The Medusa Proxy

A Tool For Exploring User-Perceived Web Performance

Mimika Koletsou and Geoffrey M. Voelker

Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Dr., MS 0114
La Jolla, CA 92093-0114 USA
{koletsou,voelker}@cs.ucsd.edu

Abstract

In this paper we describe the Medusa proxy, a tool for exploring user-perceived Web performance. The Medusa proxy is a non-caching forwarding proxy used in conjunction with a user's browser. The two key features that the Medusa proxy provides are (1) the ability to simultaneously mirror HTTP requests from the browser to different Web delivery systems and directly compare the results, and (2) the ability to transform requests, e.g., to transform Akamai URLs back into URLs to the customer's origin server.

We show how the Medusa proxy can be used to explore user-perceived Web performance using two experiments that determine (1) to what extent using large-scale caching systems like the NLANR cache hierarchy improve user-perceived latency, and (2) to what extent content distribution networks like Akamai improve latency compared with using their customer's origin server. We found that the NLANR cache hierarchy served 63% of HTTP requests in our workload at least as fast as the origin servers, resulting in a decrease of average latency of 15%. By transforming and mirroring Akamai URLs to both Akamai edge servers and their customer's origin servers, we find that Akamai edge servers are able to serve HTTP requests an average of 5.7 times as fast as their customers. However, because requests to Akamai edge servers are only 6% of our workload, we find that using Akamai reduces overall mean latency by only 2% for our workload.

Keywords: User-Perceived Web Performance, Web Proxy, Web Caching, Content Distribution Networks

1 Introduction

In this paper, we describe the Medusa proxy, a tool for exploring user-perceived Web performance. There has been extensive work evaluating the macroscopic impact of Web infrastructure for improving overall Web performance, such as caching systems (e.g., [2, 6, 7, 9, 16, 17, 19, 21, 25, 30, 32, 34]), prefetching systems (e.g., [4, 11, 15, 18, 27, 29]), and to a limited extent content distribution networks (e.g., [20, 22]). Surprisingly, however, there has been relatively little work characterizing the impact on Web performance *as perceived by the user* of various Web delivery systems. Although we have a relatively good understanding of how improvements in Web infrastructure impact overall latency, server load, and network utilization for large groups of users, we do not have a good understanding of how those improvements affect individual users at the desktop. In particular, there is no convenient mechanism by which users can determine the impact new developments in Web infrastructure, such as caching, content distribution, or prefetching, have on their Web browsing. For example, it is not uncommon for users to believe that Web caches impose noticeable overhead to browsing, or for content distribution network (CDN) companies such as Akamai [1] to make aggressive claims about improvements in download performance. Currently, though, there is no convenient way for users to validate those beliefs or claims.

Characterizing user-perceived Web performance is challenging because of the difficulty of generalizing conclusions about performance results across a wide range of user environments and network locations. Rather than try to generalize our conclusions across a diverse group of users, we have developed the Medusa proxy as a convenient tool with which individual users can explore their Web performance, and the impact of various Web delivery systems on that performance, from their desktop.

The Medusa proxy is a non-caching forwarding proxy used in conjunction with a user's browser. The Medusa proxy provides two key features for experimenting with user-perceived Web performance. The first feature is the ability to simultaneously mirror HTTP requests from the browser to different Web delivery systems and directly compare the results. For example, the Medusa proxy can mirror requests to a proxy cache and to the origin server to evaluate the effectiveness of the proxy cache on reducing latency. The second feature is the ability to transform HTTP requests. For example, users can install a filter that transforms an Akamai URL into the URL to the original object on the Akamai customer's origin server. By transforming and mirroring requests, the Medusa proxy can then simultaneously download the object from an Akamai edge server and the customer origin server and evaluate the effectiveness of having Akamai deliver that customer's content to the user.

In this paper we describe the features of the Medusa Web proxy and its uses for exploring Web performance, such as examining HTTP request headers, mirroring requests and monitoring performance, recording and replaying traces, transforming requests for privacy or experimentation, validating responses from experimental delivery systems, and accelerating performance; in this paper, we focus on user-perceived latency as the primary performance metric, although the Medusa proxy can be used to explore other performance metrics such as bandwidth, request rate, etc. We then use the Medusa proxy to answer the following performance and optimization questions:

- To what extent do large-scale cache systems like the NLANR cache hierarchy improve user-perceived latency?
- What is the potential of using the Medusa proxy as a Web accelerator, returning the fastest-of-n mirrored parallel requests?
- To what extent do content distribution networks like Akamai improve user-perceived latency over the performance of the

customer's origin server?

