USENIX Association

Proceedings of the 2nd Workshop on Industrial Experiences with Systems Software

Boston, Massachusetts, USA
December 8, 2002

Using End-User Latency to Manage Internet Infrastructure

J. Bradley Chen and Michael Perkowitz

Appliant, Inc.

Abstract

Performance is a requirement for all interactive applications. For Internet-based distributed applications, the need is even more acute – users have a choice about where they browse, and if a site's performance frustrates them they may never return. The goal of this paper is to demonstrate the power of end-to-end latency measurements of actual site traffic for assuring the performance of Internet applications. We briefly describe a system for collecting true end-to-end latency measurements from real site traffic and describe how such measurements can be used to improve user experience. We give examples to show how to use such measurements to evaluate site performance, pinpoint failures, and elucidate capacity issues. We argue that, as services delivered via the Internet become both more widespread and more complex, accurate measurement of site performance is of vital importance for both maintaining and improving end-user experience.

1. Introduction

Performance is critical to the success of all interactive applications. Although performance is too often neglected as an explicit requirement for distributed applications, performance problems are effectively the same as availability problems – once the user is gone, it does not matter if the system was down or simply too slow. A number of previous publications [1, 3, 5, 10, 13] argue that end-to-end [16] latency as experienced by application users is a key and often neglected measure of application performance. Unfortunately, true end-to-end latency data can be difficult to obtain, and is often no easier to analyze. This paper discusses a system that collects and analyzes data from the users of a Web-based Internet application, for the detection and diagnosis of a broad range of distributed system service problems. We have found that end-user response time information can be collected with no noticeable impact on end-user experience. Given appropriate analysis techniques, the resulting information can be used to detect and diagnose a broad range of problems across the entire content delivery chain. Such performance measurements complement usage information commonly derived from server traffic log file analysis.

Our goal in this paper is build on prior work in the area of latency analysis of interactive applications. We demonstrate the broad utility of end-user response time data in managing a Web-based service. We further describe the challenges in creating a product to track site problems and help managers detect and diagnose end-to-end service problems.

2. Related Work

Approaches to performance monitoring may be divided into two general categories: *robot-based* and *traffic-based*. Robot-based techniques measure the performance of artificial robot-generated traffic as an indicator of overall performance. Because the agent doing the measurement has control over the interaction, precise measurement of various components of the interaction may be made. Additionally, variables such as client operating system, web browser, and type of internet connection are known and controlled. Unfortunately, this advantage is also the weakness of robot-based measurement: because the client and interaction are formulaically controlled, they do not mirror the ever-changing nature of user behavior or the distribution of different user types. Differences among users can affect their experience, and robot-based measurement may miss those differences. A number of popular commercial services collect performance information based on robot data, including the Gomez Performance Network (www.gomeznetworks.com), the Perspective service from Keynote Systems (www.keynote.com), and Topaz Active Watch and Freshwater SiteSeer from Mercury Interactive (www.mercuryinteractive.com). Product offerings based on robot data include Agilent FireHunter (www.agilent.com), BMC SiteAngel (www.bmc.com), RedAlert from Keynote Systems and Freshwater SiteScope from Mercury Interactive.

Traffic-based techniques measure the performance of real user traffic in order to indicate application performance. In this case, the application must generally be instrumented to provide performance data. Traffic-based approaches accurately reflect the wide variation of end users and their differing experiences of the application. However, taking measurements from a wide variety of user types, in real time, without hurting user experience presents a significant technical

challenge. The greatest hurdle is that of successfully instrumenting the client side of the interaction in order to measure true end-to-end latency. Because of the technical hurdles, traffic-based performance management services are less common than robot-based ones. One such service is offered by WebHancer (www.webhancer.com). WebHancer's technique for measuring client-side performance is to provide an executable program that users must install. This gives WebHancer the ability to collect precise and accurate information from the user desktop. The disadvantage of this approach is that users must be persuaded to install the monitor before the site can collect performance information, possibly limiting the breadth of the system's coverage.

Though not in the performance management space, several services provide usage data – information about how users interact with a web site – by monitoring real traffic. These include WebTrends from NetIQ (www.netiq.com) and Hitbox from WebSideStory (www.websidestory.com). These services typically analyze web server logs or use lightweight JavaScript content to record basic user activity. Neither offers end-to-end latency as a metric.

3. Appliant's Approach

In designing a performance management solution, Appliant was convinced of two basic principles:

1. End-to-end latency is the most accurate indicator of user experience.
2. User demographics and behavior affect user experience.

Accordingly, we were driven to make the following demands of our solution:

1. It should measure the real experience of real users.
2. It should measure the experience of as many users as possible.

A robot-based solution was ruled out by the necessity of correlating user demographics with user experience. Similarly, any solution that required users to explicitly install software – an activity many were likely to find irksome or even threatening – was ruled out.

Our approach instead was to annotate web pages with JavaScript code that would perform the necessary measurements. Because the script is loaded with the page, it executes on the client and can record when a page is first downloaded, when images are loaded, and when the page completes. Page annotation requires only that the site modify their pages (typically, modifying only site-wide templates), with no installation required of users. Because JavaScript is a widely adopted internet standard, most browsers support it, ensuring data from most of a site's visitors. In this section we discuss some of the technical hurdles facing this approach and present our methodology in more detail.

3.1. Data Collection

Appliant's browser monitor uses annotations to HTML documents that enable the Web content to monitor itself as it is rendered. The annotation consists of an HTML SCRIPT tag with an include statement that loads Appliant's JavaScript subroutines. The use of an include file simplifies the Web page annotation by shortening the annotation. It also makes it possible for the JavaScript to be cached by the browser. Caching helps minimize Internet Service Provider (ISP) and Content Delivery Network (CDN) fees, and minimizes the overhead for loading JavaScript. The annotation and JavaScript code is part of the page requested by the end user, and is unloaded along with the rest of the page when the browser loads a new document.

