Whole Page Performance

Leeann Bent 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
{lbent,voelker}@cs.ucsd.edu

Abstract

This paper explores the user-perceived Web performance of downloading entire pages, and how various common Web enhancements impact overall page performance. We use Medusa, a non-caching forwarding proxy, to collect user traces and replay them under various configurations of HTTP request optimizations. These optimizations include parallel and persistent connections, DNS caching, and the use of CDNs. We then use Medusa to characterize whole-page performance and measure the impact of request optimizations on downloading entire Web pages.

We find that the most effective optimization is parallel connections. Other optimizations provide incremental benefits due to limited opportunity for use or due to limited benefit of the optimization itself. When there is more opportunity for use, we find that optimizations in the former case provide substantial benefit.

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

1 Introduction

This paper explores the user-perceived Web performance of downloading entire pages, and how various common Web enhancements impact overall page performance. Extensive previous work has studied how various techniques, such as caching (e.g., [6,8,13,17,21]), prefetching (e.g., [2,4,5,15], content distribution networks (CDNs) (e.g., [7, 9, 11, 12]), and DNS resolution (e.g., [10, 18, 19]), impact the performance of downloading individual objects. But when browsing the Web, users are much more concerned with the performance of entire pages. Although whole-page performance is determined by the performance of its components, optimizations like parallel and persistent connections make it difficult to map directly from individual object performance to whole-page performance. As a result, even though whole-page performance is what users ultimately are most interested in, it has received scarce attention.

The most extensive work on this topic is by Krishnamurthy and Wills [14]. They studied the impact of parallel, persistent, and pipelined connections on user-perceived Web performance for embedded objects in top-level pages from a set of popular servers. In effect, our study is a follow-on to theirs from a somewhat different perspective. First, we employ real user workloads. We look at whole-page performance for all pages accessed by users, not just the top-level pages on popular servers. Second, in addition to embedded objects, we include the time of downloading the base page as a factor in whole-page performance. Finally, in addition to the connection optimizations, we also single out the contribution of CDNs and DNS to whole-page performance.

Exploring whole-page performance opens up a number of interesting questions about user-perceived Web performance. How does whole-page performance compare to individual object performance? How do various downloading optimizations improve whole-page performance? Content distribution networks (CDNs) can improve object download performance [9, 11, 12], but how do CDNs impact whole-page latency given that typically only a subset of objects comprising a page are fetched from a CDN? Since DNS resolution can increase download latency [3, 19], particularly when using CDNs [12], how does DNS caching amortize resolution costs across all of the objects downloaded in a page? These various questions translate into two high level goals: (1) we want to examine the extent to which the above optimizations are used and (2) we want to study the effect of these different optimizations on downloading whole pages of objects. This paper addresses these high level goals.

The primary challenge in studying user-perceived whole-page performance is the difficulty in measuring the overall effect of Web download optimizations at the page level. With objects being download over parallel connections, for example, pinpointing where exactly where slowdowns occur during user browsing can be a difficult task.

Our approach for exploring whole-page performance uses the Medusa proxy [11]. The Medusa proxy is a non-caching forwarding proxy deployed in conjunction with a user's browser. Its key features are that it can transform requests and mirror them to multiple destinations. For this study, we extend the Medusa proxy to support HTTP connection optimizations, including parallel and persistent connections, as well as measuring DNS and CDN effects. We then use the Medusa proxy on user traces to characterize whole-page performance and measure the impact of request optimizations on downloading entire pages.

In general, our results show what one would expect when comparing the relative contributions of the performance optimizations. The easiest and relatively most effective connection optimization is simply to use parallel connections. Meanwhile, other optimizations provide incremental benefit. In some cases this is due to a limited opportunity for use, while in other cases it is due to the limited benefit of the optimization itself. In the former case, improvements are shown to be substantial when there is more opportunity for use.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the Medusa proxy and the extensions that support the HTTP request optimizations we use to study whole-page performance. Section 4 in addition to our methodology and section 5 presents our experimental results. Finally, Section 6 concludes.

2 Related Work

There have been numerous studies on the impact of various optimizations on the latency of downloading individual objects, including caching, prefetching, CDNs, and DNS resolution. These studies tend to examine particular optimizations in isolation, whereas our goal is to examine their relative effects on whole-page performance.

Surprisingly, there have been relatively few studies on whole-page performance. Liston and Zegura [16] describe a proxy used in a similar way as Medusa to measure client-side Web performance. Their initial goal is to study the contribution of DNS resolution to the latency of downloading web pages, although they only describe the design and implementation of the proxy and do not include any experimental results.

As discussed in the introduction, Krishnamurthy and Wills have done the most extensive published research on this topic. This work extends theirs by (1) looking at real user workloads, (2) looking at all pages accessed by users, (3) looking at base page download times in addition to embedded object times, and (4) examining CDN and DNS contributions.