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the features and implementation of the Medusa proxy. Section 4 describes our experimental methodology and results for evaluating the NLANR, Akamai, and Squid Web delivery systems on user-perceived Web performance, and Section 5 concludes.

2 Related Work

The work most closely related to the Medusa proxy is the Simultaneous Proxy Evaluation (SPE) system proposed by Davison [13]. The SPE system has a similar goal in that it proposes to use mirroring to compare the performance of Web delivery systems such as proxy caches from the perspective of clients. However, the goals of comparing a Web delivery system with the origin servers, or transforming requests, is not proposed (although it would seem straightforward to incorporate). SPE was not implemented when originally proposed, but Davison and Krishnan are currently implementing it as the Rutgers Online Proxy Evaluator (ROPE) [14].

There have been a number of studies investigating Web user behavior and client performance using passive monitoring. Cunha et al. collected an early client trace at Boston University by instrumenting the Mosaic browser used by people in their department, and derived statistical characteristics of the client workload such as object size and popularity [12]. Catledge and Pitkow also collected a trace of Mosaic events from users at Georgia Tech, and used it to study methods of user interaction with the browser as well as user navigation patterns [8]. Kelly has recently collected a very large anonymized client trace from WebTV, and has used it to explore browser cache performance, reference locality, and document aliasing [23]. All of these studies have used passive measurement to explore user behavior and client performance, whereas we use the Medusa proxy to compare the performance of various Web delivery systems using active measurement.

Krishnamurthy and Wills [26] studied end-to-end Web performance from a collection of client sites distributed around the world to over 700 popular Web servers. They used active measurements to study the performance tradeoffs of parallel, persistent, and pipelined HTTP connections, as well as the interactions of those connection schemes with caches and multi-server content distribution. Their study compared the user-perceived performance of various HTTP protocol mechanisms, whereas we compare the user-perceived performance of various Web delivery systems. They also transformed Akamai URLs to compare data rates of Akamai edge servers with Akamai customer origin servers, and characterized the prevalence of Akamai-served content in popular Web servers. We use the same transformation technique to compare response times between Akamai edge servers and customer origin servers, and characterize the prevalence of Akamai-served content in a small user workload.

Rajamony and Elnozahy recently presented a framework for measuring the user-perceived response time of clients accessing a particular Web service [31]. With this framework, the Web service provider can characterize client response times to the service to determine whether service performance needs to be improved. This framework measures user-perceived performance across all clients from the perspective of a single service, whereas the Medusa proxy measures user-perceived performance from a single user to all accessed services through a potential set of Web delivery systems.

There has been comparatively little work experimentally evaluating the performance of content distribution networks. In one recent study, Johnson et al. compare the performance of two CDNs, Akamai and Digital Island, in terms of their ability to select the servers with minimum latency to the client [22]. However, they do not evaluate the impact of the CDNs on user-perceived latency, only on the relative performance of server selection within the CDN.

There are tools that report user Web download performance, such as Net.Medic [28], but they do not enable users to answer exploratory questions such as whether a proxy cache is degrading their performance, or how much added value CDNs are actually providing to them. Commercial systems such as the Lateral Line Network from Appliant.com [3] and various services from Keynote [24] report on the health and performance of customer Web sites from the perspective of many distributed users. However, as with the personal Web performance tools, these services do not compare alternative delivery systems and their impact on performance.

3 Medusa Proxy

The Medusa proxy is a non-caching forwarding proxy that supports the Proxy Request form of HTTP requests. Since it was designed to explore user-perceived latency, it is typically configured as a personal proxy for the Web browser and it executes alongside the browser on the same machine. It can also serve as a proxy for multiple clients, but we have yet to explore its use in that way. For portability, we implemented the Medusa proxy in Java to support widespread use. Finally, our support for HTTP 1.1 persistent and pipelined connections is ongoing, and the Medusa proxy currently only supports HTTP 1.0.

The Medusa proxy has a number of features for exploring user-perceived Web performance:

Mirroring. The Medusa proxy can be configured to mirror HTTP requests from multiple incoming sources to multiple destinations. It can accept incoming connections sequentially to limit interference, or multiple simultaneous connections in parallel for performance. It can also mirror requests sequentially so that they do not interfere with each other, or in parallel to further accelerate performance. In both cases of parallelism, Medusa uses threads to implement the parallelism; because we expect the Medusa proxy to be primarily used by a single client, the number of simultaneous connections and destinations, and hence the number of threads, is reasonably low.