The JavaScript code consists of state management routines plus a collection of event handlers that capture various events provided by the Document Object Model (DOM) [6, 18] to recognize and record state transitions in the browser. The Appliant routines install event handlers for the following JavaScript events:

- *onAbort* – loading interrupted
- *onError* – an error occurred while loading an image
- *onLoad* – a document or image has finished loading
- *onReadyStateChange* – the ready state of a document or image has changed
- *onStop* – loading of a web page is stopped by user

Care must be taken to insure that Appliant event handlers pass events through to any handlers previously registered by the page. Yet another problem is handlers that fail to pass events through to the Appliant handlers. This problem is common enough that we support a monitor option that uses a periodic JavaScript timer interrupt to confirm that the Appliant event handlers are registered within the browser.

The browser monitor measures *fetch time* as the time to load the HTML document, and *render time* as the time to fully render that document within the browser

environment. We report render time as beginning when the last byte of the HTML document is delivered to the browser, and ending when the *onload* event fires indicating the page has fully loaded, including the time necessary to load and render any included images. Although this simplification ignores parallelism in the browser it simplifies interpretation of the data and effectively reveals cases where render rather than fetch is responsible for poor response time.

Conceptually, fetch time begins when a hyperlink is selected to be loaded in the browser. In practice, it is not always possible to begin measurement at that instant. In a worst-case scenario, since our browser monitor is not resident in the web browser, the first chance it has to record a time stamp for the beginning of the fetch time is when the top of the HTML document arrives in the browser. In our preferred deployment technique, a cookie is used to retain a timestamp from the unload event of the previous page, which occurs when the user clicks on a link. In this way our monitor can provide very accurate fetch time measurements for consecutive requests from the same managed web site.

Even without using cookies, the fetch time measurement can be very effective. Dynamic web sites typically send a block of static HTML first, allowing the browser to get started while other dynamic components are being computed. If the entire page arrives quickly, the fetch time measurement will be inaccurate in a relative sense, although in an absolute sense the page was very fast and performance is a non-issue. If the page is delayed by slow content generation, the fetch time measurement captures that slowness, and the relative error is small. If the page is slowed due to network delays, our fetch time measurement will reflect that slowness unless the slowness is isolated to connection setup. Overall, we have found that our slight handicap in measuring connection setup time is more than offset by the benefits of complete data, and rarely interferes with our ability to identify problems with end-to-end performance.

There are many opportunities for enhancing browsers to be better sources of performance management data. Currently we are able to capture response time for each image request, but the browser does not expose whether the image was loaded from the browser cache or retrieved via the network. Another handicap is that the precise size of an HTML document is difficult to determine reliably. Such additional data from the browser could substantially improve the quality of performance data we collect.

Once monitoring of a page is complete, a data record is transferred to the data manager using a standard HTTP request. Complete data records are received by the data manager, which generally resides in the same facility as the servers for the managed web site. The data is processed either daily or in real time, depending on the Appliant product. The analysis subsystem generates summaries that support multiple reporting systems. This enables customers to access the information in various forms, including canned reports via email, an interactive Excel workbook, and a Web front-end to a database.

Browser-side filtering of data is available, based on response time thresholds, connection type or other properties of the performance event. The browser monitor also supports statistical sampling. This permits our customers to manage the capacity of their data management system, and greatly simplifies deployment on high-traffic sites.

3.2. Data Analysis

Given the potentially large volume of performance data resulting from end-user monitoring, the next challenge is data collection. Realities such as firewalls oblige us to stick with the well-established standards for Web data transport to the desktop: HTTP/TCP/IP over port 80. Appliant systems use a centralized data collection scheme layered over a simple Web server. Though there are no theoretical problems to solve, there are many practical challenges in delivering a product that reliably handles hundreds or thousands of megabytes of response time data on a daily basis. More specifically, familiar challenges arise in terms of making the system scalable and reliably available. Similar challenges related to high data volume occur in data analysis, but nothing unique to our system. Web log collation and analysis provides a relevant example of strategies for processing similar volumes of data. We apply standard approaches such as cluster-based parallelism and statistical sampling to address these system-level challenges.

An interesting problem that arises in the analysis process is treatment of 'noise' and outliers in the data stream. A small amount of erroneous data is not impossible in distributed systems that handle millions of records per day over WAN TCP/IP connections [17]. Data inaccuracies can also occur due to phenomena such as defects in browser implementations, local clock adjustments that occur during a measurement, and connections that operate at effective speeds of tens of bytes per second. A more fundamental problem is that wide-area network response time data commonly

exhibits an asymmetric heavy-tailed distribution [14], for which the mean is not a robust statistic. For such non-normal distributions, a trimmed-mean is a common alternative measure that provides a more robust measure of central tendency than the mean. [8, 12] A trimmed mean is computed by excluding the extreme n% of data points, where a common value for n is 5%. Although this would provide a robust mean, identifying the 5th percentile is computationally equivalent to a median computation, which requires *O(n)* space, too expensive for the large data sets handled by our system. Instead we simply exclude values that exceed a predefined "trim" threshold, and then confirm that the amount of data trimmed was less than 5%. By default the system trims data points with response time exceeding 180 seconds. To date, simple strategies such as these have been adequate to cope with the noise inherent in end-user data.

Clock skew between the web server, browser client, and data collector is a potential problem that was avoided through the Appliant system design. A reality of the Internet and the diversity of servers it hosts is that clock skew is a common phenomenon – we routinely encounter clocks skewed by hours or days even on high-traffic, professionally managed sites. In recognition of this reality we were deliberate in creating a design that tolerates skewed clocks. The basic strategy is that timestamps are assigned only on a small number of dedicated servers within the host data-center. Desktop machines do generate latency measurements, but they do not assign time stamps. Clock skew is not an issue for latency measurements since both the start and stop time are measured relative to the same clock.

