Towards Informed Web Content Delivery

Leeann Bent[1], Michael Rabinovich[2], Geoffrey M. Voelker[1], and Zhen Xiao[2]

[1] University of California, San Diego
9500 Gillman Dr. MS-0114
La Jolla, CA 92093-0114 USA
{lbent,voelker}@cs.ucsd.edu

[2] AT&T Labs-Research
180 Park Ave.
Florham Park, NJ 07932 USA
{misha,xiao}@research.att.com

Abstract. A wide range of techniques have been proposed, implemented, and even standardized for improving the performance of Web content delivery. However, previous work has found that many Web sites either do not take advantage of such techniques or unknowingly inhibit their use. In this paper, we present the design of a tool called Cassandra that addresses these problems. Web site developers can use Cassandra to achieve three goals: (i) to identify protocol correctness and conformance problems; (ii) to identify content delivery performance problems; and (iii) to evaluate the potential benefits of using content delivery optimizations. Cassandra combines performance and behavioral data, together with an extensible simulation architecture, to identify content delivery problems and predict optimization benefits. We describe the architecture of Cassandra and demonstrate its use to evaluate the potential benefits of a CDN on a large Web server farm.

1 Introduction

A wide range of techniques have been proposed, implemented, and standardized for improving the performance of Web content delivery, from compression to caching to content distribution networks (CDNs). However, previous work has found that many Web sites either do not take advantage of such techniques, or unknowingly inhibit their use. For example, Web sites do not fully exploit the potential of persistent connections [6], and indiscriminate use of cookies can unnecessarily and unknowingly limit the benefits of content delivery optimizations like caching [5].

This gap between potential and practice is due to a number of challenging factors. First, HTTP/1.1 is a complex protocol [22] that requires considerable expertise to fully take advantage of its features. Even achieving protocol compliance is challenging (as evidenced by intermittent success [17]), much less optimizing its use. For instance, HTTP/1.1 provides a number of advanced features for cache coherence so that sites can maximize the effectiveness of downstream caching mechanisms. However, for Web site developers to take full advantage of cache coherence, they must understand the complex interactions of over half a dozen HTTP headers in terms of their standardized semantics, legacy interactions, as well as practical conventions [25]. Second, the use and content of contemporary Web sites have complex requirements that, used naively, can interact poorly with content delivery optimizations. For example, many sites use cookies to personalize, restrict, or track access, but, again, indiscriminate use can severely inhibit downstream caching optimizations. Finally, measuring Web site workloads and evaluating site performance requires considerable time and effort, and evaluating the potential effectiveness of content delivery optimizations rarely extends beyond the realm of research. Top-tier Web sites like Google and Amazon.com may have the resources to aggressively improve their content delivery, but even medium-sized and small Web sites can benefit significantly from a number of improvements [5].

We propose that this gap between potential and practice can be bridged with a general tool for analyzing and predicting Web site performance and behavior. By encapsulating protocol and optimization complexities within a tool, a much broader user base of Web site developers and maintainers can more easily quantify the effect of their content delivery decisions and evaluate potential performance optimizations.

In this paper, we present the design and use of such a tool called Cassandra. Web site developers can use Cassandra to achieve three goals. First, Cassandra can identify protocol correctness and conformance issues, such as properly setting cache control and coherence headers. Second, it can identify content delivery performance problems, such as the indiscriminate use of cookies. Third, it can evaluate the potential benefits of using content delivery optimizations for a particular site, such as tuning cache control headers, using cookies more efficiently, using CDNs, enabling persistent connections, etc. Based on such evaluations, a site developer can decide whether pursuing a particular optimization is worth the effort and expense before committing to it.

To achieve these goals, Cassandra's design combines an extensible architecture of *analysis modules* with site-specific performance and behavioral data. The analysis modules model protocol behavior, such as content cacheability, and simulate performance optimizations, such as compression and CDNs. To apply these analyses to a particular site, Cassandra can use a combination of raw packet traces, Web server logs, and active probing to gather workload data as input.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes the architecture of Cassandra and outlines its analysis modules. Section 4 describes the prototype implementation of one analysis module, the analysis of CDN benefits. Section 5 presents a case study that applies the CDN analysis module in Cassandra to the top 100 Web sites in a large server farm. Section 6 concludes this paper and outlines directions for future work.

2 Related Work

Most of the previous studies that analyze Web sites were conducted on a small scale (sometimes just a single, highly popular site) [13, 17, 20, 21, 24]. Some of them (e.g., [17, 20, 24]) only consider access patterns to root pages. Moreover, they typically only focus on a specific aspect of the Web site. For example, the work in [17] focuses primarily on protocol compliance, while the work in [20] focuses on persistent and parallel connection usage. While these studies may yield valuable insights on certain Web site design issues, their results are not as comprehensive as those obtainable a tool like Cassandra.

The benefits of using a CDN for content delivery were previously studied in various contexts. Jung et al. investigate the ability of a CDN to protect a Web site from a flash crowd by analyzing known flash events that occurred on two Web sites [14]. Krishnamurthy et al. conducted a comparative performance study on the download time of pre-selected pages through various CDNs [19]. In contrast, the focus of Cassandra is to evaluate the general effects of the CDN on an entire Web site.