In the experiments in this paper, we mirror requests from an individual source (the browser or a trace) to at most two different destinations at any one time, in sequential mode, to limit interference when recording measurements. To further limit interference, the Medusa proxy could be executed on a machine other than the one running the browser, or one with multiple processors.

Note that mirroring requests is harmless when the request semantics are idempotent. However, some requests have potentially serious side effects, such as POST requests for making purchases at e-commerce sites, that would cause undesirable behavior if repeated more than once. The three POST requests in our trace log information to advertising servers, and so we did not filter them out. In general, though, we can provide a flag to conditionally

Period	Requests	Users
March 6	1548	2
March 31–June 1	540	1
June 4–June 8	2655	1

Table 1: Trace periods (all days in 2001).

filter POST requests.

Transformation. The Medusa proxy can install filters to transform HTTP requests as they are forwarded. In this paper we use this feature to transform Akamaized URLs to the original URLs that refer to customer origin servers. As an example of another use of transformation, a filter can be installed that strips or anonymizes user-specific HTTP header information, such as embedded user or login names.

Performance measurement. As it mirrors and transforms requests, the Medusa proxy can record performance information, such as request count and frequency, latency, and bandwidth consumed, in a performance log. In this paper, we focus on request count and latency.

Tracing and replay. As it receives requests, the Medusa proxy can record them in a trace for subsequent replay in non-interactive mirroring and transformation experiments. We replay traces to experiment with Akamai in Sections 4.3.

Validation. When building new infrastructure, such as a new proxy cache implementation, it is useful to be able to validate that the cache is responding with correct data. In conjunction with the mirroring feature, the Medusa proxy can be used to validate new implementations of delivery systems by directly comparing the response from the experimental cache with the response from the origin server. Comparing responses is not only useful for validating content data, but also for comparing metadata as well. Comparing responses from multiple delivery systems can also be used to explore the effects of metadata differences, such as the "age penalty" from caches [10].

In the next section we use many of these features of the Medusa proxy to explore various aspects of user-perceived Web performance.

4 Experimental Results

In this section, we use the Medusa proxy to explore the impact of using a large cache system (NLANR cache hierarchy), a content distribution network (Akamai), and a proxy cache (Squid) on user-perceived download latency.

4.1 Methodology

To generate a workload for our experiments, we used the Medusa proxy for 8 days of everyday Web browsing as shown in Table 1. The browsing was relatively bursty, with most activity in the morning, around lunch, and in the evening. The total workload consists of 4743 HTTP requests, many of which are to generally popular Web sites like nytimes.com, espn.go.com, java.sun.com, etc., and some of which are to more domain-specific sites like nsf.gov. All requests are to Web servers outside of UCSD. We chose this workload because it is exactly the workload that we would like to understand and improve. Since it is impossible to generalize these results across all users, people can use the Medusa proxy to determine the extent to which the various Web delivery systems impact their own user-perceived Web performance.

In all experiments, we executed the Medusa proxy on a 700 MHz Pentium III workstation with 256 MB of memory running Windows NT 4.0. We ran the Medusa proxy using Sun's Java 2 Runtime Environment (v1.3.0-C) with the same version of HotSpot. Because Medusa does not cache data, its memory requirements are low; it used 11 MB of memory during our experiments.

We used Internet Explorer 5 (IE) as the browser to generate our workloads. IE executed on the same machine as, and directly communicated with, the Medusa proxy. For the NLANR experiments in Section 4.2, we collected performance data and recorded the HTTP requests as we interactively browsed throughout the day to generate our workload. For the experiment in Sections 4.3, we replayed the collected traces directly using the Medusa proxy. The Medusa proxy can replay traces with variable delays between requests; we used 500 ms for the experiments in this paper as a tradeoff between reducing server load and being able to complete replays in a reasonable amount of time.

The Medusa proxy calculates download times from just before initiating a connection until it closes the connection. These times include the cost of the DNS lookup and the TCP connection setup and close. In our Java environment the Java timer has a granularity of 10 ms, limiting the accuracy of very fast downloads of small objects. Although we would prefer a much more accurate timer, we use the Java timer for the sake of portability.

4.2 NLANR Cache Hierarchy

In this section, we use the mirroring feature of the Medusa proxy to explore the extent to which a large-scale caching system, the NLANR cache hierarchy, improves user-perceived Web performance. Our goals are to characterize the effect of using the NLANR cache hierarchy on user-perceived performance, and to provide a mechanism by which users can determine whether using a cache system like the NLANR cache hierarchy improves their Web browsing performance. Because we use mirroring to explore cache hierarchy performance, we also determine the potential of using mirroring as a technique for implementing a (network-unfriendly) Web accelerator.