4. Analysis Examples

4.1. ISP and CDN Performance Profiling

Performance is central to the value proposition for ISP and CDN services. Without good performance data, it can be difficult for CDNs to justify the value of their services and for customers to know that they are getting what they paid for. The collection and analysis of end-user latency data makes it possible to obtain ISP and CDN performance information of exceptional detail. Figure 1 gives an example for an Internet community site, showing performance improvements during the first two weeks of CDN service for an example Web site. In this case, the potential CDN customer was skeptical about the CDN's ability to improve performance for dialup customers. The CDN used this information to demonstrate that performance improvements of 32-41% were achieved for the customer's site across all classes of end-users.

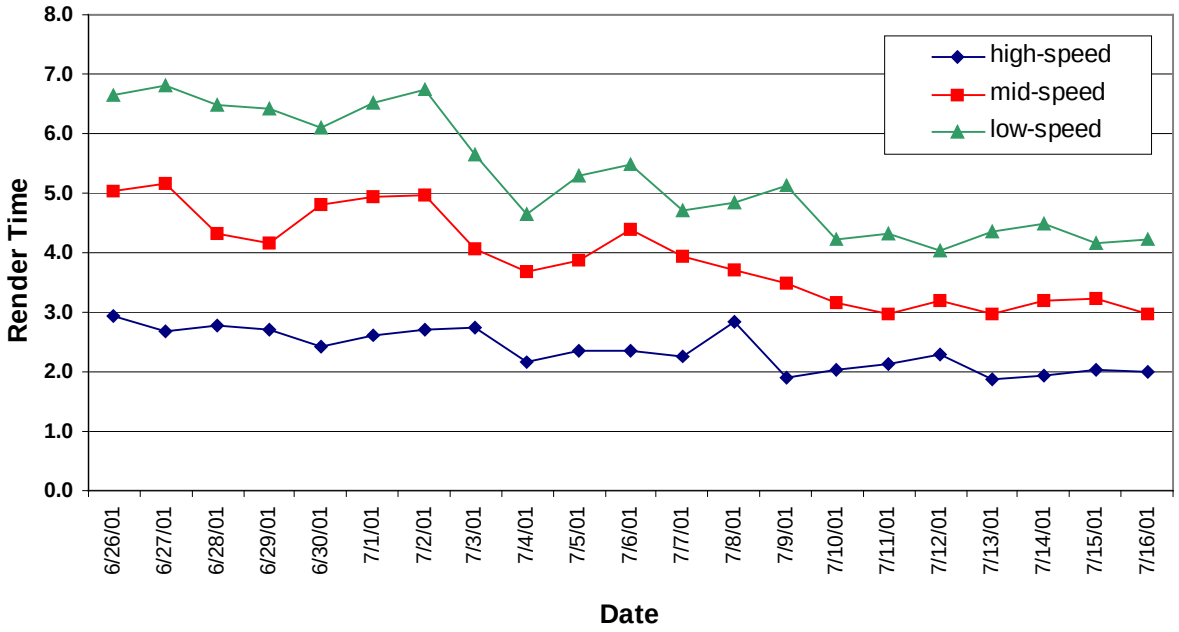

Figure 1. Documenting CDN Performance Improvements. CDN service began on 28 June 2001.

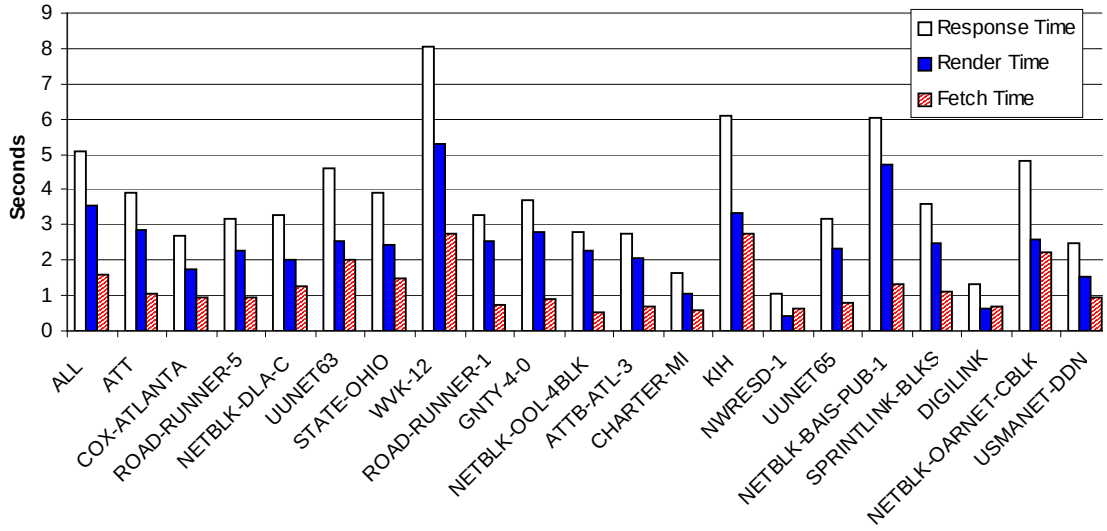

Figure 2. Response Time for Busiest ARIN Netblocks, US LAN traffic only.