There are several existing tools that measure Web server performance by generating various HTTP workloads. httperf is a commonly used tool that provides a flexible interface for constructing various benchmarks [23]. It supports the HTTP/1.1 protocol and can be easily extended to new workload generators and performance measurements. Similar to httperf, Web-Polygraph is a tool for constructing Web benchmarks [28]. Web-Polygraph includes standard workloads generated from real Web content and supports HTTP and SSL content. SURGE (Scalable URL Reference Generator) generates references that match empirical measurements of various request distributions (e.g., request size, relative file popularity, temporal locality) [4]. Banga et al. proposed a method for Web traffic generation that can create heavy, concurrent traffic through a number of client processes. It can generate bursty traffic with peak loads that exceed the capacity of the server [3]. Although such tools are helpful in measuring server performance and protocol compliance, they use synthetic workloads and thus are not specific to a given Web site. In contrast, Cassandra produces site-specific recommendations and a comprehensive set of "what-if" analyses, which greatly facilitate the understanding and improvement of Web site performance.

There are also numerous commercial tools and services for monitoring and testing the performance of Web sites. These generally fall into two categories. The first set of tools are designed to monitor the current state or performance of a Web site. They answer questions such as "Is my Web site up?" or "How long is an average transaction on my web taking from South America?". Some examples of these are Keynote's Web Site Perspective [15] and AlertSite's Server/Website Monitoring [1]. The second set are tools designed to stress test the performance of a Web site, answering questions such as "How should I provision my site for search transactions?". Some examples include Keynote's Performance tune, Empirix's e-Test suite [10], and AlertSite's Web Load Testing Services. These tools focus on measuring server and transaction performance. While useful for analyzing existing Web site performance, these tools do not easily facilitate "what-if" analysis like Cassandra.

Finally, there are free tools for testing the performance and cacheability of Web pages, e.g `www.web-caching.com` [7] or `www.websiteoptimization.com` [29]. While cachability is only one aspect of the Cassandra tool, Cassandra also provides a means for detecting pervasive problems throughout a Web site rather than on individual pages.

3 The Cassandra Architecture

The high-level goal of the Cassandra tool is to make it easier for Web site developers and maintainers to both quantify the effect of their content delivery decisions, as well as to evaluate potential optimizations for improving content delivery performance. We envision Cassandra helping Web sites achieve these goals in three ways.

Protocol Compliance. HTTP is a complex protocol with complicated semantics, particularly with respect to caching and coherence. Due to the nature in which the HTTP protocol has evolved over time, taking full advantage of protocol features is a difficult task. In addition to a detailed understanding of standardized protocol semantics, it also requires understanding how protocol features interact with legacy implementations of caches and browsers as well as practical conventions for dealing with ambiguous aspects of the protocol [25]. Cassandra could be used to evaluate protocol compliance and report anomalous behavior, such as inconsistencies among the cache-control headers.

Performance Debugging. Web sites can make content delivery decisions that can unknowingly have a profound impact on site performance. For example, when many Web sites use cookies, they often implement cookie use by requiring all requests to the site to use a cookie [5]. Using cookies in this fashion is often unnecessary, yet severely limits the cacheability of a site's content. Cassandra can identify such behavior, estimate the performance impact of a site's use of cookies (e.g., additional server load and bandwidth consumed), and identify content that may not require cookies (e.g., embedded images versus container pages).

Optimization Evaluation. A wide range of technologies have been developed over time to improve Web content delivery, such as compression, prefetching, and CDNs. However, most of these technologies are only selectively deployed, if at all. A significant hurdle for adopting a technology is evaluating to what extent it would benefit a particular Web site. Cassandra could be a valuable tool for Web site operators to address this problem by facilitating the evaluation of various technologies on a particular Web site. Based on the results of such an evaluation, Web site operators can make an informed decision regarding the optimal content delivery methods for their Web site.

We have implemented an initial prototype of Cassandra, as well as an analysis module for evaluating the benefits of using CDNs for Web sites. In the rest of this section, we discuss the Cassandra architecture and outline two other analysis modules that we plan to implement. Then, in Sections 4 and 5, we describe in

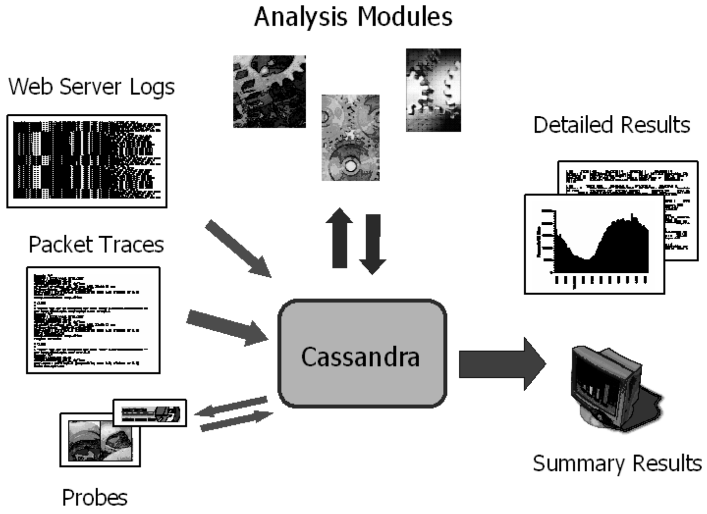

Fig. 1. Cassandra Architecture.

detail the CDN analysis module that we have implemented and evaluated as an example demonstration of using Cassandra.