The NLANR cache system consists of a hierarchy of international Squid Web caches [33]. At the root of the hierarchy is a mesh of caches primarily located at the U.S. supercomputer centers. These caches are peers and query each other for cached content, and serve as parents to other Squid caches distributed throughout the world. The cache hierarchy reduces network utilization and server load, as well as user-perceived download latency. The cache hierarchy can improve performance when users request objects that have been previously requested by other users and cached in the hierarchy. However, the cache hierarchy can degrade performance, particularly for requests to objects that miss in the cache system, because of the overhead of additional hops between the user and the origin server and the latency for querying peers [32].

All Requests (4743)	Mean	Median
Origin Server	253 ms	190 ms
NLANR Hierarchy	220 ms	111 ms
Server/NLANR	1.15	1.71
Accelerator	159 ms	100 ms
NLANR/Accelerator	1.38	1.11
Server/Accelerator	1.59	1.90

Table 2: NLANR results summary.

4.2.1 User-perceived latency

To explore the extent to which the NLANR cache hierarchy impacts user-perceived download latency, we used the Medusa proxy to mirror all HTTP requests from our Web browsers to both the NLANR hierarchy and the origin servers. The Medusa proxy first sends every request to the origin server, waits for the entire response to be downloaded, then sequentially mirrors the request to the `lj.us.ircache.net` cache, and records the download latency from each destination. The response from the server is discarded, and the response from the cache is delivered to the browser. The mirroring is done online during interactive browsing so that requests to both destinations experience similar network behavior. The requests are sent sequentially to prevent contention within the Medusa proxy (the Medusa proxy can also mirror requests in parallel to accelerate browsing, as discussed below, but we do not use the parallel feature in any of the experiments in this paper).

Our results are slightly biased towards the NLANR cache hierarchy for a couple of reasons. First, the initial request to the server presumably preloads the object in the server cache, which could reduce the miss penalty for the cache hierarchy. An alternative approach would have been to randomly choose to send the initial request to either the cache or the origin server, but we preferred the known, deterministic algorithm of sending to the cache first. Second, we have excellent connectivity to the `lj.us` cache, which has a 1 ms ping round-trip time from the machine running the Medusa proxy. Although we do not expect this performance to be typical, this is what we experience and hence what we explore.

The top half of Table 2 summarizes our results comparing the performance of NLANR and the origin servers. The "Origin Server" row shows the mean and median download latencies for downloading objects in our workload directly from the origin servers as determined by the URL in the request. The "NLANR Hierarchy" row shows the corresponding latencies for downloading the objects through the cache hierarchy; the "Server/NLANR" row shows the speedup NLANR provides relative to the origin server; and we discuss the remaining rows in Section 4.2.2 below. On average, NLANR noticeably improves download latency: the mean latency for NLANR is 15% faster than using the origin server, and the median latency for NLANR is 71% faster.

To display the effect of using the NLANR cache hierarchy on all requests in our workload, Figure 1 shows the cumulative distributions of the relative performance of the NLANR cache hierarchy and the origin server across all requests. The x-axis shows relative speedup (on a log scale), and the y-axis shows the fraction of requests in the workload. Since the cache hierarchy is not necessarily faster, we separate the requests into two curves. The solid "Server Faster" curve shows the speedup of those requests where downloading from the origin server was faster, and the dashed "Cache Faster" curve shows the speedup of those requests where download from the cache hierarchy was faster. For example, looking at the solid "Server Faster" curve, 31% of all requests were downloaded faster from the server and were downloaded with a speedup of 3 or less than from the cache hierarchy. Similarly, looking at the dashed "Cache Faster" curve, 52% of all requests were downloaded faster from the cache hierarchy and were downloaded with a speedup of 4 or less than from the server.

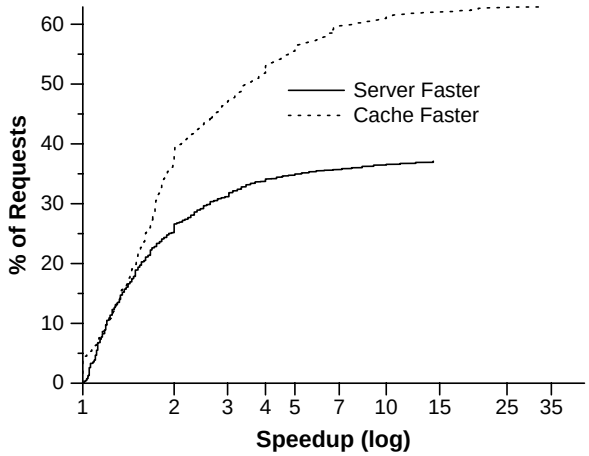

Figure 1: Distribution of the relative download latency of the NLANR cache hierarchy and origin servers. Note the use of a log scale for the x-axis.