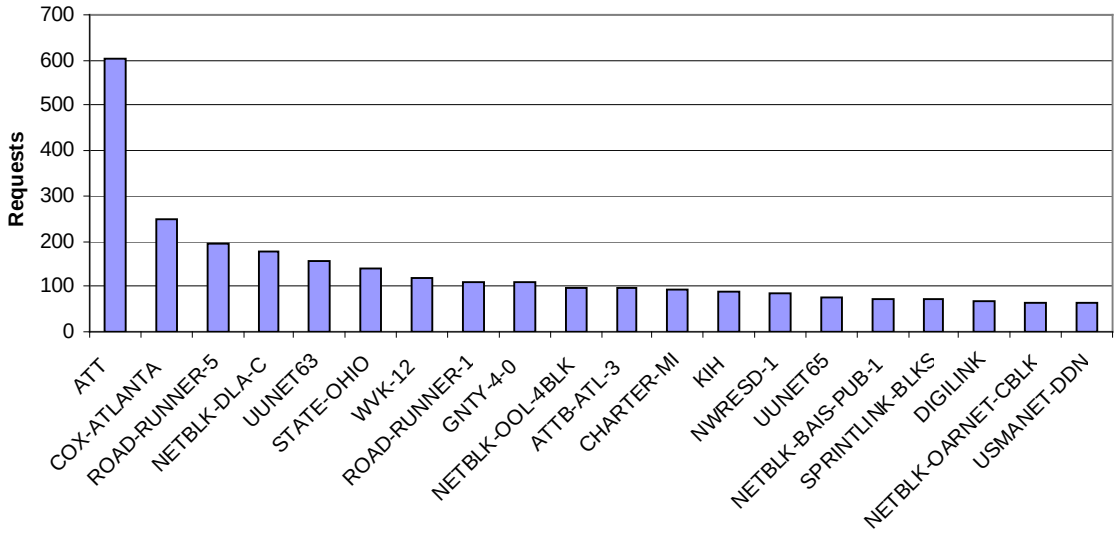

Figure 3. Traffic for Busiest ARIN Netblocks. US LAN traffic only.

Even when ISPs and CDNs deliver good performance, they do not provide equal benefit to all Internet users. Performance data from end-users can be used to quantify site performance all the way to the desktop, thereby identifying network subnets that get the best and the worst service. Figure 2 shows response time in terms of fetch time and render time for an Internet sports site, for the busiest IP address ranges as defined by ARIN *whois* data[1]. Figure 2 shows US LAN-traffic only[2], as desktop connection type has a significant impact on end-to-end response time. The figure shows below-average performance for netblocks maintained by the West Virginia K-12 system (WVK-12), Kentucky Department of Information Systems, and Bell Atlantic (now administered by Verizon). Together with Figure 3, showing traffic levels for the same address

[1] See ftp.arin.net and www.arin.net for more information.

[2] Connection type is obtained using the "clientCaps" facility in Microsoft Internet Explorer. For more information see http://msdn.microsoft.com/workshop/author/behaviors/reference/behaviors/clientcaps.asp.

ranges, this information enables a web site to identify the Internet subnets that are most important in terms of traffic volume, and compare their performance to the site average (given by the ALL category). This information can indicate situations where poor ISP peering relationships or poorly performing CDN caches are causing performance problems for specific end-users.

Although these figures are useful for a high-level analysis, additional detail is generally required to validate problems and identify a cause. A key observation is that the blocks of IP addresses defined by whois data vary in size, and some are quite large. For a finer-grain analysis, we commonly use a mapping based on 24-bit prefixes. Table 1 shows WVK-12 subnets, ordered by their contribution to the overall WVK-12 mean response time. These subnets include the slowest WVK-12 subnet (168.216.124), and the fastest (168.216.107). Assuming that the WVK-12 subnets share the same path(s) to the Internet, the variance in performance across these subnets implies that performance problems are internal to WVK-12, as their fastest subnets (and, by implication, all external Internet factors) have satisfactory behavior.

Independent	Requests	Subnet RT	delta Mean
168.216.51	33	6.45	1.84
168.216.161	3	28.83	0.75
168.216.141	7	11.48	0.69
168.216.123	2	33.65	0.58
168.216.67	4	13.90	0.48
168.216.107	17	3.14	0.46
168.216.94	10	5.30	0.46
168.216.56	6	8.07	0.42
168.216.48	2	20.90	0.36
168.216.124	1	38.12	0.33

Table 1. WVK-12 Subnets with largest contribution to Mean.

As a final example of finer-grained data, Table 2 shows response time for the slowest 24-bit subnets with at least 0.1% of total site traffic. We note that "whois" data can often identify these subnets with surprising precision. In this table, 216.236.222 is a satellite services provider, and 134.134.248 is an Intel facility in Santa Clara, CA. Making effective use of this level of detail remains a challenge. Specific challenges include:

- Achieving a classification of appropriate granularity
- Isolating ISPs such as AOL and NewSkies that skew results due to atypical network properties
- Distinguishing "last-mile" problems from "middle-mile" problems

Regrettably, due to these challenges, solving network problems remains in the domain of "networking experts." The goal of our continuing work is to further automate the process of recognizing performance incidents and identifying them to their authentic source.

Subnet		*n*	RT
216.236.222	NEW SKIES SATELLITES N.V.	15	47.0
207.108.252	U S WEST INTERNET SERVICES	17	23.5
134.134.248	INTEL CORPORATION	20	21.2
206.129.0	N2H2	12	14.3
209.133.187	STATE OF SOUTH CAROLINA	14	10.2
170.158.130	ONONDAGA BOCES	16	9.7
12.96.122	SHASTA COUNTY OFFICE OF EDU.	21	9.7
204.171.48	VERIO	14	9.6
198.110.59	REGIONAL EDU. MEDIA CENTER 4 (MI)	16	9.3
24.187.188	OPTIMUM ONLINE (CABLEVISION)	14	9.1
216.81.96	ALMA TELEPHONE	12	8.6
209.124.103	AMNET US	23	8.5
205.154.229	GROSSMONT UNION H.S. DISTRICT	23	7.7
207.70.63	MICRON INTERNET SERVICES	18	7.5
206.78.5	VISALIA UNIFIED	14	7.1
150.176.63	FLORIDA INFO. RESOURCES NET	26	6.8
65.204.164	BBG COMMUNICATIONS	32	6.6
209.205.203	PACNET	26	6.6
216.229.196	MISSISSIPPI DEPT. OF EDUCATION	19	6.5
168.216.51	WV DEPARTMENT OF EDUCATION	33	6.5

Table 2. Performance Detail by Class-C subnet. US LAN traffic only, for subnets with at least 0.1% site traffic.

