PERFORMANCE OPTIMIZATION FOR ENTERPRISE WEB APPLICATIONS THROUGH REMOTE CLIENT SIMULATION

Tomas Cerny[1] **and Michael J. Donahoo**[2]

[1] Department of Computer Science and Engineering, Czech Technical University,
Charles Square 13, 121 35 Prague 2, CZ
[2] Department of Computer Science, Baylor University,
P.O. Box 97356, 76798-7356 Waco, TX, US

tomas.cerny@fel.cvut.cz(Tomas Cerny)

Abstract

Contemporary enterprise web applications provide complex user interface functionality to attract users with desktop-like features. Naturally, such increasing UI complexity and user expectation results in greater application resource demands, which may degrade performance. System performance is measured by application responsiveness for end users in service delivery. Web application performance optimization occurs at all levels of the application, from server hardware, to database, etc. In this paper, we focus on the network delivery of application assets, identify bottlenecks in the service delivery, and provide suggestions for optimization. To accomplish this, we provide tools to simulate a variety of remote user communication scenarios and identify the application assets with the greatest impact on application performance. Based on the simulation results for the production application, we provide optimization options, which may be iteratively applied and simulated until the desired application performance is achieved.

Keywords: Web page load time optimizations, Performance evaluation, Network emulation, Remote client simulation.

Presenting Author's Biography

Tomas Cerny. Research in networking and simulation, specializing in web application optimization with a recent publication at MOSIS 2010. Other research interests include software engineering and model-driven development with publications at ICISA 2010, TechEd 2010 and ISD 2010.

1 Introduction

Enterprise web applications have experienced significant expansion in the last decade [1]. More and more web users expect modern applications to provide a rich user experience, typically implemented with multiple technologies such as AJAX, AHAH, JavaScript, Flash, etc. While rich applications provide a better user experience, performance can be negatively affected because the server-side needs to perform more processing and provide more data per application request. Currently if we want to design an application, we most likely utilize technologies that involve rich-client experience, but we need to be aware of its performance. Web applications typically utilize the Hypertext Transfer Protocol (HTTP) protocol [2] [3] [4] for the data transfer. This protocol is built on top of Transmission Control Protocol (TCP) in a layered architecture, which may also contribute to lower performance [5] when using HTTP.

In this work, we look at optimization of communication between an application server and a client connected via HTTP. We investigate factors that impact application responsiveness for end users in service delivery. We describe the elements of TCP directly impacting performance, which provides a motivation for optimization improvements. Since web applications provide service to a wide range of clients, we develop a simulation tool that allows specification of network conditions and remote user emulation. Next we conduct a network simulation study with optimization evaluation of an enterprise web application. Through iterative simulation and revision, we demonstrate improvement in the end user's performance. The optimizations were ultimately applied to a production application.

This paper is organized as follows. In Section 2, we discuss the difficulties with the underlying technologies and recommend application resource configuration. We provide a simulation case study and apply the simulation results to a production application in Section 3. Related work is discussed in Section 4. Finally, we end with a summary of our contributions and conclusions.

2 Underlying Technologies

In this section, we describe the technologies underlying the typical web application and characterize how they impact end user service delivery. Consider a user browsing a news server such as *cnn.com*. The user navigates his web browser to the URL *cnn.com*. In the background, the web browser translates the URL *cnn.com* into an IP address (e.g., 157.166.255.19) by DNS lookup and initiates a connection to the *cnn.com* IP address on server port 80. The server then sends an HTTP request for the default or index page. The HTTP request is wrapped with the TCP header (20 bytes) which is extended with an IP header (20 bytes), link frame header and passed to the network[2][3]. For our work, we focus on only HTTP and TCP. TCP initializes the connection with a 3-way handshake where the two endpoints negotiate their connection parameters before they start communication[3]. Here the client sends a SYN segment to the server (*cnn.com*), the server replies with SYN-ACK segment, and the client replies with an ACK segment. Then the connection is established and we can send the actual data, which still includes the TCP header overhead. The connection is closed in a similar way to its initiation where both endpoints must send a FIN and receive an acknowledgement to complete.