Although the NLANR cache hierarchy improves mean latency by 15%, Figure 1 shows that for 63% of the requests the cache hierarchy returned responses faster or as fast as the origin server; for 37% of the requests, the origin server was faster; and for 4% of the requests they had equivalent download latencies (where the dashed "Cache Faster" curve crosses the y-axis). Both the server and the cache hierarchy had large speedups relative to the other. These large speedups were for requests to small files (e.g., small images) where one system responded quickly (e.g., 10–20 ms) but the other experienced network delay or server load that caused significant relative latency (e.g., 100–500 ms).

4.2.2 The Medusa proxy as a Web accelerator

Since the origin server is frequently faster than the cache hierarchy, it raises an interesting question of whether mirroring could be used as a technique to accelerate user-perceived Web performance. With the Medusa proxy, we can potentially mirror requests in parallel and return the fastest response to the browser, whether from the cache or origin server. Since mirroring consumes roughly twice the resources of normal Web browsing, it is a network-unfriendly technique and we do not condone its use in this way. However, we were intrigued to see what the potential was for such a technique.

To estimate the potential benefits of a mirroring accelerator, we selected the fastest response latency for each request in our workload and calculated the mean and median latencies across the requests. The results are shown in the bottom half of table 2. The "Accelerator" row shows the mean and median latencies of using the Medusa proxy as an accelerator; the

"NLANR/Accelerator" row shows the relative performance of the accelerator compared with just using the NLANR cache hierarchy; and the "Server/Accelerator" row shows the relative performance of the accelerator compared with just using the origin servers. The results indicate that a mirroring accelerator has potential for improving performance: relative to the cache hierarchy, it improves mean latency by 38% and median latency by 11%. Furthermore, based upon median latency, at least half of the HTTP requests would be downloaded in under 100 ms.

4.2.3 Discussion

From these results, we conclude that using the NLANR cache hierarchy measurably improves the download latency for our workload. However, it is not clear what the user-perceived impact of this improvement is. If a page has only one object, then an average absolute improvement of 33 ms (NLANR) or 61 ms (accelerator) is below the human threshold for noticing the difference [5]. However, if a page has many embedded objects, then the cache system has the potential to make a large absolute improvement. The exact improvement, however, depends upon the number of objects and the overlap of multiple parallel connections, and remains an open question.

4.3 Akamai Content Distribution Network

In this section, we use transformation to explore the user-perceived performance of Akamai, a large content distribution network [1]. Content distribution networks have the potential to significantly improve user-perceived Web performance because they bring content closer to users and are able to balance load across a large number of distributed servers. In contrast, most content providers have a single server site from which they deliver content, and this site is potentially far from many users and has to scale to handle all user load. In our experiments, we directly compare the performance of accessing Akamaized objects from Akamai edge servers with their customer's origin server, and estimate the overall impact of having content served from Akamai on our workload.

To perform our experiment, we transformed all HTTP requests to the Akamai network in our trace into requests to the Akamai customer's origin server (e.g., Krishnamurthy and Wills [26] and Cohen and Kaplan [10] employ the same technique). Akamai uses a well-formed scheme to "Akamaize" customer content by prefixing an Akamai domain name and additional information to the original URL of the object on the customer's origin server. For example, a typical Akamai URL might be structured as follows:

```
http://a8.g.akamaitech.net/
  f/8/621/12h/www.customer.com/...
```

We can recover the original URL to the customer's origin server by removing the Akamai prefix:

```
http://www.customer.com/...
```

With both the Akamai URL and the customer server URL, we can then use the Medusa proxy to mirror requests to both the Akamai edge servers as well the customer origin servers and compare the resulting latencies.

As our workload, we used the traces recorded during the NLANR experiment in Section 4.2 and replayed them through the Medusa proxy, transforming Akamai URLs, mirroring requests, and recording the download latencies. We used the machine described in Section 4.1 to execute the Medusa proxy and replay the traces. We replayed our traces starting at noon PDT on Monday, June 11, 2001 to capture network congestion and server load, conditions which emphasize the positioning and distribution of Akamai edge servers.