3.1 Architecture

To achieve the above goals, Cassandra's design combines an extensible architecture of *analysis modules* with site-specific performance and behavioral data. As illustrated in Figure 1, Cassandra provides a framework for incorporating workload data as input, evaluating that data according to specified analyses, and providing support for producing user feedback in terms of tabular and graphical results of analyses as output. In our usage model, Web site maintainers gather the workload data, invoke Cassandra to perform the desired analysis, and interpret the evaluation results to decide whether they should make content delivery changes to their site.

The ability of Cassandra to perform its analyses depends on the site workload data available to it. Cassandra can use any combination of raw packet traces (such as from the Gigascope network appliance [9] or `tcpdump` [27]) , Web server logs, and active probing for workload analysis. Packet traces provide the most detailed workload data, since they include headers, payload, and timing information. If packet traces are unavailable, Cassandra can use Web server logs to obtain higher level workload information, such as identifying popular content or content that affects Web site performance most significantly. After identifying important content, Cassandra can combine active probing or short `tcpdump` traces with server logs to obtain header and performance data. This detailed data drives its performance analyses.

Cassandra uses an extensible architecture of *analysis modules* to support a large range of Web site analyses. The analysis modules model protocol behavior and simulate performance optimizations. For example, a coherence module can model the cache coherency policies and mechanisms of HTTP, and a CDN module can model the potential benefits of using a CDN for a Web site. Cassandra provides a framework in which analysis modules can be developed and used independently, as well as in combination. For example, Cassandra implements a Web cache simulator, and this simulator can be used in combination with other analyses such as those that modify the cacheability of site content.

Our goal is for Cassandra to be able to support a wide range of Web site analyses and content delivery optimizations, from caching to prefetching to persistent connections. We are developing modules for a wide range of common analyses, such as caching, cache coherence, compression, and CDN usage. The Cassandra architecture also makes it straightforward for other researchers to extend Cassandra with their own analyses.

In addition to analyzing individual Web sites, Cassandra supports the simultaneous analysis of multiple Web sites using workloads containing interleaved accesses to those sites, and the reporting of results on a per-site basis. Such support is useful for situations where a hosting service provider wants to analyze performance across a number of its Web site customers. With this support, Cassandra avoids the need to split the trace into individual Web site accesses and simplifies the per-site analysis. We use this feature for our case study in Sections 4 and 5.

3.2 Cacheability Analysis Module

A cacheability analysis module can evaluate the use of a Web site's cache-coherence mechanisms and policies. Content cacheability is important because it increases the effectiveness of downstream proxies and browser caches. Our previous work in [5] motivates a number of problems that this analysis module can identify. For example, in that workload study only a small fraction of responses (9%) used cache-controlling headers, thereby requiring downstream caches to use heuristics and historical practices to provide coherency for the objects in the responses. Explicit and pervasive use of such headers can greatly assist caches in managing cached content, further reducing server load and bandwidth. The cacheability analysis module can identify content that could benefit from using cache-controlling headers, model the effect of using those header on a site's workload, and predict the caching benefits of doing so. As another example, sites often use cookies indiscriminately on their content. Such indiscriminate use severely limits the benefits of caching, and is often unnecessary. For instance, when using cookies to track user accesses, it is typically sufficient to track accesses to container pages and infer accesses to embedded objects. The cacheability analysis tool can measure cookie use, flag excessive usage, identify content that may not require cookies (e.g., embedded objects), and predict the caching benefits of using cookies in a more informed approach.

The cacheability analysis module can perform other analyses as well. It can check that the cache control headers are internally consistent (e.g., `Date`, `Age`, `Cache-Control` and `Expires` headers). And the module can predict the effect on overall workload of tuning time-to-live (TTL) values. For example, repeated `If-Modified-Since` requests to an object might indicate that the site is setting the object's TTL value too low, and Cassandra can estimate the resulting impact on server load and bandwidth consumption of such settings.

Cassandra can use object headers, obtained by either probing objects on a Web site or by looking at packet traces, to obtain cacheability information about the objects on a Web site. Packet traces would allow both request and response object headers to be examined; probing would only allow response object headers to be examined. Such headers include `Pragma` headers, `Cache-Control` headers, `Cookie` headers, `Age`, `Expires`, and `Last-Modified`. The cacheability analysis module would use these headers, combined with object access information, to identify and report specific problems with a Web site's content cacheability or delivery. Object cacheability would be determined according to the HTTP/1.1 specification [12] as well as known prevalent caching policies used in practice.

3.3 Compression Analysis Module

A compression analysis module for Cassandra can simulate the benefits of using compression on some or all of the content of a Web site. Content compression is useful because it can reduce the bandwidth demands of a Web site, as well as reduce the download latency of objects. For example, Google compresses the results of search queries to reduce the number of packets required to return search results [8]. The goal of this module is to estimate the benefits of compression (bandwidth and latency reduction) as well as the costs of using the optimization (e.g., additional server load).

Since compressing all content on a Web site might be prohibitively expensive, we plan to model compression of only a subset of objects on a site. These objects can be chosen according to their popularity and contribution to bandwidth consumption as determined from server logs or packet traces. To estimate compressibility, we intend to use Cassandra's site prober to actually download objects or `tcpdump` to reconstruct these objects from network traces. After obtaining the objects, we will apply a common compression utility to them. Once objects are obtained and compressed, we have a measure of object size in bytes, both with and without compression. We can compute bandwidth reduction for a particular site or group of sites by replaying access logs with both compressed and uncompressed objects.