In [12], the authors look at a variety of different performance aspects of CDNs. They found that, while CDNs improve performance, they can also incur high DNS costs. They also quantified DNS effects on persistent connections and pipelining. While our study does consider many of these same factors, we look at real web pages encountered in actual web browsing. In addition we look at optimizations on a whole page basis, rather than per object.

In [20], Wills et al. use bundling to deliver embedded web objects from a server as an alternative mechanism to pipelining and parallel connections when fetching multiple objects. They find that fetching compressed bundles offers faster downloads than pipelining or parallel retrieval. We look at the use of parallel options, as well as other factors that influence web download performance such as CDNs and DNS usage. However, we do not consider bundling in the optimizations we explore.

3 The Medusa Proxy

The Medusa proxy is a non-caching forwarding proxy [11]. It was designed to explore user-perceived Web performance and is typically configured as a personal proxy for the Web browser and executes alongside the browser on the same machine. Medusa has a number of features, including:

Tracing and replay. As it receives requests, the Medusa proxy can record them in a trace for subsequent replay in non-interactive experiments.

Performance measurement. As it handles interactive or replayed requests, the Medusa proxy tracks and records performance information, such as request latency, DNS overhead, and connections used.

Transformation. The Medusa proxy can install filters to transform HTTP requests. We use this feature to transform Akamaized URLs (ARLs) to the original URLs that refer to customer origin servers.

Optimization options. Different download optimizations can be toggled in the Medusa proxy during object requests, such as using parallel and persistent connections.

The initial version of the Medusa proxy supported basic HTTP 1.0 request functionality. For the experiments in this paper, we extended Medusa to support a number of new features, including page delimitation, parallel connections, and persistent connections.

To capture the effects of downloading entire pages, Medusa records page delimiters in its traces and supports the concept of page downloads during trace replay. In this mode, Medusa requests all objects in a page according to its configured optimization mode, and then waits for all responses before requesting the objects in the subsequent page.

To support parallel connections, Medusa records the number of different connections used by the browser during trace collection, as well as which requests are sent over which connection. With this information, Medusa is able to faithfully recreate the connection behavior used during browsing. Requests are fetched in a sequential fashion on each connection in the order in which they were originally requested, with all connections beginning requests at the same time. As a result, parallel download time is optimistic but also consistent and repeatable.

Persistent connections are implemented according to both the HTTP 1.0 convention and the HTTP 1.1 specification. Medusa has several persistent connection modes: it can attempt to always keep the connection open, it can attempt to mirror the original connection state found in the trace, or it can always close the connection. Should a persistent connection fail, we re-establish the connection and continue.

The Medusa proxy is highly configurable. Any of the connection options can be enabled in any configuration. In addition, Medusa now records the DNS resolution time for each object. Medusa can either replay traces using recorded DNS resolutions or re-resolve DNS resolutions. In either case, DNS resolution time (previously recorded or current lookup) can be toggled to count towards object download cost (or not).

As with the original version, Medusa can still transform requests, such as converting Akamai requests to origin server requests, online or during replay. This is limited to Akamai Freeflow ARLs, however, and does not consider sites completely hosted on CDNs. We use these options in various configurations to explore whole-page download performance as described in the following section.

4 Methodology

For our experiments, we use the Medusa proxy to record everyday browsing from six different users over the course of four days. We then replay these traces, again using the Medusa proxy, while toggling different performance optimizations and capturing individual object download times. Finally, we use the individual object download times measured during replay to compute download cost for whole pages.

In this section, we describe how the Medusa proxy is used to compute download cost, describe our trace characteristics, and outline the performance optimizations we use to explore whole-page performance.

4.1 Computing Page Download Time

4.1.1 Measurements

In this study, we define user-perceived latency for whole pages to be the total wall clock time required to download the base page and all of its embedded objects. We refer to this time as the *page download time*. We calculate *page download time* using *object download time*, as defined below. Of course, the browser can display useful information before all of the page's objects are downloaded. In practice, though, it is difficult to determine at what point the user considers a page downloaded, and we consider the total time to download the page a reasonable conservative and repeatable metric.

The Medusa proxy calculates *object download time* from just after the DNS lookup until it closes the connection; this cost includes the TCP connection setup and close. When persistent connections are enabled, cost is calculated from just after DNS lookup until full object return, and does not include connection close. Where DNS overhead is included in page download time, we have added the original recorded DNS overheads to object download time after replay. For our timings in our Java implementation of Medusa, we used a custom native timer implementation with a 3ns granularity to gather accurate results. Although the native call to the timer incurs a 51 microsecond overhead, this overhead is still negligible relative to the time durations we measure.

Calculating *page download time* depends on the download optimizations used. For experiments that do not use parallel connections, we sum the total object download time for all objects on a page (as dictated by the page delimitation algorithm) to get total page download time. For experiments with parallel connections enabled, we sum up the total object download time for each sequence of objects downloaded serially, then take the maximum sequence time as page download time. Examples of calculation of *page download time*, in both cases, are given in the following section.