From the description above, we see that to receive one resource from *cnn.com* over HTTP actually sends multiple TCP segments to open and close each connection. This overhead can add significant cost to the transport of each resource, especially for small resources. If there is a significant latency between the client and the server, the connection setup and teardown handshake may cause a notable delay. As we look at the main page of *cnn.com*, we see that it consists of multiple resources that may be requested separately, including the content *html* page, multiple *style sheet* files (CSS), *JavaScript* files, images, icons, additional *html* fragments, etc. as shown in Fig. 1. At the time of this writing, the CNN homepage requires 121 resources for a total of 1.3and a load time of approximately 9 seconds. If the web browser uses HTTP 1.0 or older, then the TCP connection is closed after a single request/response pair. Fortunately [6] improves this behavior with persistent connection that stays open after a request/response pair and can handle additional requests; however, this only partially solves the multi-resource problem.

Network characteristics can significantly impact performance. The first characteristic we consider is end-to-end propagation delay, which is impacted by both topological distance and network congestion. For web applications that serve an international audience, the geographic separation results in some clients that are necessarily topologically distant from the server. The second characteristic we consider is bandwidth. Clients vary in the bandwidth quality of their network connection or in specific bottlenecks in ISP peerings [7]. High latency and/or low bandwidth can significantly impact page load time. From the client's perspective, we see that the page would load faster if some of the following factors were reduced (Fig. 2): First, the size of the content impacts bandwidth requirements; consequently, we should reduce the total size in order to deliver the content faster. Second, reducing the number of resources decreases both the overhead of TCP segment headers and the number of round-trip times for each resource request. Finally, we may expect that a client navigates among similar pages, which use the same design, and we can leverage this locality by caching static resources involved in the page design, which can speed up the consequent page loads.

3 Simulation Case Study

In this section, we provide a case study simulation of a web application on an emulated network. We apply our experiment on the Contest Management (CM) application under International Collegiate Programming Contest (ICPC)[8] [9]. This enterprise web application is hosted in Texas and used by around 2,000 universities

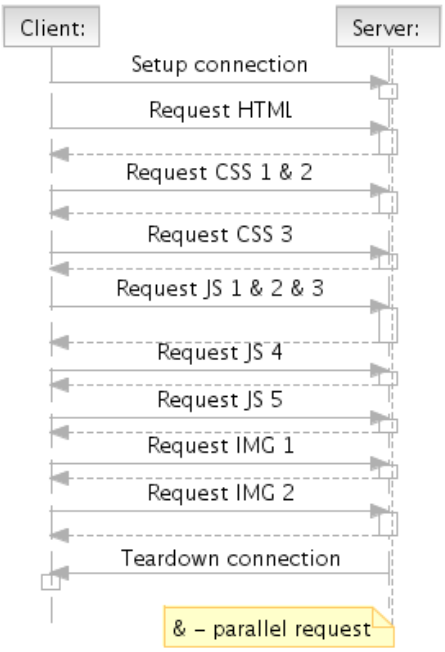

Fig. 1 Unoptimized client/server communication

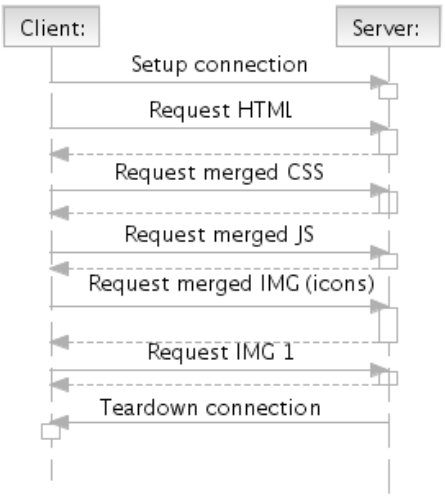

Fig. 2 Optimized client/server communication (no cache)

worldwide to register for a programming competition. Naturally, users expect this application to provide fast service delivery irrespective of network conditions.

Application performance testing requires the evaluation of user experience for a wide range of network bandwidth and/or latency. Developers often overlook problems with limited network performance because development environments typically exhibit very high bandwidth and low latency. To identify the application bottlenecks, we need to test the application under conditions reflecting the experience of the range of typical users. We propose providing this range of testing through network emulation by controlling the network latency and bandwidth as described in [7]. To accomplish this, we developed an open-source tool, CZProxy [10] [11], that provides network emulation where such network conditions can be configured. Commercial applications with similar functionality also exist[12].