4.2. Capacity Planning

To effectively plan site capacity, a site administrator must be able to answer a number of questions:

- What is the capacity of the system?
- What is the offered load on the system?
- What resource or resources are the bottlenecks?

Examples of potential bottlenecks include CPU cycles, physical memory, memory bandwidth, IO bandwidth, and LAN bandwidth. Bottlenecks can occur anywhere in the system: Web servers, application servers, database servers, or the network. When peak demand exceeds system capacity, the result is performance problems, system failures, or both.

End-user performance data is a very effective way to answer these questions. Figure 4 shows a chart of page requests by hour over a day for an example web site. It shows a common pattern of traffic, with a peak in traffic in the daytime and a corresponding trough at night. The response time chart (Figure 5) and page request charts have different shapes, indicating that

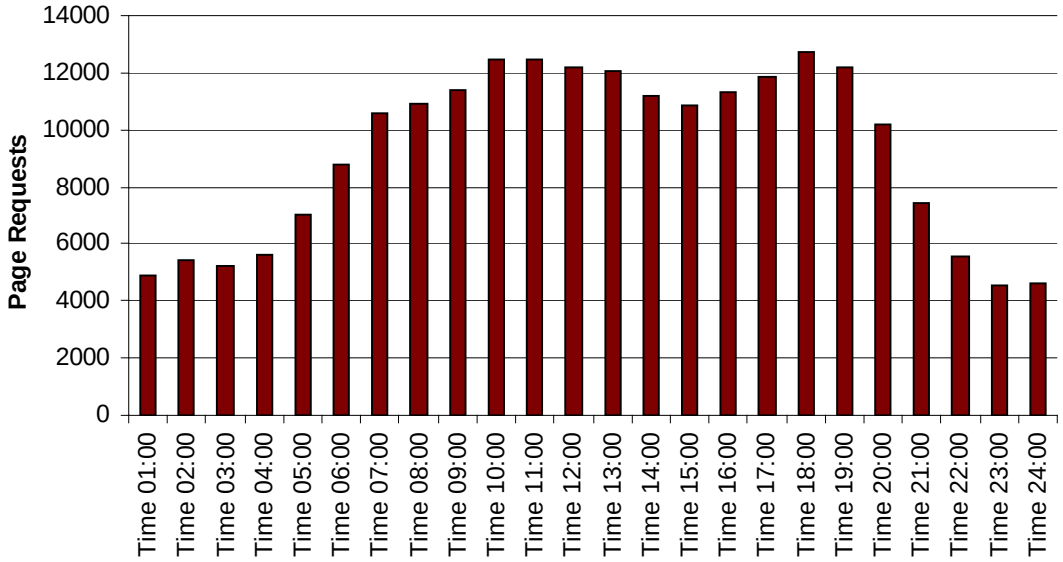

Figure 4. Page Requests vs. Time over 24 Hours

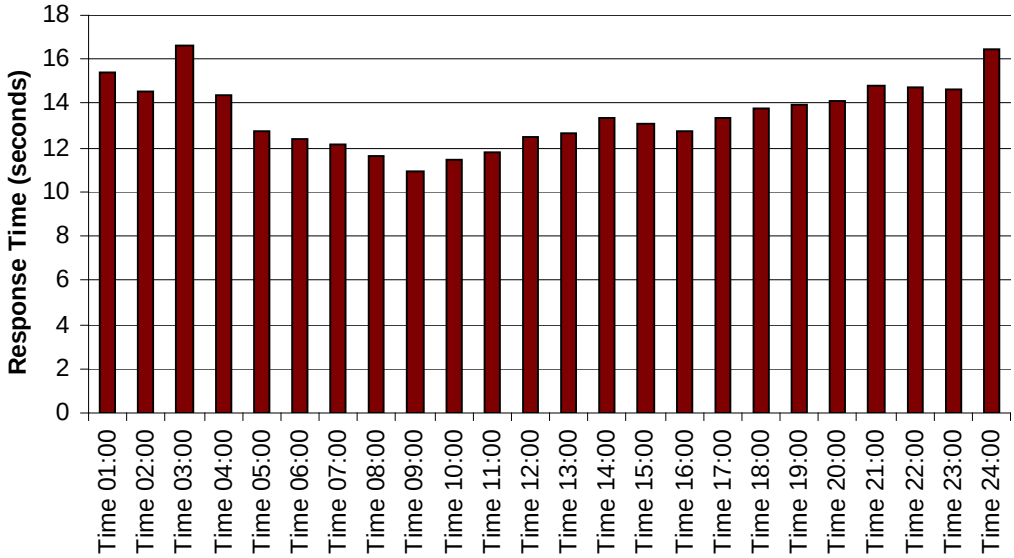

Figure 5. Site Load vs. Performance over 24 Hours

there is not a direct relationship between response time and load. In this case the response time curve shows a common pattern, with a faster average response time when many users access the site from their fast work connections, and slower traffic when more people use dialups at home. If response time tracked page requests, it would indicate a direct relationship between performance and load, evidence that an internal performance bottleneck was being stressed. The independence of response time and load in these charts indicates that the site has excess capacity, even at peak hours. If it were the case that response time tracked page requests, we could identify the bottleneck by examining additional resource charts to discover where resource utilization is correlated with performance problems.

4.3. A Partial Server Failure

Availability management tools and robot-based performance monitors can be very effective at identifying complete site failures. They simply test the site periodically and report as appropriate when a server or the whole site appears to be offline. The problem becomes more difficult, however, when a large and complex content system needs to be tested, or when server clusters are hidden behind a single IP address. When a site becomes sufficiently mature to have performance expectations as well as availability, end-user data can identify a much broader class of site problems.