To account for variation in individual download times, we download each page in the trace five times, one right after another. We then take the median page value as the representative page download time for that page. Across sets of pages, we report both average page download time and median page download time.

4.1.2 Example

We use an example to illustrate the calculation of *page download time*. Suppose we replay a trace to

	Download Time	DNS Time
Obj1	150	5
Obj2	200	40
Obj3	100	4
Obj4	250	90

Table 1: Example object download and DNS times for a page with four objects. All times in milliseconds.

Method	Download Seq.	Seq. Time	With DNS
Serial	Obj1,Obj2,Obj3,Obj4	700	839
	Total page download time:	700	839
Parallel	Obj1, Obj3	250	259
	Obj2, Obj4	450	580
	Total page download time:	450	580

Table 2: Example page download times when using (1) serial connections, and (2) parallel connections using two connections. All times are in milliseconds. ID refers to sequence ID identifying a set of objects that must be downloaded serially.

download a 4-object page with the *object download times* and DNS resolutions times as shown in Table 1. Note that, even in trace replay, we would use the DNS times from the original user trace. For the purpose of this example, persistent connections are turned off. In actual replay situations, times with and without persistent connections enabled would, of course, be different.

Table 2 shows how we would determine page download times when downloading the page using serial connections and when using parallel connections. With serial connections, Medusa opens and closes new connections to download the objects one after the other in sequence. In this case, the total page download time is simply the sum of the individual times: 700ms without the DNS times, and 839ms with them.

With two parallel connections, Medusa downloads the objects in parallel. It first opens two parallel connections to download the first two objects. When the first object finishes, it opens another connection to download the third object; when the second object finishes, it opens another connection to download the fourth. Since these object download sequences happen in parallel due to the use of parallel connections, the total page download is the maximum total download time of either sequence of object downloads: 450ms without the DNS times, and 580ms with them.

Note that in both methods Medusa opens and closes a new connection for each object; if we had used persistent connections in this example, then Medusa would have reused connections if the objects came from the same server.

4.2 Traces

To generate a workload for our experiments, we collected traces of everyday Web browsing from six different users in our research lab from Saturday, April 27 through Tuesday, April 30, 2002.

Table 3 summarizes our user traces. We originally recorded 22,338 objects in 1,455 pages. However, upon replaying the traces, not all of the objects in the traces could be successfully downloaded again due to connections errors and 4XX/5XX responses. We therefore removed all pages with any objects that failed to download from the original trace. The result was a workload of successfully downloaded pages consisting of 13,747 HTTP requests comprising 920 pages, for an average of 15 requests per page.

To explore the individual and combined impact of different download optimizations, we replayed these traces using Medusa while enabling different optimization configurations. In all replay experiments, we executed the Medusa proxy on a 1133MHz PIII with 512KB cache and 1GB of memory running Linux 2.4.2-2. We ran the Medusa proxy using Sun's Java 2 Runtime Environment (v1.4.0) with the same version of HotSpot. We used Internet Explorer 5 (IE), Netscape V4.79, and OmniWeb v4.1b5 as the browsers to generate our workloads. Replay measurements were gathered overnight on May 6–7 and June 22–27 with each of seven machines replaying parts of the traces. To minimize warming effects, for each object we take the median download time across five downloads.

User	Reqs	Pgs	Ave Reqs/Pg
A	1212	87	13.9
B	872	103	8.5
C	10341	568	18.2
D	650	70	9.3
E	477	68	7.0
F	195	24	8.1
Total	13747	920	15.0

Table 3: The six user traces of successfully replayed pages collected from Saturday, April 27, 2002 through Tuesday, April 30, 2002.

4.3 Optimizations

For each experiment, we used the Medusa proxy to replay user traces. During playback, we enabled different optimization options to mimic optimizations that are commonly used by Web browsers and servers. Below we describe our methodology for page delimitation and the optimizations we study.

Page Delimitation. To divide user traces into pages we used the relatively common heuristic of calculating inter-object times in the original trace to distinguish page boundaries. While [5] used an intra-page reference time of 2.25 seconds, we found that a value of 2 seconds was sufficient for our users who had excellent connectivity to the Internet. We have verified that using 2 seconds as a page break allows us to match known page boundaries in our traces.

We use page delimitation both in calculating total page costs and in limiting optimizations to a page. Specifically, Medusa uses the page delimitation to limit parallelization to within a page.

(1) Parallel Connections. The Medusa proxy tracks the number of unique concurrent connections used by the browser during trace collection. These connection numbers are then used to replay parallel download.

In parallel replay mode, all parallel downloads are begun at the same time, even though this is not the case during browser download. As a result, our numbers represent an optimistic lower bound for parallel page performance.

