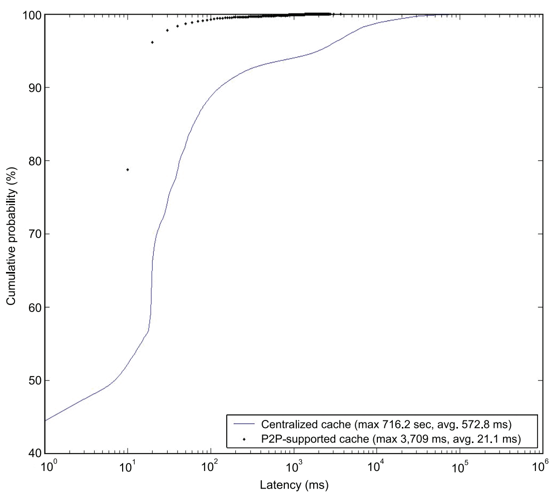


(a)

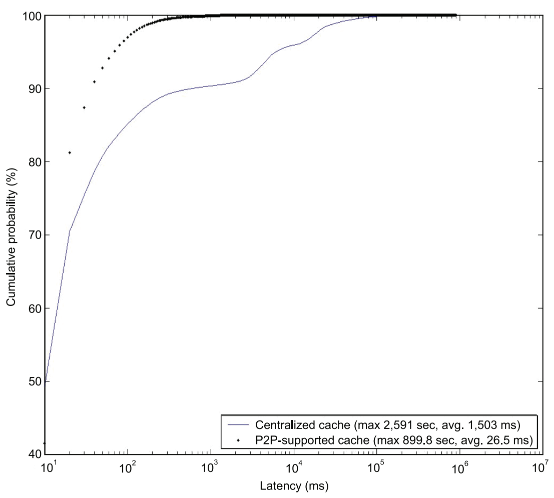


(b)

Fig. 10. CDF: Cache server bandwidth usage: (a) KAIST trace and (b) Waseda trace.



(a)



(b)

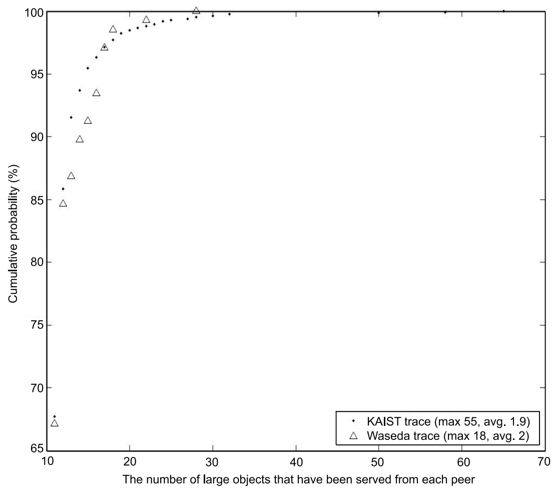
Fig. 11. CDF: Latency taken to deliver small objects: (a) KAIST trace and (b) Waseda trace.

redirecting 5,669 requests for large objects was only 265.7 kB over the one day experiment. Fig. 10(b) plots the cumulative distribution of the cache server bandwidth usage for the Waseda trace. With the proposed scheme, the volume of the peak traffic at the cache server was reduced by 70.7%, from 672.1 Mb/s to 197.2 Mb/s. Average traffic volume decreases from 4.1 Mb/s to 3.3 Mb/s.

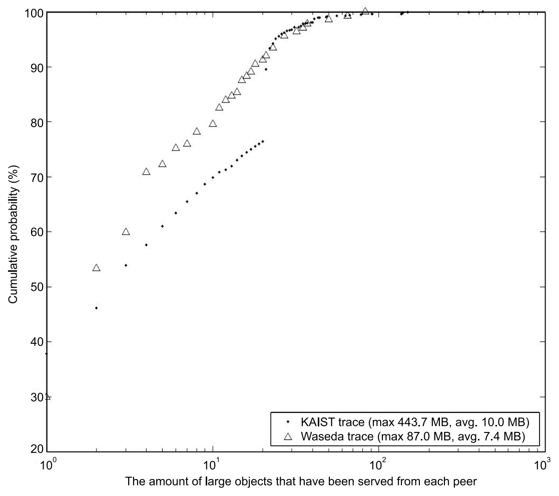
Fig. 11(a) plots the cumulative probability distribution of the latency required for delivery of small objects from the cache server, for the KAIST trace. The first is the conventional web caching scheme, and the other is for the proposed caching system. The proposed scheme significantly reduces the latency required for delivery of small objects, to which 99.88% of all web requests are directed. With the proposed scheme, the maximum latency is significantly reduced, from 716.2 sec to 3,709 ms. Average latency decreases from 572.8 ms to 21.1 ms. Fig. 10(b) plots the results obtained using the Waseda trace. With the proposed scheme, the maximum latency is reduced from 2,591 sec to 899.8 sec. Average latency significantly decreases from 1,503 ms to 26.5 ms. Compared to the results shown in Table 5, these results indicate that delivery of small objects is neither delayed nor failed due to requests for large objects, when our peer-to-peer caching scheme is used for caching and delivery of large objects.