As an example, Figure 6 illustrates response time by server for a cluster of servers over a 24-hour period for a web-based news site. At 6:00 AM, for example, most servers are delivering similar performance of about 6 seconds average response time, but one server (server 5) has response time of about two seconds slower. Across the 24-hour day, the same server was consistently about two seconds slower. Such problems can be extremely difficult to detect with robot-based data sources.

Considering request rates for the same time period, Figure 7 shows two kinds of servers. The bottom server (server 1) is a four-way multiprocessor. It receives about half as much traffic as the eight-way machines along the higher curve. The surprise is that the top server is the same eight-way system (server 5) that had response time problems in Figure 6. The load balancer is actually directing about 20% more traffic to the slowest server in the cluster. Load balancers generally use server-side metric such as CPU utilization to make decisions about distribution of load. In this case those decisions were wrong. Ultimately the explanation for these behaviors was that the deviant server was using beta software. Such behavior could be caused by throughput optimizations, for example, delaying requests for a brief period so-as to apply a scheduling optimization to a group or requests.

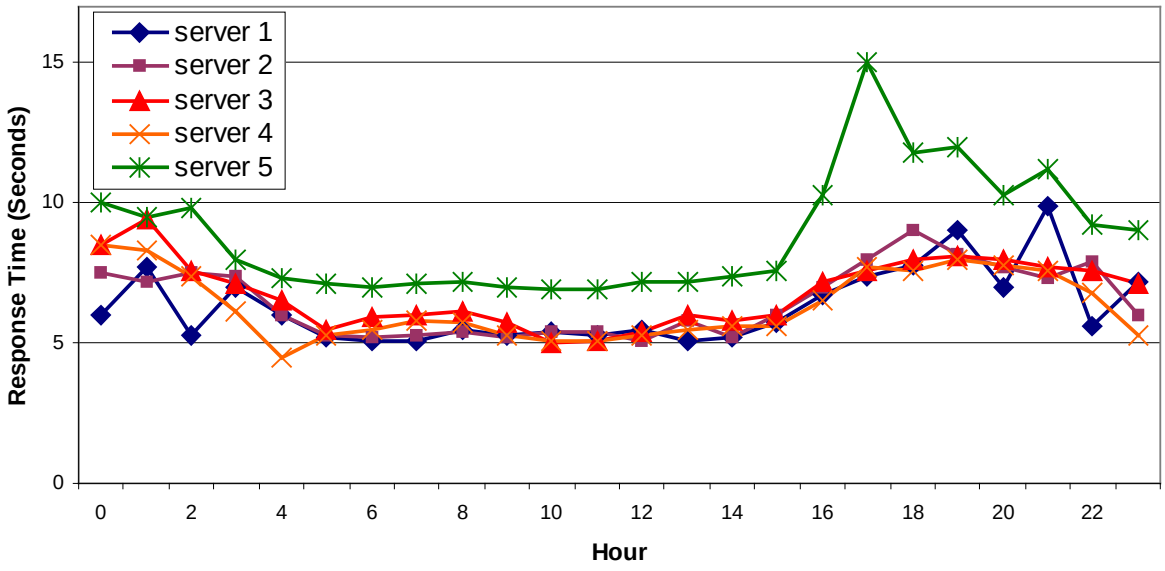

Figure 6. Response Time by Server over 24 Hours

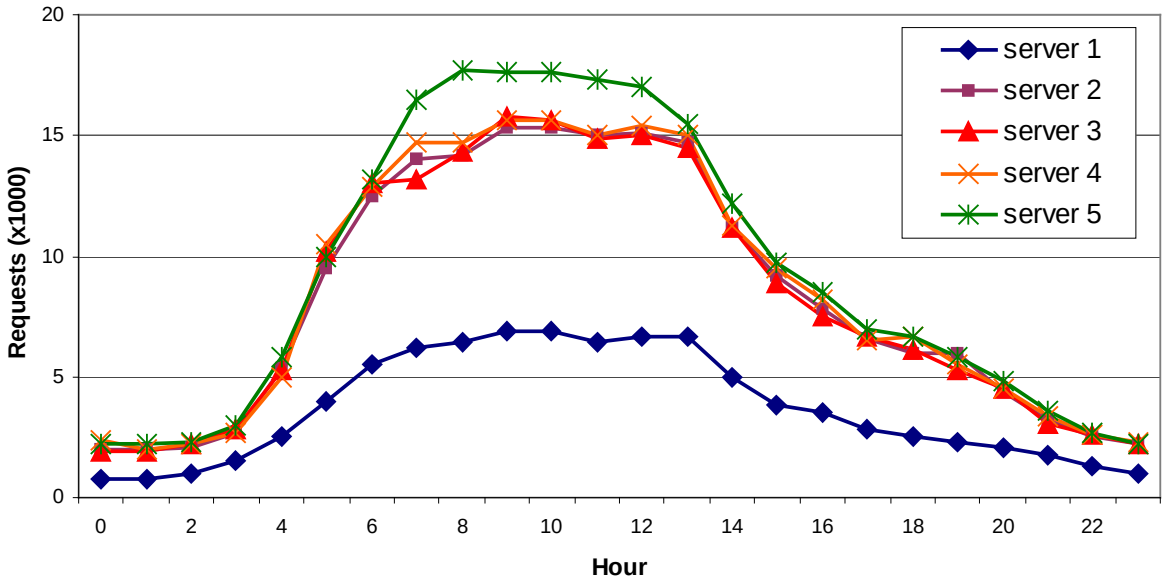

Figure 7. Page Requests by Server over 24 hours

5. Discussion

The direct presentation of our technical results obscures many of the challenges behind the development of the software we describe in this paper. Indeed this system had its share of challenges. In this section we document some of the technically relevant ones.