To report parallel page costs, we compute the time spent per download sequence. We then take the maximum sequence download time as the page download time, since this connection will dominate page download time. Although this may underestimate download time, it excludes overhead Medusa may introduce.

(2) CDN Usage. As explained in [11], some objects hosted on Akamai servers are named with URLs that reveal their location on origin servers. When the CDN usage optimization is disabled, Medusa uses a filter to remove the references to Akamai servers and replaces them with references to the origin servers. When CDN usage is enabled, Medusa simply leaves the traces in their original state. We do not attempt to simulate more CDN usage than is found in the original trace.

Since Akamai accounts for 85–98% of CDN-hosted objects [12], we have focused on identifying and removing Akamai hosted objects only.

(3) DNS Caching. We simulate idealized optimal DNS caching by simply not including DNS timing information when Medusa replays objects. When idealized DNS caching is turned off, we add the original DNS lookup cost (as measured by Medusa when the trace was originally generated during user browsing) to the object cost. In both cases, Medusa does not record DNS lookup during replay. Note that this original cost will reflect the benefits of standard locality-based DNS caching, or the cost of DNS redirection as the case may be, since Medusa measures and records the time to do the original DNS resolution. As a result, the difference between enabling and disabling Medusa's DNS caching reflects the overhead of DNS resolution in the original trace.

(4) Persistent Connections. As with parallel connections, with persistent connections Medusa attempts to mirror the connection state of the original trace. Medusa supports both HTTP 1.0 KeepAlive and HTTP 1.1 style persistent connections; in our experiments, we use whichever request syntax the original browser used. As with CDN usage, we do not attempt to model more aggressive persistent connection usage than would be possible given the number of connections in the original trace; Medusa only uses persistent connections during replay for objects that were downloaded over persistent connections in the original trace. We assume that the browser has already attempted maximum persistent connection usage, and has recorded failures as close connection events in our user trace. We are thus simulating persistent connections with perfect knowledge. We do not count failed persistent connection attempts in our timing data, we simply begin timing at the new connection open.

4.4 Configurations

We ran experiments with the following optimization combinations.

- No optimization options (serial download).
- Persistent connections only.
- Parallel connections only.
- Parallel connections and CDNs.
- Parallel connections and DNS caching.
- Parallel connections and persistent connections.
- Parallel connections, CDN usage and persistent connections.
- Parallel connections, CDN usage and DNS caching.
- Parallel connections, DNS caching and persistent connections.
- Parallel connections, CDN usage, DNS caching, and persistent connections.

5 Results

As stated in the introduction, the high level goals of this paper were to examine the extent to which the optimizations above were used and to study the effect of these optimizations on whole page performance. The following sections address both of these goals by characterizing the pages in our trace and evaluating whole-page performance using the combinations of download optimizations listed above.

We downloaded 920 pages with a total of 13747 objects for an average of 15.0 objects per page. Pages with one or more download errors during replay were discarded (this includes both connections errors and 4XX/5XX responses).

5.1 Page Characterization

We first characterize the pages in our traces in terms of the number and types of objects per page, as well as the use of embedded objects from Akamai and advertisement servers.

5.1.1 Objects per page

Figure 1 shows the distribution of different page sizes in terms of the number of objects in each page. The values on the x-axis correspond to the number of objects in a page, and the values on the y-axis correspond to the number of pages in the traces that have a given number of objects. Pages with only one object have no embedded objects. The large number of small object pages is due to a combination of factors, such as caching and refresh effects.

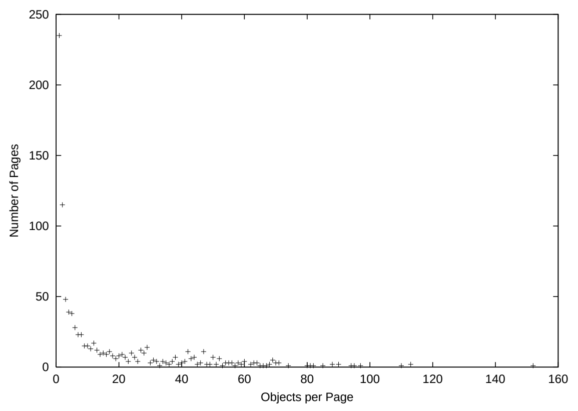

Figure 1: Distribution of objects per page.

Objects per Page	No. of Pages	No. of Objects
1	235 (25.5%)	235 (1.71%)
2-5	240 (26.1%)	720 (5.24%)
6-15	165 (17.9%)	1577 (11.5%)
16+	280 (30.4%)	11215 (81.6%)
Total	920	13747

Table 4: Number of pages successfully downloaded in each page size category. Mean objects per page is 15, median is 5.

Because of this we categorize pages according to the number of objects they contain.

Table 4 shows the breakdown of pages in the trace according to the number of objects in the pages. Since the median number of objects per page is 5 and the average is 15, we use these values as category boundaries. Note that, although relatively small pages (1–5 objects per page) comprise 51.6% of the pages, they only constitute 9% of the requests in the trace.