A critical requirement of a peer-to-peer system running as a background job on desktop machines is that the extra load it imposes on the peer is low, and that a high load is never sustained for long [20]. Therefore, we quantify the extra load on participating peers in the proposed system. Fig. 12 depicts the cumulative probability distribution of the number and the volume of large objects that had been served by each peer. For the experiment with the KAIST trace, the number of peers that had served other peers at least once was 1,252, 76.1% of all group members. Of these, 1,230 nodes, 98.2% of the 1,252 peers, had served less than 10 requests to other peers over a one day period. The busiest, most popular one, had served 55 requests to other peers over the entire day, transferring a total of 91.5 MB. On average, peers had served 1.9 large objects, transferring 10.0 MB. For the same experiment with the Waseda trace, the number of peers that had served other peers at least once was 137, 25.1% of all group members. Of these, 135 nodes, 98.5% of the 137 peers, had served less than 10 requests to other peers. The busiest, and popular one, in terms of the number of objects that had been served, had served at most 18 requests to other peers over the entire day, transferring a total of 37.4 MB. On average, peers had served two large objects, transferring 7.4 MB over 24 hours.

The results of the performance evaluation presented in this section indicate that the proposed system in a high bandwidth network performs the task of caching and delivery of large objects in an efficient and cost-effective manner, without generating significant overhead for participating peers.



(a)



(b)

Fig. 12. CDF: The number and total volume of large objects served by each peer: (a) Number of large objects served by each peer and (b) total volume of large objects served by each peer.

V. COMPARISON TO RELATED WORK

Over the previous decade, the research on improving web performance and workload characterization has attracted significant attention [2], [3], [6], [9], [13]. These studies have shown that web workloads exhibit significant short-term burstiness (self-similarity) and are heavy-tailed. The performance benefits of web caching in reducing the average time, network bandwidth usage and server load has been shown in [9], [21], [22]. Most of these studies focused on the average load, or issues such as the impact of low bandwidth connections and non-cacheable objects; while our study investigates the impact of the delivery of large objects on web cache servers in a high bandwidth network environment. A noteworthy exception, Raunak *et al.* [6], found that unlike the substantial impact they have on the average load, web caches have a limited impact on the tail of the load distribution, primarily due to the characteristics of large objects; even a single cache-miss for an extremely large object can often cause a large volume of traffic to be fetched from its distant original web server. They focused on the reduced effects of web caching on the tail of the load distribution, which is complementary to this paper.

Surprisingly, there have only been a few studies on large object transfers in web proxy caching. Jung *et al.* pioneered this topic. They found that the transmission performance and storage utilization at web cache servers was poor, especially in delivering large objects, since there is a fundamental trade-off between a cache server's internal resource utilization and improving the performance [23]. They also found that large multimedia data transfers often fail over wide-area networks. They proposed an on-demand cumulative pre-fetching scheme, to avoid unnecessarily re-fetching of the entire body of failed objects [4]. While their work focused on measuring and improving the quality of large object transfers through a web proxy cache, this paper focuses on the cache bandwidth bottleneck problem caused by large object transfers; we measured the extent to which these large object transfers affect the performance of cache servers in handling requests for the other concurrent small objects for the first time. We then solved the bandwidth bottleneck problem. Since the replicache system proposed in [23] is solely designed to improve the quality (i.e., speed) of large object transfers without consideration of the quality of concurrent small object transfers, the proposed system makes the cache bandwidth bottleneck problem worse.