In [7] we measure network conditions for various client conditions, allowing us to reproduce application behavior similar the remote user's experience. For this case study, we focus on the performance for a client with a latency of 600ms and bandwidth of 251KBs. This is based on our actual network performance measurements for a client in Japan accessing a server in the Czech Republic. We then select three, typical test pages from the CM application with differing complexity and evaluate the performance. In our simulation measurements, the initial, unoptimized version of the CM application produces pages from 907 to 1675KBs and serves our test pages in 40 to 60 seconds. A service delivery with timing in 40-60 seconds is really poor; consequently, we developed the following optimization approaches [10] based on the discussion in Section 2:

Obfuscation of resources

Obfuscation involves eliminating unnecessary whitespace and comments from CSS and JavaScript. Such data does not need to be sent over the network because they do not have any meaning to the final web page composition. Reduction of content to send over the network means faster service delivery to the client. In our simulation, we experience page size reduction of 195 to 323KBs, about 15-50% size reduction per resource.

Merging resources

We may merge static content such as CSS, JavaScript, and icons into a single file per type (see Fig. 2), resulting in reduction in cumulative TCP round-trip times, equilibrium discovery, etc. We can merge multiple icons by combining them into a single image and then using by CSS to display just a fragment of the merged image. This optimization is commonly applied in existing JavaScript frameworks.

For the CM application, there are two sources of CSS and JavaScript resources: custom and RichFaces. Custom resources are files created specifically for our application. RichFaces provides a third-party interface component library, and its resources are generated by the library. We evaluate two options for merging.

We begin by evaluating the performance of merging our custom CSS and JavaScript and merging RichFace's CSS. Here we reduce the number of resources per page by an average of 25. In our simulation, the typical page contains approximately 90-100 resources. We did not evaluate the impact of icon merging.

Next we add RichFace's JavaScript merging to our previous experiment. This had an impact on the

overall size, which increased by 321 to 648KBs per page; however, the number of resource per page was reduced by 40-60. This is a better option if we consider client web browser cache when the user needs to load the static resources once in the initial request and then use the cache for the resources. This initial load time amortizes over application use and in result may have better performance than the previous merge option.

Browser cache

Modern web browsers allow web application developers to specify which resources to cache and for how long. The client still issues a resource request to the server checking for update of cached resources. This may also reduce some other dependant requests that are invoked by application of a fetched resource. In fact, not all cacheable resources are cached. We can find these by using the CZProxy tool looking at the traffic on HTTP response headers for *304 Not Modified*. It is also important to follow separation of layout and structure and to factor out CSS from the dynamically generated HTML files, and if possible the same with the page embedded JavaScript. In addition to an abstract separation, this also allows caching the separated CSS and JavaScript files, which cannot be cached in case of dynamically generated HTML pages.

In our experiments, we found that caching reduced the fetched resource size by approximately 1MB for the second request in case of merged CSS and JS files.

Compress the data transfer

The most significant improvement to the service delivery involves compression to reduce the required bandwidth. Although this option is a great improvement for distant clients, for a bandwidth-rich client it can decrease the performance since the compression on the server-side and decompression on the client-side takes some time; however, in general we benefit with the compression option in most of public production level enterprise web applications. In our case study, compression reduces the uncached resources from 1.7MBs to 400KBs (1.3KBs to 300KBs) and the cached requests decrease by 26-78KBs.

In our simulation, we reduce the initial load time for the simulated Japanese client from 40-60 seconds to 7-10 second and subsequent access with the applied cache to 2-5.5 seconds. These optimizations take place on the server-side where we set the server to compress the content and the application to cache specific resources and to merge the CSS and JavaScript. The latter one can be automated by YUI compressor tool [13]. A more details of the measurement can be found at [7].

We have applied the simulation results to the production level application hosted in Texas and received positive feedback from regions where we originally had slow performance. The simulation helped to discover bottlenecks in the application for distant users. In addition we applied the simulation also for feature evaluation [7] and identified the delay contributors for the replacement.

4 Related Work

Recent research in optimizing web application for varying network conditions proposes changes and extensions to HTTP. The Structured Hypertext Transfer Protocol (STTP)[14] includes new messages to control the resource transmission for a particular web page. HTTP-MPLEX[15] employs header compression and response encoding scheme for HTTP. It is similar STTP in that it multiplexes multiple responses to a single sustained stream of data to speed response times and improve application layer use of TCP. While experiments show solid performance improvement with these protocols, they are not adopted by modern browsers. Our work focuses on optimization built on top of widely adopted technologies.

Optimization using persistent HTTP[16] may also improve application scalability. It is crucial for correct use of persistent HTTP to applying proper settings for the persistence connections timeout and max-client boundary. These are in fact additional optimizations that can be applied to the server from the perspective of multiple users. In our work, we focused at optimization of communication between a server and a client.