To reduce variation, we replayed the traces five times and used the median download latency across the five runs for each object as its representative download time. Replaying the trace multiple times biases both the Akamai edge servers and the customer origin servers towards better performance since the replays warm their caches. When looking at the overall impact of Akamai, this makes our results conservative by giving the benefit of doubt to Akamai. Replaying also reduces the overhead of DNS lookups for Akamai; however, Akamai URLs in the trace only refer to 19 unique edge servers, so Akamai would significantly benefit from DNS caching even without multiple replays.

Akamai Requests (278)	**Mean**	**Median**
No Akamai (Origin)	141 ms	120 ms
Akamai	24.7 ms	20.0 ms
No Akamai/Akamai	5.7	6.0
All Requests (4743)	**Mean**	**Median**
No Akamai (Origin)	259 ms	190 ms
Akamai	253 ms	190 ms
No Akamai/Akamai	1.02	1.00

Table 3: Akamai results summary.

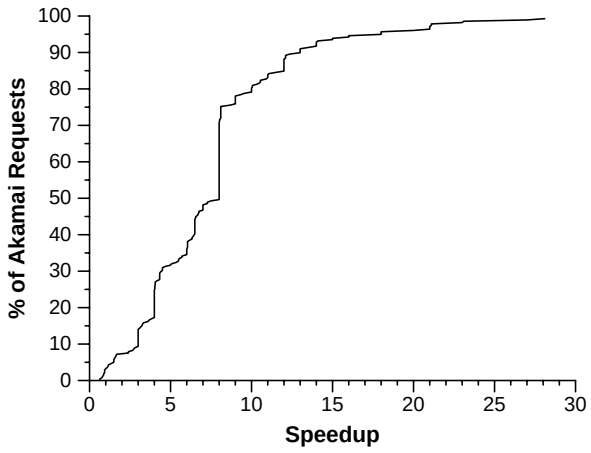

Figure 2: Distribution of the relative download latency of Akamai and customer origin servers.

4.3.1 Direct impact

The top half of Table 3 summarizes our results for directly comparing the performance of downloading objects from the Akamai edge servers and the customer origin servers. These results take into account only those requests that were originally addressed to the Akamai network (in this case, 278 requests). The "No Akamai" row shows the mean and median download latencies for objects downloaded from the customer origin server; the "Aka-

mai" row shows the corresponding download latencies for downloading those objects from the Akamai network; the "No Akamai/Akamai" row shows the speedup Akamai provides relative to the customer server; and we discuss the remaining rows in Section 4.3.2 below.

As these results show, both the mean and median latencies for Akamai are quite low, on the order of 20–25 ms (the sample Akamai edge server we resolved has a ping round-trip time of 7 ms). Furthermore, Akamai delivers impressive speedups of 5.7 for the mean download latency and 6 for the median latency.

To display the performance across all Akamai requests, Figure 2 shows the cumulative distribution of the relative performance of Akamai to the customer origin servers for all Akamai requests. The x-axis shows the speedup for sending the request to the Akamai network relative to the customer origin server, and the y-axis shows the fraction of Akamai requests in the workload. Each point on the curve corresponds to one request, and the requests are sorted in increasing order of speedup. As we can see from the graph, Akamai always delivers content faster than the customer origin server and, for nearly 20% of the requests, Akamai is at least 10 times as fast. The very large speedups are from requests to small images, where network conditions and customer origin server load add significant relative latency.

In terms of the content that Akamai delivers, the user-perceived performance of Akamai is excellent for our workload. However, this performance improvement is only for the portion of our workload that is served by Akamai. Since users are more interested in the performance impact on their overall workload, we explore this question next.

4.3.2 Overall impact

To estimate the overall performance impact of Akamai on our workload, we used the workload results from our Web browsing in Section 4.2 and replaced the fast download latencies of Akamai requests with the much larger latencies of downloading the same content from the customer origin servers obtained above. This replacement corresponds to a workload in which no content is served from Akamai and all content is served from origin servers. With the original workload that includes requests to Akamai and the modified workload that only has requests to origin servers, we can estimate the overall impact Akamai has on performance.

Our results are shown in the bottom half of Table 3. These results take into account all requests in the workload. The "No Akamai" row corresponds to the workload where Akamai request latencies are replaced with the corresponding latencies for downloading from the customer origin server. The "Akamai" row corresponds to the original workload that includes requests to the Akamai network. And the "No Akamai/Akamai" row shows the relative performance of the two workloads. From the table, we see that the Akamai workload improves average download latency by only 2.4% over the workload that does not use Akamai. To put this result in perspective, recall that using the NLANR cache hierarchy improves average overall download latency for our workload by 15%. In other words, for our workload Akamai improves average user-perceived latency only slightly, and we would have experienced similar average performance if all of our content was from origin servers. Note that this result assumes that the absence of Akamai would not increase network congestion and customer origin server load considerably, but we cannot experimentally evaluate the effect of this assumption.