The most obvious solution to overcome the bandwidth bottleneck is to upgrade the memory and bandwidth resources, when it is both feasible and cost effective to do so, with commercially available cards and modules. When upgrading cannot solve the scalability problem, tightly-coupled clusters of machines have been deployed to provide a scalable web caching service [24]. However, while a cluster-based centralized web caching scheme may be scalable to a certain degree, in terms of the user population and number of concurrent user requests in a low bandwidth network environment, it is neither cost-effective nor scalable in a high bandwidth network environment, where (1) The bandwidth of numerous downstream clients approaches that of the local web caching facility and (2) there are popular objects for which the size is so large that even a small number of concurrent requests for them can saturate a cache server's resources easily. In such an environment, it will eventually become a bottleneck in handling sudden peak traffic generated by concurrent requests for very large objects. While the problem can be alleviated to a certain degree by resource overprovisioning and over-investment, it is too expensive to install and maintain [20], considering that the requests for large objects comprise less than 0.1% of total access counts.

From a technical perspective, a distributed web caching system [24]–[28], where multiple cache servers are deployed at multiple locations, may be a candidate solution. Much work has been done on distributed cooperative web caching systems over the last decade. However, this has generally focused on reducing upstream regional, domestic or wide-area network traffic, by making cooperative cache servers, across multiple access networks. Installing and maintaining multiple servers at multiple locations within a single access network (e.g., campus network) to handle requests for large objects comprising less than 0.1% of total access counts is surely too expensive.

A dedicated server-based approach, whether it is centralized or distributed, has the fundamental limitation that the fixed volume of memory and bandwidth resources allocated to the system limits the maximum throughput of the system. Our analyses in the previous section showed that the cache server's performance is significantly degraded, even when it handles a limited number—only tens—of concurrent requests for large objects. The bandwidth and memory cost required for delivery of each large object is too high. Therefore, conventional server-based web caching systems cannot provide a solution that is both scalable and cost-effective, particularly when many downstream clients concurrently request large objects. We need a more intelligent web caching scheme to solve the cache bottleneck problem. The volume of burst traffic caused by concurrent requests for large objects is reduced to a level where the cache server does not become overwhelmed by them. Thereby, requests for small objects are not delayed.

Harchol-Balter *et al.* [16], [29], [30] is a recent noteworthy work, although it addresses the problem of web server performance, rather than web proxy cache servers. Traditionally, requests at a web server are handled via FAIR scheduling: The web server allocates its resources fairly among those requests ready to receive services (as with the *squid* proxy cache). Instead, they proposed *unfair* scheduling, in which they give higher priorities to requests for small files or requests with a small remaining file size, in accordance with the shortest remaining processing time (SRPT) scheduling policy. Results indicate that SRPT-based scheduling of connections yields significant reduction in delays at the web server, and hardly penalizes requests for large files. We are investigating the extent to which the size-based scheduling technique can alleviate the cache bottleneck problem caused by transfer of very large objects. The results will appear in a future paper.

Iyer *et al.* proposed a fully decentralized peer-to-peer web caching scheme called squirrel [20]. It needs no dedicated infrastructure other than the resources allocated by desktop machines. On top of a decentralized request routing and object location scheme called pastry [31], squirrel gathers and pools resources from desktop machines, to achieve comparable performance to a dedicated web cache server. An obvious drawback of squirrel's fully decentralized approach is that it is affected by additional overhead in the request routing and object location process, due to its fully decentralized indexing, lookup, and routing schemes. A distributed lookup scheme, such as chord [32] and pastry [31], has a lookup cost in the order of $\mathcal{O}(\log(N))$ operations, where N is the number of nodes in the system. The lookup delays are regarded as negligible when large objects are delivered; however, for small objects that are generally delivered to users within a few milliseconds, the overheads can often be a significant proportion of the transaction latency. They thereby degrade the overall system performance significantly. To avoid such unnecessary delays in handling small objects that comprise 99.9% of total access counts, while exploiting the benefits of the peer-to-peer content sharing model in handling large objects, we adopted a hybrid web caching scheme, where large objects are cached and delivered using desktop machines' resources; while small objects are still cached and transferred through a cache server, as in conventional web caching schemes. More recently, [33]–[37] proposed and evaluated the performance of various peer-to-peer web caching schemes. Nonetheless, none of the work focused on the performance impact of large object transfers on web proxy caching.