Although the results in this paper relied exclusively on data collected via JavaScript content annotations, considerable effort went into alternative approaches to data collection that we ultimately determined were not suitable for our company or products. We briefly considered building a network packet sniffer, such as that used by Wolman et al. [19]. Although this approach has many nice properties in terms of completeness and accuracy, a number of practical considerations caused us to exclude it early on, in particular:

- Business requirement of a software (not hardware) product
- Lack of in-house expertise
- Inability of sniffers to manage 3rd party content and desktop errors
- Anticipated performance and scalability issues

These factors led us to exclude a packet monitor early in our design process.

In addition to the JavaScript-based browser monitor, early Appliant web-management systems also used a server-side software component to measure application-level response time plus system-level performance metrics (such as CPU and memory statistics) from the server perspective. The anticipated benefits included a more precise measurement of server-side latency for content generation, plus additional information for diagnosing problems within a server. Initially we found this component a challenge to support, although the support burden became easily manageable after the bumps of early releases. Ultimately we decided to remove this monitor as a feature for the following reasons:

- Considerable cost for developing and testing a software component on a large matrix of OS/Web Server platforms
- Customer resistance against installing an executable software component on their production systems
- Marginal data value, as response time is available from desktop, and system-level metrics are provided by management frameworks.

As we describe in Section 4.3, powerful server diagnosis is possible without a server-side software install.

Another option we considered and then dismissed is a client-side installation of executable code. The potential benefits of such a configuration are significant. The stability and richness of the OS APIs could provide more precise measurements, a richer set of metrics, and more complete measurements as well. Many of these

benefits derive from the fact that an executable component avoids the security and privacy restrictions of the browser environment. Ultimately we made little use of this approach for the following reasons:

- Considerable cost for developing and testing a software component on an exceptionally large matrix of OS/Web browser platforms
- Customer resistance against installing executable software components on desktops
- Additional metrics are of marginal incremental value over data already available from JavaScript monitor
- Use of a desktop executable raises many new security and privacy issues

As compelling as the results we have achieved may be, it can fairly be said that the hardest problems have not been solved. Some of the key challenges we are considering in our ongoing work include:

- Reliably recognize abnormal behavior
- Reliably ignoring non-problems
- Classifying problems by impact and significance
- Combine multiple performance measurements into a single indication of the root cause

5.1. Combining Performance and Other Data

Application performance has a direct impact on user satisfaction, but performance data alone is not sufficient to show this relationship. By correlating performance data with other information illustrative of customer satisfaction or bottom-line success, we can determine how tolerant users are and how important performance improvements will be. Useful additional data can include user behavior or sales figures. For example, we may wish to find the correlation between average performance and user session length; do users who experience poor performance spend less time on the site? Another option would be to examine user behavior immediately following a slow page-load; do users give up and depart when a page takes a long time to load? A third possibility would be to look at the correlation between performance and conversion rate – the probability that a user will make a purchase.

Such analysis is, of course, useful in justifying the importance of good application performance, but it can also be used to prioritize infrastructure improvements. Users tend to be more tolerant of some problems than others, and limited infrastructure dollars can be channeled to the most irksome problems. For example, poor performance on catalog pages may discourage browsers from ever becoming buyers, but once a customer has committed to the checkout process, he is less likely to abandon his session because of poor performance. In this case, catalog infrastructure should receive priority attention.

5.2. Social/Economic Challenges

There are many practical problems that have substantial impact on our ability to deploy our solution and on the feature set we are able to support. A few of the most common problems are noted here.

Our solutions give strong Internet businesses an opportunity to tune and focus the effectiveness of their site. Just as high-end retailers use customer experience to differentiate themselves from discounters, we believe high-end web sites will use customer experience to improve the experience for their users, and thereby improve their bottom line. Unfortunately, many Web sites are not ready for this level of service delivery. They are overwhelmed with the struggle to keep systems running, manage content updates, while coping with tight and shrinking budgets. These problems are exacerbated by software churn, which tends to prevent systems from reaching maturity. As a result, many online services are unwilling to monitor or assure end-to-end service levels.

Another social challenge relates to privacy issues. At a technical level we believe our position on privacy is very strong. We assume that the managed Web system generates or is able to generate log files for site usage analysis. Given this assumption Appliant systems do not change the balance of privacy for end users, because they collect no additional information on behavior. They do augment existing data with performance and error information for diagnostic purposes. All data is co-located with the managed web site and under the control of the site administrator. Since the site administrator already has access to complete usage data for their site, the privacy balance remains unchanged. The product does not require cookies, although it can make use of cookies that are already used by the site. The product does not support or require the collection of zip codes, address information or account numbers of any kind.

Although our position is quite defensible at a technical level, prospective customers are frequently unable or unwilling to consider the technical details. Frequently

they are obliged to take a very conservative position on privacy: that any additional data collection is unacceptable, regardless of the fact that additional data is of diagnostic value only and does not reveal information about individual users. For these sites, an opt-out program can be used to achieve an acceptable solution.

5.3. Prior Research in Latency

In the research community, the argument for use of latency as a measure of system performance emerged from prior concerns about the effectiveness of micro-benchmarks and throughput benchmarks for measuring system performance [1, 10, 11, 13]. Endo et al. made a direct argument advocating such techniques [5]. Whereas the Endo study was limited to interactive applications on a standalone desktop, our interest is in interactive distributed applications on the Internet. Since that publication, relatively little work has been done on the subject, with the bulk of the work occurring in the context of real-time but non-interactive applications such as streaming media. Flautner et al. [7] included latency-based analyses to show that real-time apps benefit less from threading on a multiprocessor than on a uniprocessor (15% vs. 4%). Jones and Regehr [9] used latency measurements to analyze thread-scheduling issues in support of real-time applications.