In addition, we can also consider compression in conjunction with CDN usage. Most Web access are image downloads, which are already in a compressed format, thus undermining the effectiveness of compression. However, while images are responsible for most downloads, they are also the object most amenable to CDN caching. For a site using a CDN, most of the residual (non-image) content is compressible. Further, static content can be stored in compressed form and does not incur computation overhead, while compression of dynamic content might have lower relative overhead because it already requires more computation to be generated. Cassandra can test whether objects are dynamic by the presence or absence of the `Last-Modified` header together with the response code. This heuristic can be used to determine whether it would be beneficial to compress the object, and how to do so. We can study the effect of using a CDN (both ideal and realistic) with compression, similar to the way we study the effect of using a CDN with cacheability improvements. Cassandra would compute the benefit of applying compression with a CDN by compressing those objects as identified above, and using those compressed objects in the CDN simulation as in Section 4.

Compression can also be used to decrease client download times. It is a little more difficult to compute download latency and we do not lay out a complete blueprint here. However, we can make a preliminary estimate of download latency improvements for individual objects by comparing the download time of the original objects with objects that are approximately the same size as compressed objects.

Thus, Cassandra will be able to analyze realistic benefits of compression for a Web site, taking into account pre-compression of static objects, and considering compression in combination with CDN caching. Since the CPU overhead of compression depends on the exact platform used, Cassandra can be used in initial analyses to ascertain whether compression would be worthwhile to consider. In addition, some CDNs offer compression services, and Cassandra could help Web site administrators weigh the benefits promised by these services.

4 Analysis of CDN Benefits

Content delivery networks (CDNs) deploy caches throughout the Internet to move content closer to the client and decrease client access time. CDNs are also used to decrease hit rate and bandwidth consumption at the Web site. Without Cassandra, Web site operators must decide whether to subscribe to a CDN based on faith or intuition. Cassandra provides the means to make an informed decision.

4.1 CDN Analysis Module

Cassandra currently implements a model of a full-time revalidation CDN, which assumes that all user requests are routed through CDN caches, and uses a standard time-to-live (TTL) approach to decide the validity of cached responses. For expired responses, Cassandra's CDN module simulates `If-Modified-Since` requests.

The CDN analysis module in Cassandra simulates the benefits of using a CDN through trace-driven simulation. Our current implementation uses a packet-level trace gathered using the Gigascope appliance [9]. The trace includes the first packet of every request and response and contains all HTTP header information. In the future, we forsee using a less rich Web access logs as input, augmenting them with Web site probing or short `tcpdump` traces as discussed in Section 3. We assume that CDN caches have unlimited cache capacity given plentiful disk resources.

To map clients to CDN caches, we group clients into clusters using a network-aware clustering tool [18]. Cassandra assigns clients to CDN caches at the granularity of clusters: all clients in the same cluster forward requests through the same cache. Client clusters are randomly assigned to a cache when the cluster's first client generates its first request, and that cache is used throughout the simulation. We do not measure latency effects in this study, hence a different cache assignment will not change our results in any significant way.

The Cassandra CDN analysis module generates estimates of the lower and upper bounds of the potential benefits of using a CDN on a workload. It generates the upper bound by assuming ideal content cacheability: it assumes unlimited lifetimes for cached objects and allows CDN caches to serve all subsequent requests to an object. It generates the lower bound by modeling actual content cacheability and consistency: it simulates the effects of existing cookie and cache-controlling headers in the trace, and considers the trace response sizes in simulating bandwidth consumption. In practice, CDNs will fall between these bounds. The upper bound is idealistic because it assumes that all content is cacheable and never changes. The lower bound is overly conservative in terms of bandwidth consumption because a shared CDN cache can convert some requests (e.g., those with cookies) from unconditional downloads to `If-Modified-Since` requests.

Web sites can benefit from these results in two ways. First, the lower bounds indicate the potential benefits of using a CDN on its current workload. Second, the upper bounds indicate to what extent the site could benefit from improving the cacheability of its content (e.g., by making more informed, targeted use of cookies), and how such changes increase the benefits of using a CDN. Based upon these results, sites can decide whether it is worthwhile to pursue the use of a CDN, or to improve the cacheability of its content.

Due to the highly extensible nature of Cassandra, one can easily add modules for other flavors of CDNs, such as server-driven invalidation CDNs, content push CDNs, or overflow CDNs (which is engaged by a Web site dynamically at peak loads). A discussion of the trade-offs involved among different flavors of CDNs can be found in [11].