Our proposed caching scheme is a combination of the concept of cooperative web caching [24]–[28] and peer-to-peer multimedia sharing applications. Since the proposed system maintains an object index at a local cache server and redirects incoming requests to other candidate peers, it is more like the distributed Internet cache model [27], rather than the hierarchical web cache model [28]. For the same reason, it is more like napster [38] where a centralized index is maintained, rather than fully decentralized systems such as gnutella [39] and squirrel [20]. The idea of block-level file transfer and replication among participants has been used in various systems, such as swarmcast [40], edonkey [41], KaZaA [42], and logistical networking [43]. We leave the adoption of such an idea (i.e., slicing a large file into many smaller objects and managing them in a distributed manner among peers) for our future work. Padmanabhan *et al.* [19] and Stading *et al.* [44] proposed a peer-to-peer model to address the flash crowd problem of web servers, rather than cache servers. Squirrel is proposed as a fully decentralized scalable peer-to-peer routing-based scheme. It needs no dedicated infrastructure, other than the desktop machines themselves [20]. To the best of our knowledge, this is the first study that explicitly defines, addresses and solves the cache server bandwidth bottleneck problem, particularly the one caused by concurrent requests for large objects from downstream clients in a high bandwidth network.

VI. CONCLUSION

In this paper, we studied the impact of large file transfers on the performance of web proxy caching in a high bandwidth network environment. We conducted a series of thorough performance analyses and profiling of various resource components in our experimental *squid* proxy cache server. The analyses shows that the requests for large objects, for which the access count comprises less than 0.1% of the total, often overwhelm the cache server's bandwidth resource. This makes it a bottleneck in web content delivery. We argue that the cache bandwidth bottleneck problem caused by large objects can lead to sporadic disconnections and isolation of users' access network from the global web network.

We proposed a peer-to-peer cooperative web caching scheme to address the bandwidth bottleneck problem and then showed that it performs the task of caching and delivery of large objects in an efficient and cost-effective manner, without generating significant overhead for participating peers. We have also examined previous related work and compared them to the proposed scheme.

ACKNOWLEDGMENT

We would like to thank all those who allowed us to access to their logs, without whom this research would have been impossible; Professor Shigeki Goto and Atsushi Ito at Waseda University, and Sun-kyu Kim and Taejin Yoon at KAIST. We are also grateful to Duane Wessels and Kc Claffy for their invaluable feedback. Our special thanks to anonymous reviewers, whose feedback was vital to improve the presentation of this work.

REFERENCES

[1] A. Myers, J. Chuang, U. Hengartner, Y. Xie, W. Zhang, and H. Zhang, "A secure, publisher-centric web caching infrastructure," in *Proc. INFOCOM*, USA, Apr. 2001.

[2] M. Arlitt, R. Friedrich, and T. Jin, "Workload characterization of a web proxy in a cable modem environment," *ACM SIGMETRICS Performance Evaluation Review*, vol. 27, no. 2, pp. 25–36, Sept. 1999.

[3] M. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: Evidence and possible causes," *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp. 835–846, Dec. 1997.

[4] J. Jung, D. Lee, and K. Chon, "Proactive web caching with cumulative prefetching for large multimedia data," *Computer Networks*, vol. 33, pp. 645–655, May 2000.

[5] J. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolsky, "Internet backplane protocol: Storage in the network," in *Proc. Netstore Symp.*, Oct. 1999.

[6] M. Raunak, P. Shenoy, P. Goyal, and K. Ramamritham, "Implications of proxy caching for provisioning networks and servers," *IEEE J. Sel. Areas. Commun.*, vol. 20, no. 7, pp. 1276–1289, Sept. 2002.

[7] Squid Web Proxy Cache. [Online]. Available: http://www.squid-cache.org

[8] A. Ogawa, K. Kobayashi, K. Sugiura, O. Nakamura, and J. Murai, "Design and implementation of DV-based video over RTP," *WIDE Workshop*, Stanford University, Jan. 2002.

[9] A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich, "Performance of web proxy caching in heterogeneous environments," in *Proc. IEEE INFOCOM*, Mar. 1999.

[10] Proxycizer. [Online]. Available: http://www.cs.duke.edu/ari/cisi/Proxycizer

[11] L. Rizzo, "Dummynet: A simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, Jan. 1997.

[12] J. Cooper. (2001, July). Squid cache market penetration. [Online]. Available: http://www.squid-cache.org/mail-archive/squid-users/200107/0639.html

[13] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *Proc. INFOCOM*, Mar. 1999.