5.1.2 Object Type

In terms of the types of objects requested, we found that 80.7% of URLs were images, 3.3% were javascript, and 3.0% were CSS files. 3.8% were found to be cgi, perl, or class files. Surprisingly, only 5.6% identified as ".html" or ".htm" files (this does exclude pages with no identifying suffix). The remaining 3.6% of pages were pages with no suffix and miscellaneous files (pdf, txt, etc.).

5.1.3 CDNs

We found that the number of pages with explicit Akamai-hosted objects is small, with 48 pages or 5.2% of pages containing Akamai-hosted objects. Akamai-hosted objects accounted for only 216, or 1.6%, of total downloaded objects. For pages that contained an akamai reference, there were an average of *216/48* = 4.5 CDN objects per page. Pages that contained CDNs had an average of 12.06 objects per page.

5.1.4 Persistent Connections

We found that the number of pages that use persistent connections is also small, with only 229 pages or 20% of the pages in out trace using persistent connections. These pages contained 2211 persistently downloaded objects, accounting for 16.1% of total objects downloaded.

5.1.5 Ad-Servers

We searched for ad servers by looking for references to *hosts* that were named with the phrases "ads" or "adservers". Thus, objects such as

http://rmads.msn.com/images_47144_date0429_50.jpg

were flagged, but objects such as

http://graphics4.nytimes.com/ads/scottrade_sov.gif

were not.

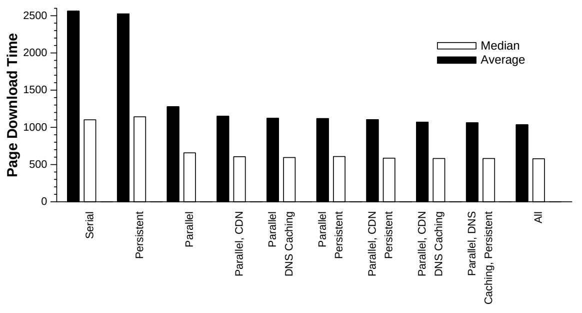

Figure 2: Average page cost with different optimization options enabled. Numbers along the bottom correspond to numberings in methodology section. Averages are the top points for each option, and medians are the bottom points.

The number of total ads served by identifiable adservers in our dataset was 211, which were distributed across 87 (9.5%) pages. This is an average of 2.4 ads per page for those pages that contained ads. Pages that contained ads had an average of 30.75 objects per page.

5.2 Whole Page Optimizations

In this section we evaluate the performance of the different download optimizations. Figure 2 shows the average page performance with different optimizations enabled. The x-axis corresponds to a configuration of optimizations numbered according to the list in Section 4, and the y-axis corresponds to the page download time. As this figure shows, the optimization that has the greatest incremental effect across all pages is enabling parallelism. For example, using parallel connections alone improves performance 100.5%. All other improvements are moderate: CDN improvement is 2.5% with all other options turned on and DNS improvement is 6.7% with all other options turned on. Surprisingly, even persistent connections are not as helpful as expected. With no other options turned on, persistent connections improve performance only 1.5%, and with all other options turned on they only improve performance 3.4%.

We also find that all two option settings perform better than one option settings, and all three option settings perform better than two option settings with the four option setting of using parallelism (1), CDNs (2), complete DNS caching (3), and persistent connections (4) being the most effective. Figure 2 also shows that parallelism and any one option gives almost equivalent improvement to having all options enabled. The maximum overall benefit comes from adding the first two optimizations. Finally, notice that the medians are much lower than the means in this graph. This is both because of the high variance for page download, and a skew towards smaller pages in the data set (distribution of page sizes).

From this graph we can draw some overall conclusions. The first is that parallelism has the greatest effect across the pages in this workload. This follows from the fact that parallelism is the only performance optimization used aggressively *on all pages* (a related study found that a browser may open as many as 28 simultaneous connections [1]). While none of the other optimizations are ineffectual, they all offer only incremental benefits. There are two reasons for this. In some cases (e.g. DNS), the overheads involved may be small, allowing for only modest improvement. In other cases, looking at the average over all pages may not provide a realistic estimate of the improvements offered by an optimization (e.g. persistent connections). Not all pages implement all optimizations, and not all pages that implement those optimizations use them aggressively. Below, we examine each of the other optimizations in detail, examining factors that contribute to their modest improvement.

5.3 Ideal DNS Caching

We found that the average DNS cost per object is 7.1 ms and the average DNS cost per page is 529.7 ms. This represents total time spent doing DNS lookups, not time the browser spends waiting on a DNS lookup since with parallel connections lookups can also be done in parallel.