4.2 Validation

Eventually, we will validate the simulation models used in the Cassandra tool with real implementations. Our approach is to do this validation on a per-analysis module basis. Here, we explain how we could validate one optimization: CDN caching. We show the validation architecture in Figure 2. The validation setup contains a workload generator driven by requests from the original trace, the server stub, which supplies responses of the proper size and with proper HTTP

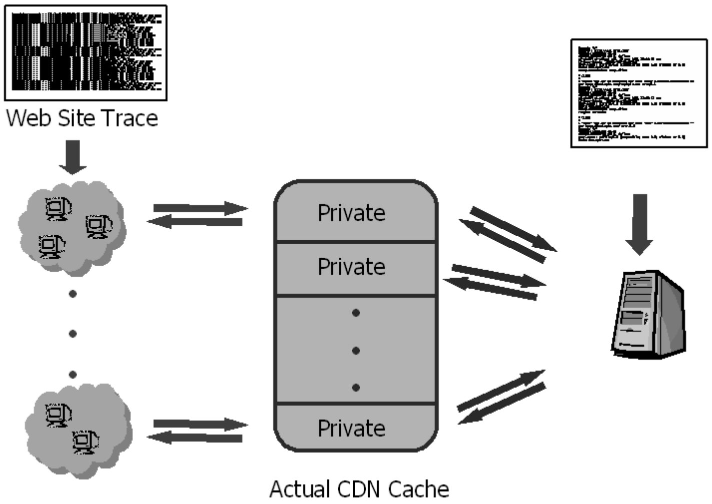

Fig. 2. Validation of Cassandra recommended improvements.

headers taken from the trace, and a real CDN cache in the middle. For the current CDN analysis module we are only interested in validating hit rate and byte rate for CDNs. These two metrics are straightforward to compute based on proxy cache statistics.

The workload generator, or simulated clients, could be any tool which allows replay of the request stream such as Medusa [16], httpperf [23], or wget [30]. The server stub would mirror either the actual response content from the Web site or mimic responses of the appropriate size. These objects would be served with the appropriate headers taken from the trace, modified as necessary to change the cacheability properties of the objects. The server stub would serve these objects using either a standard Web server implementation, such as the Apache HTTP Server [2], modified as necessary, or a custom Web server stub implementation. Finally, the CDN cache would be a real proxy cache implementation, such as the Squid Proxy Cache [26]. For our validation, both the server stub and the CDN cache will operate on separate machines connected by a LAN. The workload generator will also be connected via LAN to the CDN cache, however, depending on the workload requirements, more than one machine may be necessary.

The main challenge in validating a CDN simulation is to model accurately a large number of CDN caches. Clearly, it is impractical to replicate the entire CDN network. Our trick to do this is as follows. Our stub origin server inserts a `Cache-Control:Private` header in every response it sends to the sole CDN cache we are using. This header allows our cache to store and use the responses for repeated requests from the same client, but prevents sharing cached responses across clients. In effect, it compartmentalizes the cache into partitions serving each client separately. Further, we assign each client in the trace to a CDN cache in our target CDN configuration, so we end up with client groups corresponding to each CDN cache. Finally, we associate a "pseudo-client" with each of these client groups. When sending a request to the cache, our workload generator replaces the real client information in the request headers with the pseudo-client associated with the client group to which the real client belongs. Thus, the cache will use cached responses to service all requests from the same pseudo-client. In other words, the cache partitions are done for pseudo-clients, not real ones. Since each pseudo-client aggregates all requests that would have gone to the corresponding CDN cache in the target CDN, this results in exactly the same caching behavior assuming our validation cache has enough cache space.

5 Results of CDN Analysis

To provide a concrete example of using Cassandra, we now use it to evaluate the potential benefits of an optimization on Web sites. In this case study, we use a simulator of CDN caches as the optimization and apply it to a workload of HTTP requests and responses to and from a Web server farm in a large commercial hosting service of a major ISP. We simulate the use of 20 CDN cache nodes based on a known CDN provider's configuration. We focus on the top 100 Web sites in the following analyses because these sites are the ones most impacted by the use of a CDN. This trace workload consists of over 17 million request/response pairs to 3,000 commercial Web sites over the course of 21 hours in July, 2003 [5].

For the experiments in this paper, we make the following modifications to the trace. Our CDN simulations require cacheability information, which is not present in all responses, specifically `304 Not Modified` responses (other responses with no cacheability information are negligible [5]). When a `304 Not Modified` response is the first response to a request for an object, our simulation must discard it, since this response contains no meaningful information for our CDN cache (remember that we are simulating `If-Modified-Since` requests for objects once they are in-cache). Subsequent responses for this same object may contain cacheability information, which we use. This reduces the range of requests to 2.04 million requests for the most popular site and less than 100 requests for the least popular site. The request rates to the 100 most popular Web sites change only slightly.

5.1 Peak Request Rate

We start by examining the impact on the peak request rates of the Web sites when using a CDN. Reducing the peak request load on origin servers is one of the key benefits of using a CDN. Figures 3 and 4 show the peak request rate of the 100 most popular Web sites with and without using a CDN. Figure 3 shows peak request rates using the actual content cacheability characteristics of the requests