[17] proposes strategies for web page fetching in low bandwidth environment. Specifically, this work demonstrates the disadvantage of the common approach of greedy loading. The paper investigates communication between TCP and HTTP to analyze factors of the large response times. This research paper provides a browser-side optimization mechanism to reduce the user response time.

Related work mostly focuses on optimization through development of new protocols. Naturally, the main issue with these is the backward compatibility with legacy applications and adaptability of web servers and web browsers. To address this, [18] suggests focusing on improvements of legacy technologies rather than outright replacement. In our paper, we focused on a simulation of remote locations, which may help to identify bottlenecks and also with research experiments of new protocol adoption. In addition we proposed various optimization factors that should be applied regardless of application of any new protocol.

5 Conclusion

In this work, we face the issue of growing demand on web application features and capabilities, which directly impacts application performance. We discuss the background of the HTTP and TCP protocols and demonstrate the drawbacks of the technology of which one needs to be aware when dealing with application performance. We identify various technological impact factors on remote clients and apply these in our simula-

tion with emulated network conditions using a proxy tool we developed for an enterprise web application. We demonstrate significant improvements in the simulated application by proposing and applying optimization techniques, which results in faster service delivery for remote clients in the production application. A brief survey of popular web applications such as news servers CNN, iDnes, EuroNews, BBC etc., shows that all of these typically have pages with more than 100 resources, which could be merged down to a few. We see that all can benefit from our optimization suggestions. We believe that both legacy and new application development can benefit from our suggestions, and application developers must be aware of the underlying technologies to design better products from improving end user service delivery.

6 References

[1] Tomas Cerny and Michael J. Donahoo. Metamorpic: Self-contained photo archival and presentation. In *ISD 2010: Proceedings of the conference on International Conference on Information Systems Development*, Prague, Czech Republic, 2010. Springer.

[2] W. Richard Stevens. *TCP/IP Illustrated (vol. 1): The Protocols*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[3] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 2002.

[4] Hypertext transfer protocol – http/1.1, 1999. http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[5] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[6] Jeffrey C. Mogul. The case for persistent-connection HTTP. In *SIGCOMM 1995: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 299–313, New York, NY, USA, 1995. ACM.

[7] Tomas Cerny and Michael J. Donahoo. Evaluation and optimization of web application performance under varying network conditions. In Jan tefan Petr Peringer, editor, *Proceedings of the 44th Spring Innternational Confenece*, page 8, Ostrava, April 28 2010. Brno University of Technology, FEEI, VB - Technical University Ostrava, Wroclaw University of Technology, MARQ, Tiskarnicka.

[8] Contest management web application for ICPC, 2010. http://cm.baylor.edu.

[9] Tomas Cerny. The next generation web application framework for ICPC: Contest management system version 3 (CM3). Master's thesis, Baylor University, Computes Science dept., Waco, TX, USA, 2009. http://cm.baylor.edu.

[10] Tomas Cerny and Michael J. Donahoo. A tool for evaluation and optimization of web application performance. In Jan tefan Petr Peringer, editor, *Proceedings of the 44th Spring Innternational Confenece*, page 6, Ostrava, April 28 2010. Brno University of Technology, FEEI, VB - Technical University Ostrava, Wroclaw University of Technology, MARQ, Tiskarnicka. http://sourceforge.net/projects/cz-proxy.

[11] Tomas Cerny and Michael J. Donahoo. CZProxy - HTTP proxy/monitor, 2008. http://sourceforge.net/projects/cz-proxy.

[12] Charles - http proxy/monitor. http://www.charlesproxy.com.

[13] YUI compressor, 2010. http://developer.yahoo.com/yui/compressor.

[14] Bin Swen. Outline of initial design of the structured hypertext transfer protocol. *J. Comput. Sci. Technol.*, 18(3):287–298, 2003.

[15] Robert L. R. Mattson and Somnath Ghosh. HTTP-MPLEX: An enhanced hypertext transfer protocol and its performance evaluation. *J. Netw. Comput. Appl.*, 32(4):925–939, 2009.

[16] Akiyoshi Sugiki, Kenji Kono, and Hideya Iwasaki. Tuning mechanisms for two major parameters of apache web servers. *Softw. Pract. Exper.*, 38(12):1215–1240, 2008.

[17] Tae-Young Chang, Zhenyun Zhuang, Aravind Velayutham, and Raghupathy Sivakumar. Webaccel: Accelerating web access for low-bandwidth hosts. *Comput. Netw.*, 52(11):2129–2147, 2008.

[18] Anthony Finkelstein and Jeff Kramer. Software engineering: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 3–22, New York, NY, USA, 2000. ACM.