We can quantify the amount of time that the browser spends waiting on a DNS lookup by comparing average whole-page latency with and without DNS caching. We see that DNS cost is 12.2% of parallel page cost, 6.9% of parallel/CDN page cost, 5.1% of parallel, persistent connection page cost, and 6.3% of parallel, persistent, CDN page cost.

Looking at Figure 2, we see that the benefit of idealized DNS caching is moderate, but that it is relatively helpful in all cases. This is due to two factors. The first is the fact that we already have less-than-ideal, but still effective DNS caching in place. Thus, not every DNS lookup needs to query a remote nameserver. The second factor is that shown above, DNS queries overlap with other page download operations. This leaves only modest room for improvement, even when specific DNS requests have high overhead.

5.3.1 DNS and CDNs

We also examine the interactions between DNS and CDNs at the page level by looking at DNS overhead for only those pages which contain CDN hosted objects.

We were interested in whether DNS costs are higher for those pages which contain a large number of Akamai-hosted objects. We find that is not necessarily true, however. For example, DNS contributes 0.8% to parallel connections with CDNs for pages that use Akamai, compared with 6.9% on average. For parallel, persistent connections with CDNs, DNS contributes 6.0% to pages that use Akamai, compared with 6.3% for average pages.

5.4 CDNs

While perfect DNS caching is only incrementally effective because existing DNS caching is already

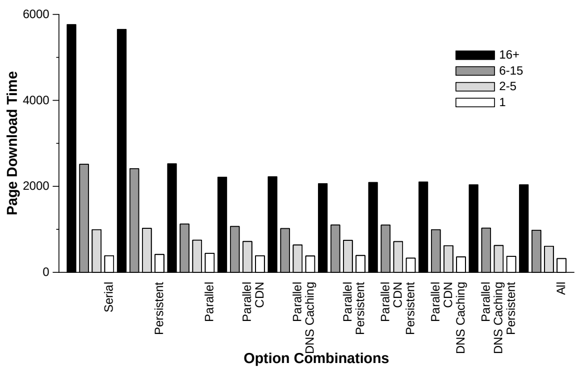

Figure 3: Average page download time where objects are grouped by page size.

effective, the same is not true for CDNs. Previous studies have shown that CDNs are highly effective for individual objects [11]. We note that while CDNs do give improvement in our overall results (1.3% -11.1% and 2.5% with all other options enabled), overall their contribution is small.

This small improvement corresponds to the similarly small number of Akamai-hosted objects found in our traces. We found that only 48 pages (5.2%) contain Akamai-hosted objects, with a total of 216 objects for an average of 4.5 Akamai-hosted objects per page. As in [11], from an overall workload perspective, CDNs have an impact on whole-page download performance that corresponds to their limited part of user workloads.

5.5 Persistent connections

We have observed that persistent connections do not appear to have as substantial of an impact on performance as we would expect. The reason for this is that looking at all pages in the trace does not tell the whole story: Many pages have little opportunity to benefit from persistent connections. For example, pages with one object count as much as pages with 152 objects, even though connection optimizations will be much more effective on larger pages. In this section, we characterize the potential for the pages in our trace to benefit from persistent

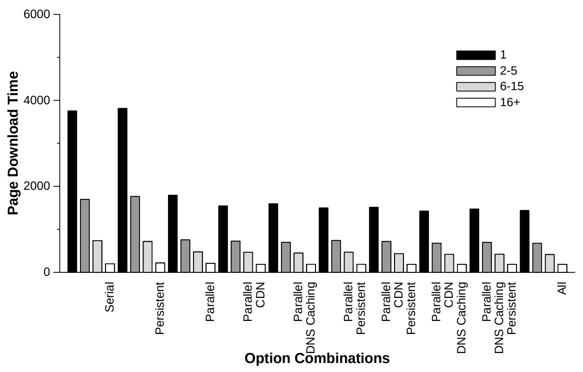

Figure 4: Median page download time where objects are grouped by page size.

connection, first by objects per page, then by percentage persistent objects. We present the performance impact of persistent connections.

5.5.1 Page Breakdown

We begin by looking at objects per page as a possible explanation for the lack of impact of persistent connections. Figures 3 and 4 show the effect the different optimizations have on pages with differing numbers of embedded objects. We have categorized pages based on the median number of embedded objects per page (5) and the average number of embedded objects per page (15). We find that pages with one object only improve 20.8% from *all* optimizations, and pages with 2-5 objects only improve 63.4%. Meanwhile, larger pages do quite well. Pages with 6-15 embedded objects improve 156.9% overall, while pages with 16 or more objects improve 182.9% overall. However, we still find that persistent connections are not as effective as we would like, even on the large pages. For example, on pages with 16 or more objects, persistent connections only improve the serial case 2.0% and only improve the parallel case (with no other options) 22.5%.