[14] J. P. Savage. (2005, Aug.). "A letter to the U.S. congress." [Online]. Available: http://www.ftthcouncil .org/documents/247684.pdf

[15] Bsdsar. System Activity Reporter. [Online]. Available: http://www.googlebit.com/bsdsar

[16] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal, "Size-based scheduling to improve web performance," *ACM Trans. Comput. Syst.*, vol. 21, no. 2, pp. 207–233, May 2003.

[17] H. Kim, D. Lee, J. Lee, J. Suh, and K. Chon, "A measurement study of storage resource and multimedia contents on a high-performance research and education network," in *Proc. IEEE High Speed Network and Multimedia Communications*, July 2003.

[18] I. Foster, "P2P and Grid Computing," in *Internet2 Peer-to-Peer Workshop: Collaborative Computing in Higher Education—Peer-to-Peer and Beyond*, Jan. 2002.

[19] V. N. Padmanabhan and K. Sripanidkiulchai, "The case for cooperative networking," in *Proc. Int. Workshop on Peer-to-Peer Systems*, Feb. 2002.

[20] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized peer-to-peer web cache," in *Proc. ACM Symp. Principles of Distributed Computing*, July 2002.

[21] A. Rousskov and V. Soloviev, "On performance of caching proxies," in *Proc. ACM SIGMETRICS*, June 1998.

[22] M. Crovella and P. Barford, "The network effects of prefetching," in *Proc. IEEE INFOCOM*, 1998.

[23] J. Jung and K. Chon, "RepliCache: A new approach to scalable network storage system for large objects," in *Proc. Int. Web Caching Workshop*, Apr. 1999.

[24] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier, "Cluster-based scalable network services," in *Proc. ACM Symp. Operating System Principles*, Oct. 1997.

[25] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson, "Adaptive web caching: Towards a new global caching architecture," *Computer Networks and ISDN Systems*, vol. 30, pp. 22–23, Nov. 1998.

[26] R. Tewari, M. Dahlin, H. M. Vin, and J. Kay, "Beyond hierarchies: Design considerations for distributed caching on the Internet," in *Proc. Int. Conf. Distrib. Comput. Syst.*, June 1999.

[27] D. Povey and J. Harrison, "Distributed internet caches," in *Proc. Australian Computer Science Conf.*, Feb. 1997.

[28] A. Chankhunthod, P. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrel, "A hierarchical internet object cache," in *Proc. USENIX Technical Conf.*, Jan. 1996.

[29] B. Schroeder and M. Harchol-Balter, "Web servers under overload: How scheduling can help," *ACM Trans. Internet Technologies*, vol. 6, no. 1, pp. 20–52, Feb. 2006.

[30] M. Crovella, B. Frangioso, and M. Harchol-Balter, "Connection scheduling in web servers," in *Proc. USENIX Symp. Internet Technologies and Syst.*, Oct. 1999.

[31] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location, and routing for large-scale peer-to-peer systems," in *Proc. Int. Conf. Distrib. Syst. Platform*, Nov. 2001.

[32] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," in *Proc. ACM SIGCOMM*, Aug. 2001.

[33] Z. Xu, Y. Hu, and L. Bhuyan, "Exploiting client caches: A scalable and efficient approach to build large web cache," in *Proc. Int. Parallel and Distrib. Process. Symp.*, Apr. 2004.

[34] P. Linga, I. Gupta, and K. Birman, "A churn-resistant peer-to-peer web caching system," in *Proc. ACM Workshop on Survivable and Self-regenerative Systems*, Oct. 2003.

[35] P. Linga, I. Gupta, and K. Birman, "Kache: Peer-to-peer web caching using kelips," in submission, June 2004.

[36] W. Shi and Y. Mao, "Performance evaluation of peer-to-peer web caching systems," *J. Syst. Softw.*, vol. 79, no. 5, pp. 714–726, May 2006.

[37] Y. Mao, Z. Zhu, and W. Shi, "Peer-to-peer web caching: Hype or reality?" in *Proc. Int. Conf. Parallel and Distrib. Syst.*, July 2004.

[38] Napster. [Online]. Available: http://www.napster.com

[39] Gnutella Open Source Community. [Online]. Available: http://gnutella.wego.com

[40] Project Swarmcast. [Online]. Available: http://sourceforge.net/projects/swarmcast

[41] Edonkey. [Online]. Available: http://www.edonkey2000.com

[42] KaZaA. [Online]. Available: http://www.kazaa.com

[43] M. Beck, T. Moore, and J. Plank, "An end-to-end approach to globally scalable network storage," in *Proc. ACM SIGCOMM*, Aug. 2002.