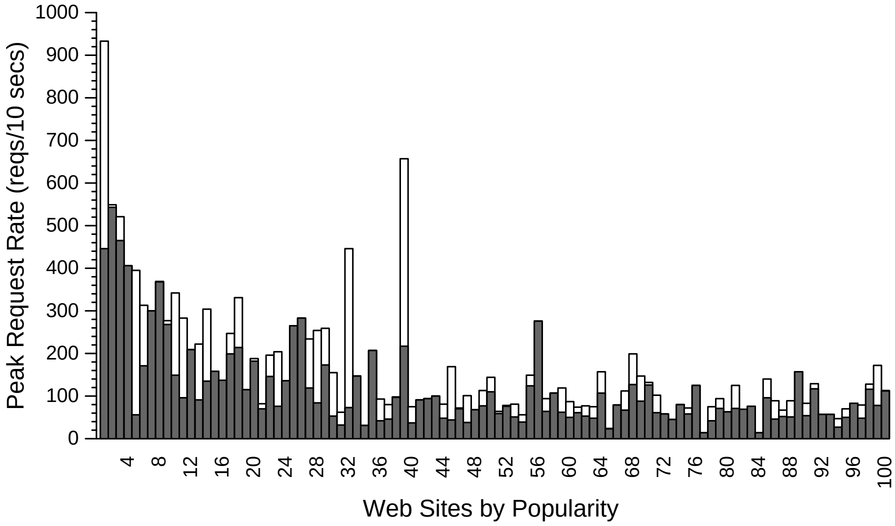

Fig. 3. Comparison of peak request rate with a CDN (dark part of each bar) and the peak request rate without a CDN (total height of each bar). Models content cacheability and consistency.

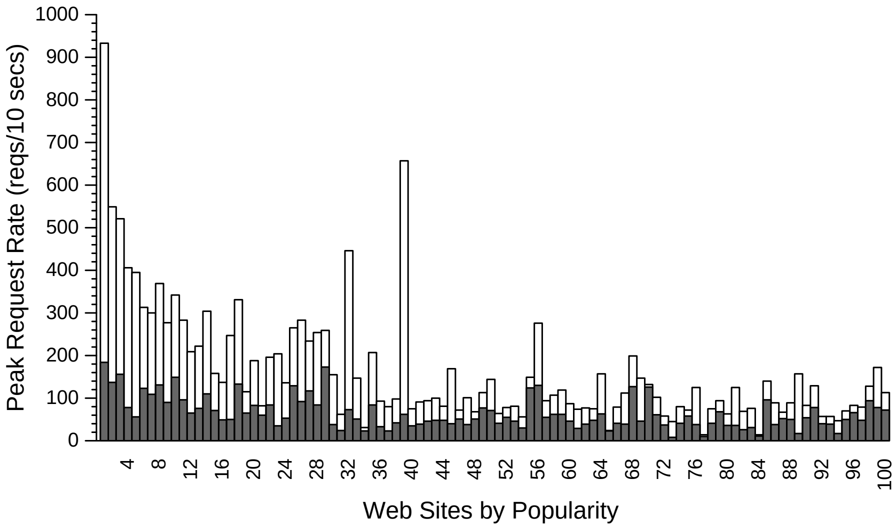

Fig. 4. Comparison of peak request rate with a CDN (dark part of each bar) and the peak request rate without a CDN (total height of each bar). Assumes ideal cacheability. Similar to [5].

and responses in the trace. Figure 4 shows peak request rates assuming content is ideally cacheable: cached objects have unlimited lifetimes, and CDN caches can serve all subsequent requests to an object. Note that Figure 4 repeats the optimistic CDN simulation from [5], but `304 Not Modified` request/response pairs are removed when they are the first access to the object. Figure 3 shows new results from a more advanced simulator that accurately models content cacheability and consistency in detail.

We compute the peak request rate for each Web site at the granularity of 10 seconds across the entire trace. Each bar corresponds to a Web site and the Web sites are shown in order of decreasing request popularity. The entire bar shows the peak request rate for that Web site across our trace without using a CDN. The dark part of each bar corresponds to the peak request rate of that Web site when a CDN is used. The white part of each bar shows the portion of the peak request rate handled by the CDN. It shows directly the benefit of using the CDN for that Web site in terms of the reduction in peak request rate.

Both the maintainers of the Web server farm and individual Web sites can use results such as these to determine the potential benefits of using a content distribution network to serve their content. The Web server farm can determine the effect of using a CDN for all sites it serves. For the 100 most popular sites shown in Figure 3, a CDN would decrease the peak request rate across all sites by a factor of 1.4. If the sites improved the cacheability of their content, in the ideal case a CDN would decrease the peak request rate across all sites by a factor of 2.5, a 79% improvement.

Apart from the Web server farm as a whole, individual sites can use these results to make independent decisions about whether to use a CDN to help deliver their content. Since some sites benefit significantly from using a CDN while others do not, different sites will arrive at different conclusions. For example, Figure 3 shows that the most popular site has its peak request rate more than halved (reduced by 52%) when using a CDN, but the second most popular site achieves little benefit (reduced by 1.1%).

Similarly, each site can compare its more realistic performance with its ideal performance to determine whether (1) it is worthwhile to improve the cacheability of its content, e.g., by making more informed use of cookies, and (2) it can gain additional benefit from using a CDN as a result. For example, by comparing Figures 3 and 4 we see that the second most popular site significantly benefits from improving the cacheability of its content for delivery by a CDN. With current cacheability, a CDN decreases the site's peak request rate by only 1.1%, but by improving the cacheability of its content a CDN could decrease the site's peak request rate by 75% in the ideal case (an improvement of 67%). Of course, the ideal case is an upper bound that may be difficult to achieve completely in practice. Nevertheless, a site can use the difference between the realistic and ideal cases to determine whether it is worth the effort to examine and consider the cacheability of its content. In the case of the second most popular site, it appears worthwhile, while for several other sites (such as 70th and 97th most popular sites) it does not seem to be the case.