We note two prior publications that study latency in the context of the Web. Wolman et al. [19] studied Web end-user latency in the context of cooperative Web proxy cache performance. Ramakrishnan and Elnozahy [15] describe a system for collecting response time on end-user browsers that has some similarities to ours, particularly in the raw data that is collected. Although this paper clearly predates the present work, we believe that our systems were developed and released as products prior to theirs. The Ramakrishnan paper includes an ample discussion of the data collection methodology on the client, along with one example of analysis for a small Internet Web site. The present paper builds on the Ramakrishnan publication by documenting challenges in working with large production Web sites, in contrast to the relatively small site used by Ramakrishnan. Further, our experience in working with response time data allows us to explore in more detail the exceptional diagnostic power of end-user information.

6. Conclusions

The performance of web sites and Internet applications varies widely and is often the main factor in determining the quality of the end-user experience. End-to-end response time is a key measure of performance, and we believe it to be a vital differentiator for end users. Response time measurements, moreover, are an important potential resource for site development and operations. In this paper we have demonstrated how such information, derived from real traffic, can be used to systematically analyze situations that are very difficult to detect or diagnose without such measurements.

The systems described in this paper anticipate a day when Web browsers and other end-points for Internet applications expose detailed, accurate service quality metrics through stable APIs. Ideally such measurements would include end-to-end application-level performance measurements, error detection, and failure reporting. Although Appliant systems can provide this information for the current Web infrastructure, the lack of stable, consistent management APIs complicates the implementation. We believe that the importance of service level monitoring systems will increase as Web services mature. In this context, outsourced infrastructure and integration with third-party services becomes more common, and their performance becomes a requirement and a success factor. This is in contrast to the status quo, where distributed application performance is often considered only after deployment.

Acknowledgements

We would like to acknowledge the hard work and dedication of the Appliant product and development team. The results described in this paper are a direct result of their commitment to fidelity in distributed application performance management. We would also like to thank Craig Partridge and the WIESS Program Committee for their comments on preliminary drafts of this paper.

Author Contact Information

Please note preferred email addresses for the authors as follows:

J. Bradley Chen: brad.chen@acm.org

Mike Perkowitz: mike@perkowitz.net

7. References

1. Brian N. Bershad, Richard P. Draves, and Alessandro Forin, "Using Microbenchmarks to Evaluate System Performance." *Proceedings of the Third Workshop on Workstation Operating Systems*, IEEE Computer Society, Los Alamitos CA, April 1992.
2. J. Bradley Chen, "SLA Promises," Appliant Technical Note, June 2002. Available from http://www.appliant.com/techresource/techresource.php.
3. Erik Cota-Robles and James P. Held, "A Performance of Windows Driver Model Latency Performance on Windows NT and Windows 98," *Third Symposium on Operating System Design and Implementation*, USENIX Association, Berkeley CA, February 1999.
4. Kenneth Duda and David Cheriton, "Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose schedule." *Seventeenth ACM Symposium on Operating System Principles*, ACM, New York NY, December 1999.
5. Yasuhiro Endo, Zheng Wang, J. Bradley Chen, and Margo Seltzer, "Using Latency to Evaluate Interactive System Performance." *Second Symposium on Operating System Design and Implementation*, USENIX Association, Berkeley, CA, October 1996.
6. David Flanagan, "JavaScript, The Definitive Guide, Third Edition." O'Reilly and Associates, Sebastopol, CA, 1998.
7. Kriztian Flautner, Rich Uhlig, Steve Reinhardt, and Trevor Mudge, "Thread-Level Parallelism and Interactive Performance of Desktop Applications," *Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, New York NY, November 2000.
8. David C. Hoaglin, Frederick Mosteller, John W. Tukey, *Understanding Robust and Exploratory Data Analysis*. John Wiley and Sons, Hoboken NJ, 2000.
9. Mike Jones and John Regehr, "The Problems You're Having May Not Be the Problems You Think You're Having." *The Seventh Symposium on Hot Topics in Operating Systems*, IEEE Computer Society, Los Alamitos, CA, March 1999.
10. Jeffrey C. Mogul, "SPECmarks are leading us astray." *Proceedings of the Third Workshop on Workstation Operating Systems*, IEEE Computer Society, Los Alamitos CA, April 1992.
11. Jeffrey C. Mogul, "Brittle metrics in operating systems research." *The Seventh Symposium on Hot Topics in Operating Systems*, IEEE Computer Society, Los Alamitos, CA, March 1999.
12. NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/, September 2002.
13. John Ousterhaut, "Why Operating Systems Aren't Getting Faster As Fast As Hardware." *Proceedings of the Summer 1991 USENIX Conference*, USENIX Association, Berkeley CA, June 1991.
14. Vern Paxson and Sally Floyd, "Wide Area Traffic: The Failure of Poisson Modeling." *IEEE/ACM Transactions on Networking*, Volume 3, Number 3, 1995, pg 226-244.
15. Ramakrishnan Rajamony and Mootaz Elnozahy, "Measuring Client-Perceived Response Times on the WWW." *USENIX Symposium on Internet Technology and Systems*, USENIX Association, Berkeley, CA, March 2001.
16. Jerome H. Saltzer, David P. Reed, and David D. Clark, "End-to-End Arguments in System Design," *ACM Transactions in Computer Systems*, Volume 2, Number 4. ACM, New York NY, November 1984
17. Jonathan Stone, Michael Greenwald, Craig Partridge, and James Hughes, "Performance of Checksums and CRCs over Real Data." *IEEE/ACM Transactions on Networking*, Volume 6, Number 5, ACM, New York NY, Oct 1998.
18. World Wide Web Consortium, "Document Object Model (DOM) Level 2 HTML Specification, Version 1.0." Candidate Recommendation, June 2002. Available from http://www.w3.org/TR/2002/CR-DOM-Level-2-HTML-20020605
19. Alec Wolman, Geoff Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry M. Levy, "On the Scale and Performance of Cooperative Web Proxy Caching." *Seventeenth ACM Symposium on Operating System Principles*, ACM, New York NY, December 1999.