4.3.3 Discussion

The difference between the direct and overall impact of using Akamai is due to the fact that Akamai only serves 6% of all requests from our workload: 278 Akamai requests out of 4743 total requests. Although Akamai has excellent relative performance for the content it serves, it only captures a small fraction of our overall workload. These results are very much workload dependent, but, for rough comparison, Krishnamurthy and Wills found that 1.5% of base URLs to the home pages of 711 popular Web servers had embedded objects served by Akamai [26].

There are a number of conclusions one could make from these results, depending upon one's perspective. From Akamai's perspective, these results demonstrate that Akamai serves its content extremely well compared to their customer's servers. From an Akamai customer's perspective, these results show that Akamai improves the download latency for the content on Akamai's servers while offloading requests from the customer's servers. However, Web pages are compound documents. Since Akamai primarily serves images, the remaining content that comes from the customer's origin servers remain the bottleneck. Again, according to Krishnamurthy and Wills, 31–47% of objects and 56–67% of bytes on the home pages of a selection of popular Web servers that use Akamai still come from the customer's origin server [26] (Table 7). Finally, from the user's perspective, Akamai has impact on those pages that include embedded objects served by Akamai, but only a minimal impact on the overall workload.

The overall conclusion that we make is that a content delivery network like Akamai is a very effective system for distributing content close to the user, and has great potential for making a significant impact on user-perceived latency. However, until CDNs like Akamai serve a larger fraction of the overall content requested in user Web workloads, the full potential that CDNs have to offer for improving user-perceived latency will remain unrealized.

5 Conclusion

In this paper, we describe the Medusa proxy, a tool for exploring user-perceived Web performance. We describe its use to determine (1) to what extent using large-scale caching systems like the NLANR cache hierarchy improve user-perceived latency, and (2) to what extent content distribution networks like Akamai improve latency compared with using their customer's origin servers.

For our personal workloads, we found that:

- When using the NLANR caches 63% of HTTP requests are served at least as fast from the NLANR caches as from the origin server, decreasing mean latency by 15%. However, in absolute time, this improvement is only 33 ms on average.
- Using the Medusa proxy as a Web accelerator, mirroring requests to both the NLANR caching system and the origin server in parallel and returning the fastest response, can improve mean latency by 38% compared with using the NLANR caching system alone.
- Akamai edge servers are able to serve HTTP requests an average of 5.7 times as fast as their customer's origin servers. However, because requests to Akamai edge servers are only 6% of our workload, we find that using Akamai reduces overall mean latency by only 2%.

Of course, these results depend upon the perspective of our workloads and our computing and network environment, and the results are surely different for other users. However, with the Medusa proxy, users can answer these questions relative to their own environments to determine the impact of Web delivery systems on their performance.

Acknowledgements

We would like to thank Terence Kelly and the anonymous reviewers for their helpful comments and suggestions. We would also like to thank Duane Wessels for giving us direct access to the `lj.us.ircache.net` NLANR cache. Equipment used in this research was supported in part by the UCSD Active Web Project, NSF Research Infrastructure Grant Number 9802219.

References

[1] Akamai. Akamai content delivery network. http://www.akamai.com.

[2] V. Almeida, A. Bestavros, M. Crovella, and A. deOliveira. Characterizing reference locality in the WWW. Technical Report 96-011, Boston University, June 1996.

[3] Appliant.com. Lateral line network. http://www.appliant.com/services/lln.htm.

[4] A. Bestavros. Using speculation to reduce server load and service time in the WWW. In *Proc. of the 4th ACM Intl. Conf. on Information and Knowledge Mgmt.*, November 1995.

[5] N. Bhatti, A. Bouch, and A. Kuchinsky. Integrating User-Perceived Quality into Web Server Design. In *Proc. Ninth Int. World Wide Web Conf.*, May 2000.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proc. of IEEE INFOCOM '99*, pages 126–134, March 1999.

[7] R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich. Web proxy caching: The devil is in the details. In *Workshop on Internet Server Performance*, pages 111–118, June 1998.

[8] L. D. Catledge and J. E. Pitkow. Characterizing browser strategies in the World-Wide Web. *Computer Networks and ISDN Systems*, 26(6):1065–1073, April 1995.