It is also interesting to note that Figures 3 and 4 show persistent connecions are more helpful in the parallel case than in the serial case. Lack of persistent connections contributes only 2.0% to the cost of downloading a serial page, while contributing 18.3% in the parallel case for pages with more than 16 objects. This appears to be due to the fact that requests in the serial case may get interleaved when they are sent on one connection in the order in which they were originally downloaded by the user's browser. For example, using parallel connections, two requests to the server www.cnn.com may get sent in sequence 0 while one request to ar.atwola.com may get sent in sequence 1. When Medusa serializes these requests, it is possible that they may get ordered as (www.cnn.com, ar.atwola.com, www.cnn.com), negating persistent connection benefits for one connection. Thus, the opportunity for taking advantage of persistent connections is smaller for the non-parallel case.

We can draw several conclusions from Figures 3 and 4. First, not surprisingly, all of the optimizations have more effect when there are more objects per page. This is because there is more opportunity to take advantage of parallel and persistent connections. For example, persistent connections provide a more significant improvement for larger pages, reflecting the fact that larger pages contain more objects on the same servers. Second, many of these optimizations are cumulative. Finally, single object pages can only be improved by the elimination of DNS overhead and CDNs (unless they are located on the same server as the previous page, in which case persistent connections might be useful). Thus, these pages do not exhibit much improvement overall, but they account for 26% of pages overall.

5.5.2 Mostly Persistent Pages

We would expect to see more improvement from the use of persistent connections than is found above. Consequently, we took a closer look at the use of persistent connections in our trace. We found that only 20% of the pages in our trace use persistent connections (229 pages), accounting for 16.1% of objects (2211 objects). Because of this, we focus only on those pages that have the potential to benefit from their use.

Objects per Page	**No. Pages**	**Ave. # of Objects**	**Ave. # Persistent Objects (% Objects)**	**Ave. # Servers per Page**	**Ave. # Persistent Objects/Server**
2-5	59	3.02	1.95 (64.57%)	1.48	1.32
6-15	25	9.68	6.00 (61.98%)	2.16	2.77
16+	100	46.42	17.81 (38.38%)	4.54	3.92
All	184	23.76	9.66 (40.66%)	2.89	3.34

Table 5: Various characteristics of pages where persistent connections were used in the original trace.

Table 5 shows various characteristics of the pages in our trace that have the potential to benefit from persistent connections. The pages are broken down into categories determined by the number of objects in the page; we omit the one-object category since one-object pages cannot benefit from persistent connections in isolation. These pages are the ones that used persistent connections in the original trace. Recall that Medusa only uses persistent connections during replay when downloading objects that were downloaded using persistent connections originally. Table 5 shows the number of these pages in the first column of the table; comparing the **All** row with the **Total** row in Table 4, we see that only (184/920 = 20%) of all pages in the trace used persistent connections.

The remaining columns in Table 5 give further insight into the potential benefit of persistent connections for these pages. For persistent connections to be useful when Medusa replays traces, two conditions must hold: (1) multiple objects were downloaded over persistent connections in the original trace, (2) these objects were downloaded from few servers to benefit from connection reuse. The "Ave. # of Objects" column in the table shows the average number of objects in the pages in the category. Since not necessarily all of the objects of a page were originally downloaded using persistent connections, the next column shows the average number of objects downloaded in the pages in the category; the percentage of all objects in the page is given in parentheses.

From the table we see that, for pages with less than six objects per page, on average only two out of three objects are downloaded using persistent connections. For these pages, then, the use of persistent connections only has the potential to save the connection overhead for one object compared to using serial connections. For pages with many objects ("16+"), a substantial number of objects are downloaded over persistent connections on average (approximately 18) and we would expect persistent connections to benefit these pages. Note, though, that the full potential of using persistent connections for these pages is only partially realized since only 38% of the objects on these pages are downloaded using persistent connections.

The "Ave. # Servers per Page" column shows the average number of different servers used to download the objects for the pages, and the final column shows the average number of objects downloaded using persistent connections in the page divided by the number of servers. For example, if a page used persistent connections to download six objects equally from two servers, then its corresponding entries in the columns would be 2.0 and 3.0. The final column characterizes the average "opportunity" of using persistent connections for these pages: it measures the average number of objects downloaded per persistent connection for the pages in each category. For the "2-5" pages, this metric is 1.32: on average, using persistent connections will save the connection overhead for the remaining 0.32 objects beyond the initial object from the connection (or, 1 out of every 3 such pages). On the other hand, for the "16+" pages four objects are downloaded per persistent connection on average. Furthermore, since the objects on these pages are downloaded from 4.5 servers on average, they also have good potential for using parallel persistent connections.