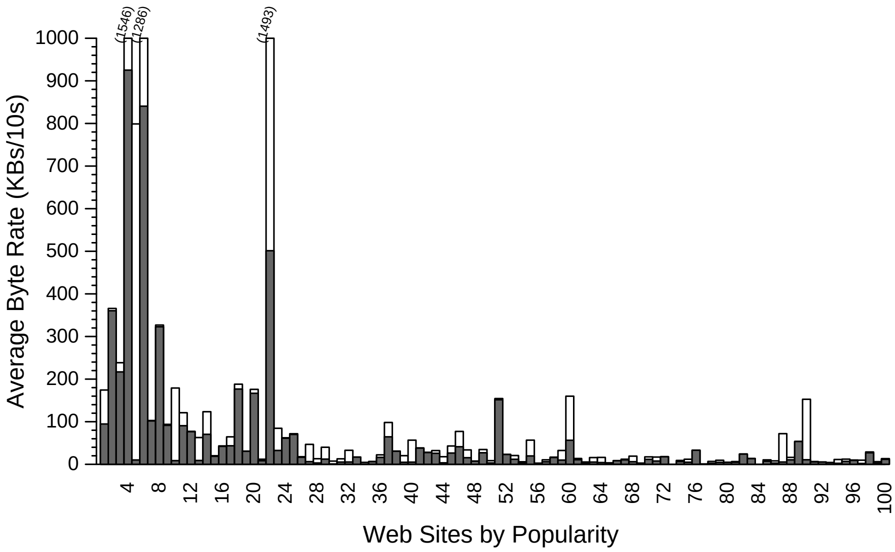

Fig. 5. Comparison of average byte rate with a CDN (the dark part of each bar) and the average byte rate without a CDN (total height of each bar). Models content cacheability and consistency.

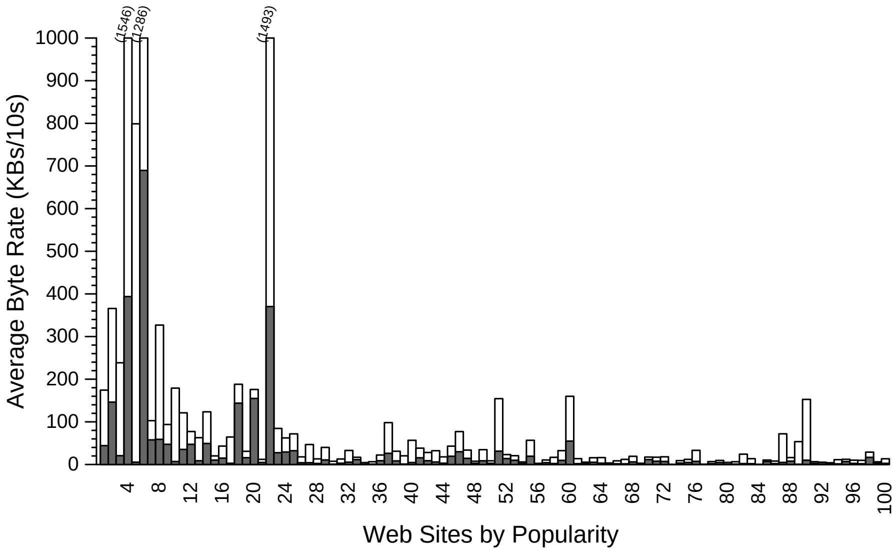

Fig. 6. Comparison of average byte rate with a CDN (the dark part of each bar) and the average byte rate without a CDN (total height of each bar). Assumes ideal cacheability. Similar to [5].

5.2 Average Byte Rate

CDNs also reduce the bandwidth requirements for Web sites, in addition to reducing server load by alleviating peak request rates. Reducing bandwidth can directly reduce costs for both individual Web sites and for Web server farms. To evaluate the impact of CDNs on bandwidth requirements, we study the impact of our simulated CDN on the average byte rate of the Web sites in our trace. Figures 5 and 6 show the average byte rate for the 100 most popular Web sites in our trace with and without a CDN. Figure 5 shows the average byte rate using the actual content cacheability characteristics of the requests and responses in the trace, and Figure 6 assumes content is ideally cacheable. Note that Figure 6 repeats the optimistic CDN simulation from [5] where `304 Not Modified` request/response pairs are removed when they are the first access to the object. Figure 5 shows new results from a more advanced simulator that accurately models content cacheability and consistency in detail. We compute the average byte rate for a Web site by totaling the header and content lengths of all transactions to the site in the trace, then dividing by the trace duration. Each bar corresponds to a Web site. The height shows the average byte rate for the Web site without a CDN. The dark part shows the average byte rate at the Web site when using a CDN, while the white part shows the average byte rate handled by the CDN.

As above, the Web server farm and individual sites can use these results to evaluate the potential bandwidth improvements of using a CDN to deliver content. For the sites shown in Figure 5, a CDN would decrease total server farm bandwidth by a factor of 1.8. These results model the content cacheability and consistency found in the original trace, and represent a lower bound on the benefits of a CDN. In contrast, for the results shown in Figure 6 a CDN would decrease bandwidth by a factor of 3.3. These results model ideal content cacheability, and represent an upper bound. From both graphs, we see that using a CDN for the server farm can significantly reduce bandwidth given its current workload, and that improving the content cacheability of the sites has the potential to make a CDN even more effective for the server farm.

Individual sites can also use these results to determine the impact of using a CDN for their content. For example, as with the peak request rate metric, the second most popular site achieves little benefit in terms of bandwidth on its current workload (Figure 5), but can potentially gain considerable bandwidth improvements by improving the cacheability of its content (Figure 6).