[9] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical Internet object cache. In *Proc. of the 1996 USENIX Technical Conf.*, pages 153–163, San Diego, CA, January 1996.

[10] E. Cohen and H. Kaplan. The age penalty and its effect on cache performance. In *Proc. of the 3rd USENIX Symp. on Internet Technologies and Systems*, pages 73–84, March 2001.

[11] M. Crovella and P. Barford. The network effects of prefetching. In *Proc. of Infocom '98*, April 1998.

[12] C. R. Cunha, A. Bestavros, and M. E. Crovella. Characteristics of WWW client-based traces. Technical Report BU-CS-95-010, Boston University, July 1995.

[13] B. D. Davison. Simultaneous Proxy Evaluation. In *Proc. of the 4th Int. Web Caching Workshop*, pages 67–77, March 1999.

[14] B. D. Davison and C. Krishnan. ROPE: The Rutgers Online Proxy Evaluator. In preparation, 2001.

[15] D. Duchamp. Prefetching hyperlinks. In *Proc. of the 2nd USENIX Symp. on Internet Technologies and Systems*, October 1999.

[16] B. Duska, D. Marwood, and M. J. Feeley. The measured access characteristics of World Wide Web client proxy caches. In *Proc. of the 1st USENIX Symp. on Internet Technologies and Systems*, pages 23–36, Dec. 1997.

[17] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary Cache: A scalable wide-area web cache sharing protocol. In *Proc. of ACM SIGCOMM '98*, Vancouver, BC, August 1998.

[18] L. Fan, P. Cao, and Q. Jacobson. Web prefetching between low-bandwidth clients and proxies: Potential and performance. In *Proc. of ACM SIGMETRICS '99*, May 1999.

[19] A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich. Performance of web proxy caching in heterogeneous bandwidth environments. In *Proc. of IEEE INFOCOM '99*, March 1999.

[20] S. Gadde, J. Chase, and M. Rabinovich. Web caching and content distribution: A view from the interior. In *Proc. of the 5th Int. Web Caching and Content Delivery Workshop*, May 2000.

[21] S. D. Gribble and E. A. Brewer. System design issues for Internet middleware services: Deductions from a large client trace. In *Proc. of the 1st USENIX Symp. on Internet Technologies and Systems*, pages 207–218, Montery, CA, December 1997.

[22] K. L. Johnson, J. F. Carr, M. S. Day, and M. F. Kaashoek. The measured performance of content distribution networks. In *Proc. of the 5th Int. Web Caching and Content Delivery Workshop*, May 2000.

[23] T. Kelly. Thin-Client Web Access Patterns: Measurements from a Cache-Busting Proxy. In *Proc. of the 6th Int. Web Caching and Content Delivery Workshop*, June 2001.

[24] Keynote. Keynote web performance services. http://www.keynote.com.

[25] B. Krishnamurthy and C. E. Wills. Study of piggyback cache validation for proxy caches in the World Wide Web. In *Proc. of the 1st USENIX Symp. on Internet Technologies and Systems*, Dec. 1997.

[26] B. Krishnamurthy and C. E. Wills. Analyzing Factors That Influence End-to-End Web Performance. In *Proc. Ninth Int. World Wide Web Conf.*, Amsterdam, May 2000.

[27] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploring the bounds of Web latency reduction from caching and prefetching. In *Proc. of the 1st USENIX Symp. on Internet Technologies and Systems*, pages 13–22, Dec. 1997.

[28] Lucent-NPS. Net.medic. http://www.ins.com/software/medic.

[29] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve World Wide Web latency. *ACM Computer Communications Review*, 26(3), July 1996.

[30] M. Rabinovich, J. Chase, and S. Gadde. Not all hits are created equal: Cooperative proxy caching over a wide area network. In *Proc. of the 3rd Int. WWW Caching Workshop*, June 1998.

[31] R. Rajamony and M. Elnozahy. Measuring Client-Perceived Reponse Times on the WWW. In *Proc. of the 3rd USENIX Symp. on Internet Technologies and Systems*, pages 185–196, March 2001.

[32] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Design considerations for distributed caching on the Internet. In *Proc. of ICDCS '99*, May 1999.

[33] D. Wessels and K. Claffy. ICP and the Squid Web Cache. *IEEE Journal on Selected Areas in Communication*, 16(3):345–357, 1998.

[34] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. On the scale and performance of cooperative Web proxy caching. In *Proc. of SOSP '99*, pages 16–31, December 1999.