In Figures 2– 4 we found that the use of persistent

Objects per Page	Method	< 50% Persistent			>= 50% Persistent		
		Pages	Mean	Median	Pages	Mean	Median
2-5	serial	7	566	476	52	1550	865
	persistent		601	482		1630	741
6-15	serial	11	1670	811	14	4000	1780
	persistent		1530	779		2680	1580
16+	serial	63	7160	3960	37	6180	3620
	persistent		7410	4040		4660	3390

Table 6: Page download times for those pages where persistent connections were used. For comparison, page download times when using both serial and persistent connections are shown. Pages are categorized by the number of objects they contain, and divided by whether more or less than half of the objects in the pages were downloaded over persistent connections. All times are in milliseconds.

connections did not have a significant impact on page download time across all pages. Table 5 above showed that the primary reason is that most pages in the trace cannot benefit from the use of persistent connections. To focus on the impact of persistent connections on just those pages that can benefit from their use, Table 6 shows the page download times for those pages that used persistent connections in the original trace. We categorize the pages according to the number of objects they contain, and compare the performance of using the serial and persistent connection optimization methods. We further divide the pages by whether more or less than half of the objects in the pages were downloaded over persistent connections.

The page download times in Table 6 show that the use of persistent connections does improve performance for those pages that can benefit from them. These are the pages one would expect: they have a substantial number of objects in the page, and most of those objects are downloaded over persistent connections. For the pages with 6–15 and 16+ objects, and at least half of the objects are downloaded over persistent connections, persistent connections improve page download times by 32.6–49.3% over using serial connections.

5.6 Ad Servers

We also looked at the relative improvements of the different options on only pages containing ads. We find that all optimizations improve the performance of pages containing ads. Adding CDNs to the parallel case improves performance 11.0%. Adding DNS caching improves the parallel case 12.3% (compared with an of average 13.9% over all pages), improves the parallel/CDN case 8.9% (compared with 7.4% over all pages), and improves the parallel, persistent, CDN case 4.9% (compared to 6.7%).

6 Conclusion

In this paper we have explored user-perceived Web performance of downloading entire pages, and how various optimizations impact overall page performance. To explore whole-page performance, we have extended the Medusa proxy to support parallel and persistent connections, as well as to record DNS lookup times. We then use the Medusa proxy on user traces to characterize whole-page performance and measure the impact of request optimizations on downloading entire pages.

In general, we found that the connection optimization that has the largest impact across all pages in our trace is simply parallel connections. We found that persistent connections, for example, were not as widely used as we expected. They were only used in 20% of the original page downloads in our trace, and they primarily benefit those pages that have at least six objects in the page and most of those objects are actually downloaded over persistent connections. We found that the connection optimizations have a greater effect the larger the pages. With an average number of 15 objects per page in our user traces, there is greater opportunity for parallelism and connection reuse. We also found that explicit Akamai-hosted objects were rare (5.2% of pages), and their effect limited. Finally, we found that perfect DNS caching had a limited but uniform effect across all pages, indicating that DNS caching is already fairly effective. Of course, as with any trace-based analysis, these results reflect our workloads and our computing and network environment, and results can differ for other environments.

7 Acknowledgements

We would like to thank Jeff Chase and the anonymous reviewers for their helpful comments and suggestions. This research was supported in part by DARPA Grant N66001-01-1-8933.

References

[1] L. Bent. Simultaneous parallel connections under two browsers, http://ramp.ucsd.edu/medusa/parallel.html.

[2] 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.

[3] E. Cohen and H. Kaplan. Proactive caching of dns records: Addressing a performance bottleneck. In *In Proceedings of the Symposium on Applications and the Internet (SAINT)*, January 2001.

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] J. Jung, E. Sit, H. Balakrishnan, , and R. Morris. Dns performance and the effectiveness of caching. In *ACM SIGCOMM Internet Measurement Workshop*, November 2001.

[11] M. Koletsou and G. M. Voelker. The medusa proxy: A tool for exploring user-perceived web performance. In *Sixth International Workshop on Web Caching and Content Distribution*, Boston, MA 2001.

[12] B. Krishnamurthy, C. Wells, and Y. Zhang. On the use and performance of content distribution networks. In *ACM SIGCOMM Internet Measurement Workshop*, November 2001.

[13] 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.

[14] B. Krishnamurthy and C. E. Wills. Analyzing factors that influence end-to-end web performance. *WWW9 / Computer Networks*, 33(1-6):17–32, 2000.

[15] 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.

[16] R. Liston and E. Zegura. Using a proxy to measure client-side web performance. In *Proc. of the 6th Int. Web Caching and Content Delivery Workshop*, June 2001.

[17] 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.

[18] A. Shaikh, R. Tewari, and M. Agrawal. On the effectiveness of dns-based server selection. In *Proc. of IEEE INFOCOM 2001*, Anchorage, AK 2001.

[19] C. Wills and H. Shang. The contribution of dns lookup costs to web object retrieval. In *Tech. Rep. TR-00-12, Worcester Polytechnic Institute*, July 2000.

[20] C. E. Wills, M. Mikhailov, and H. Shang. N for the price of 1: bundling web objects for more efficient content delivery. In *World Wide Web*, pages 257–265, 2001.

[21] 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.