6 Conclusion and Future Work

This paper describes Cassandra, a tool for analyzing the benefits of various performance optimizations to a Web site. The main advantages of Cassandra include extensibility, so that the tool can add new optimization technologies as they become available, the ability to obtain detailed workload characteristics of a Web site, and a user interface to simplify the “what-if” analysis. As a preliminary demonstration of the tool utility, we applied it to analyze the benefits from content delivery networks that a large server farm and individual Web sites hosted on that farm can expect. Web site operators currently have to either rely on intuition in deciding on whether or not to subscribe to CDN services, or to resort to ad-hoc performance analysis, which require considerable expertise and time. Cassandra will make this kind of analysis more straightforward.

Our future work includes extending the analysis module library to incorporate more optimization techniques outlined in Section 3, most notably compression. We also plan to demonstrate the use of Cassandra on a real Web site and on different kinds of workloads. Our ultimate goal is to make Cassandra a valuable tool for Web site operators to deliver Web content in an informed manner.

Acknowledgments

We want to thank Oliver Spatscheck for many helpful discussions and suggestions during this project. We would also like to thank the anonymous reviewers for comments on an early draft of this paper. Bent was supported in part by AT&T sponsorship of the UCSD Center for Networked Systems, and Voelker was supported in part by AFOSR MURI Contract F49620-02-1-0233.

References

1. AlertSite. http://www.alertsite.com/.
2. The Apache HTTP server. http://httpd.apache.org/.
3. G. Banga and P. Druschel. Measuring the capacity of a Web server. In *Proc. of the USENIX Symp. on Internet Technologies and Systems*, 1997.
4. P. Barford and M. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proc. of ACM SIGMETRICS*, 1998.
5. L. Bent, M. Rabinovich, G. M. Voelker, and Z. Xiao. Characterization of a large Web site population with implications for content delivery. In *Proc. of the 13th International World Wide Web Conference*, May 2004.
6. L. Bent and G. M. Voelker. Whole page performance. In *Proc. of the Seventh International Workshop on Web Content Caching and Distribution*, Aug. 2002.
7. Cacheability tools. http://www.web-caching.com/tools.html.
8. Y.-C. Cheng, U. Hoelzle, N. Cardwell, S. Savage, and G. M. Voelker. Monkey see, monkey do: A tool for TCP tracing and replaying. In *Proc. of the USENIX Annual Technical Conference*, June 2004.
9. C. Cranor, T. Johnson, and O. Spatscheck. Gigascope: a stream database for network applications. In *Proc. of ACM SIGMOD*, June 2003.
10. Empirix. http://www.empirix.com/.
11. Z. Fei. A novel approach to managing consistency in content distribution networks. In *Proc. of Web Caching and Content Distribution Workshop*, 2001.
12. R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1 RFC 2616, 1998.
13. A. K. Iyengar, M. S. Squillante, and L. Zhang. Analysis and characterization of large-scale Web server access patterns and performance. *World Wide Web*, 2(1-2):85–100, June 1999.

14. Y. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for CDNs and Web sites. In *Proc. of the 11th International World Wide Web Conference*, May 2002.
15. Keynote. http://www.keynote.com/.
16. M. Koletsou and G. Voelker. The medusa proxy: A tool for exploring user-perceived Web performance. In *Proc. of the 6th International Web Caching Workshop and Content Delivery Workshop*, June 2001.
17. B. Krishnamurthy and M. Arlitt. PRO-COW: Protocol compliance on the Web: A longitudinal study. In *Proc. of the 3rd USENIX Symp. on Internet Technologies and Systems*, pages 109–122, 2001.
18. B. Krishnamurthy and J. Wang. On network-aware clustering of Web clients. In *Proc. of ACM SIGCOMM*, Aug. 2000.
19. B. Krishnamurthy, C. Wills, and Y. Zhang. On the use and performance of content distribution networks. In *Proc. of the First ACM SIGCOMM Internet Measurement Workshop*, pages 169–182, Nov. 2001.
20. B. Krishnamurthy and C. E. Wills. Analyzing factors that influence end-to-end Web performance. *Computer Networks*, 33(1–6):17–32, 2000.
21. S. Manley and M. Seltzer. Web facts and fantasy. In *Proc. of the USENIX Symp. on Internet Technologies and Systems*, pages 125–133, Dec. 1997.
22. J. Mogul. Clarifying the fundamentals of http. In *Proc. of the 11th International World Wide Web Conference*, pages 444–457, May 2002.
23. D. Mosberger and T. Jin. httperf – a tool for measuring Web server performance. In *Proc. of Workshop on Internet Server Performance*, 1998.
24. V. N. Padmanabhan and L. Qiu. The content and access dynamics of a busy Web site: Findings and implications. In *Proc. of ACM SIGCOMM*, Aug. 2000.
25. M. Rabinovich and O. Spatscheck. *Web Caching and Replication.* Addison Wesley, 2002.
26. The Squid Web proxy cache. http://www.squid-cache.org.
27. tcpdump. http://www.tcpdump.org/.
28. Web-Polygraph. http://www.web-polygraph.org/.
29. Websiteoptimization.com. http://www.websiteoptimization.com/services/analyze/.
30. Wget. http://www.gnu.org/software/wget/wget.html.