[44] T. Stading, P. Maniatis, and M. Baker, "Peer-to-peer caching schemes to address flash crowds," in *Proc. Int. Workshop on Peer-to-Peer Syst.*, Feb. 2002.

Hyun-Chul Kim is a BK Assistant Professor at Seoul National University. Prior to joining Seoul National University, he had worked at the Cooperative Association for Internet Data Analysis (CAIDA, based at San Diego Supercomputer Center, UC San Diego) as a postdoctoral visiting scholar from 2006 to 2007. He received his B.S., M.S., and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 1995, 1997, and 2005, respectively. His research interests include Internet traffic classification, contents-oriented networking, analysis and modeling of end-host and human behavior pattern observed in the Internet applications such as peer-to-peer file sharing applications, MMORPGs, etc.

Dongman Lee received his B.S. degree in Computer Engineering from Seoul National University, South Korea in 1982, and M.S. and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea in 1984 and 1987, respectively. From 1988 to 1997, he worked as Technical Contributor at Hewlett-Packard. From 1998 to March, 2004, he joined, as an Associate Professor, School of Engineering, Information and Communications University (ICU). From April 2004, he became a Professor at ICU. From March 2009, he is Professor at Computer Science Department, KAIST. He is Director of Urban Computing Research Center. He received a Prime Minister Award as the recognition on the advancement of the Korean Internet in 2000 and Internet Technical Achievement Award at KRNet07 in 2007. He has served as a TPC member of numerous international conferences including IEEE COMPSAC, Multimedia, PDCS, PERCOM, PRDC, VSMM, ICAT, etc and a reviewer of international journals and magazines including ACM TOMCCAP, IEEE TPDS, IEEE Proceedings, IEEE JIE, IEEE TWC, Computer Networks, TOCSJ, JCN, IEEE wireless communication magazine, and IEEE Intelligence magazine. He has been an editor of JCN since 2004. His research interests include distributed systems, computer networks, mobile computing and pervasive computing. He is a member of KISS and IEEE, and a senior member of ACM.

Kilnam Chon is a Professor in the Graduate School of Media and Governance at Keio University, also serving as a Professor Emeritus in the Department of Computer Science at Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea. He received his M.S. in computer science and a Ph.D. in systems engineering from UCLA in 1967 and 1974, respectively. He has special interests in system architecture, including computer networking, distributed processing and information systems. He has worked as a principal investigator of a national project on workstation development, intelligent processing computer development and the information high-way.

Beakcheol Jang is a Ph.D. student in the Department of Computer Science at North Carolina State University, Raleigh, USA. His research interests include wireless and mobile systems, Internet applications, such as web and DNS, and traffic analysis with an emphasis on application performance. He received his B.S. in Computer Science from Yonsei University, Seoul, South Korea in 2001, and his M.S. in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea in 2002.

Taekyoung Kwon is an Associate Professor in the Multimedia and Mobile Communications Laboratory, School of Computer Science and Engineering, Seoul National University. He received his Ph.D., M.S., and B.S. degrees in computer engineering from Seoul National University in 2000, 1995, and 1993, respectively. He was a visiting student at IBM T. J. Watson Research Center in 1998 and a visiting scholar at the University of North Texas in 1999. His recent research areas include radio resource management, wireless technology convergence, mobility management, and wireless sensor networks.

Yanghee Choi received his B.S. in Electronics Engineering from Seoul National University, M.S. in Electrical Engineering from Korea Advanced Institute of Science and Technology, and Ph.D. of Engineering in Computer Science from Ecole Nationale Superieure des Telecommunications (ENST) in Paris, in 1975, 1977, and 1984 respectively. Before joining the School of Computer Engineering, Seoul National University in 1991, he was with the Electronics and Telecommunications Research Institute (ETRI) in 1977–1991, where he served as the Director of the Data Communication Section, and the Protocol Engineering Center. He was a research student at Center National d'Etude des Telecommunications (CNET), Issy-les-Moulineaux, in 1981–1984. He was also a Visiting Scientist to IBM T. J. Watson Research Center in 1988–1989. He is now leading the Multimedia and Mobile Communications Laboratory in Seoul National University. He is president of the Korean Institute of Information Scientists and Engineers. He is also chair of the Future Internet Forum. He is regular member of the National Academy of Engineering of Korea, and Korea and Korean Academy of Science and Technology. His research interest lies in the field of Future Internet.