A Tool for Evaluation and Optimization of Web Application Performance

Tomáš Černý[1]
cernyto3@fel.cvut.cz

Michael J. Donahoo[2]
jeff_donahoo@baylor.edu

Abstract: One of the main goals of web application developers is to design fast systems that can respond to users in short time. Fast systems response increases user satisfaction. Although this fact is well known there does not exist many tools that could help to determine system bottlenecks for distant users. The main issue for web applications is that users come from all around the globe and the application is most of the time served in one location. We may evaluate the application in laboratory conditions, but these differ from the real ones. Network characteristics may significantly impact the true bottleneck. Many techniques are employed for performance improvements, but there is not an easy way to test the impact of the improvements for a distant user in the real environment. We propose and provide a debugging web proxy, CZProxy, that can simulate the environment for distant users. Our proxy can simulate both latency stretch and bandwidth restrictions, allowing optimization over a variety of end-user connection scenarios.

Key Words: Web page load optimizations, web debugging proxy, performance evaluation

1 Introduction

To test web applications we start with correctness evaluation, continue with performance testing and optimizations. Performance evaluation should bring up the most significant application bottlenecks. Performance is measured in many ways that may involve SQL queries, service profiling or response to various end user requests. It is a common practice to test the application in a laboratory conditions where all the tests may provide better results that what the real environment prepares. Laboratory testing make sense for tests that focus at the server site, but some of the bottlenecks may not be caught for client-server interaction. Client-server interaction involves a client that requests an application page and wait for it to load. The page is composed from static and dynamically generated resources. The client-server interaction may experience specific application bottlenecks for distant users. Various distant users experience various network conditions, these conditions should be simulated for the tests to determine extra bottlenecks in the web application.

1.1 Network Emulation

The server location relative to its client impacts the end-to-end propagation delay and thus the application users are influenced by the network quality of service. Network connection differences comes from the distance of communicating locations, the underlying technology, data traffic or bottlenecks of ISP peerings. If we try to describe and compare the network connections then we look how much data can be transferred withing a given time and how

[1] Department of Computer Science and Engineering, Czech Technical University, Prague, CZ

[2] Department of Computer Science, Baylor University, Waco, TX, US

long it takes to receive the first data piece of our request. These parameters bandwidth and latency sufficiently describe various connections. Low bandwidth and high latency significantly impact the page load time. Application evaluation will differ under different network parameters. Laboratory tests most probably experience almost no latency and large bandwidth, and these may fail to identify problems with page load times. We propose a tool CZProxy for performance evaluation that allows to debug web applications and receive detailed page load time information. In addition our tool incorporates the ability to emulate various network conditions. The tool allows performance testing for a variety of clients where optimizations can be evaluated under various network contexts.

1.2 Page Load Time Evaluation

Modern web applications provide advanced features that start to compare with standalone applications. The modern trend where web developers use third-party generic libraries that provide rich functionality also bring its drawbacks. These drawbacks come in form of complexity of page resources that are not in developers direct control, and most of the time increases the application load times. To build an application full of rich features is not very hard these days, but the load times might be significantly affected in a negative way. To locate the application bottleneck we need to know all the resources that come with the requested page. Every resource comes with its load time and size. To gather all these information we may use our web debugging proxy tool. The proxy intercepts and forwards all requests and responses between the end user and the server. The proxy records time and size contribution of every resource that is being requested to load the entire page. Developers can observe the costs of various elements and identify the biggest contributors to the load time. The optimization may come in various improvements, such as service computation optimizations, caching, resource reallocation, feature reduction, fetching strategies, etc.

1.3 Organization

We organize the rest of the paper as follows. In the next chapter we focus on the web debugging proxy, we analyse the requirements and alternatives for the tool implementation, we also introduce the tool and describe the main features. The following chapter deals with related work and finally we provide a conclusion of this paper.

2 Debugging Proxy Development

CZProxy works by acting as a web proxy for the testing browser. The proxy can then observe/record all incoming and outgoing traffic, measure performance of various elements, and modify traffic to emulate network characteristics, etc. Based on information gathered during the test, application developers can experiment with a variety of optimizations under a variety of network conditions to tune the application to best server its client base.

2.1 Data Handling and Presentation

To provide the necessary insight for optimization, we must gather sufficient data during the page request based on the HTTP protocol and request/response headers [7]. However, simply knowing the data flow in each direction is not sufficient. It is important to know the timing of the data flow and the request dependence. Our proxy should measure and monitor all of this information. It is reasonable to have a chance to influence the data flow and it might be done in a perspective when we delay requests and produce an extra latency or we restrict the flow in its width to shape the bandwidth. Sometimes we may need to apply the measurement only to selected sources, so there should exist some sort of request filtration. The request filtration

feature will be useful when debugging subsequent Asynchronous JavaScript and XML (AJAX) calls, and our attentions is only at these requests, and we ignore other resources. Gathered information log needs to be presented to the user in a simple way. From these information he will be able to identify some deficiencies, mostly from the timing. All requests are lined in a list that should allow to drop an element if necessary. There might be more perspectives in which the information are displayed. We look at the gathered information as a list of all requests, but we also care about detailed information for each of the requests. Every request provides information about its socket, timing, HTTP headers and also its data. One perspective we may need are timing details, the other might be the content of lookups and the received data. There are two options for the user interface to start with. The first option is web based and the second one is standalone based.

2.2 Implementing a Web Debugging Proxy

To implement a tool with given specification we choose two layer approach as recommended by [5]. Bottom layer is a networking proxy that captures all data coming through. Upper layer is the user interface.

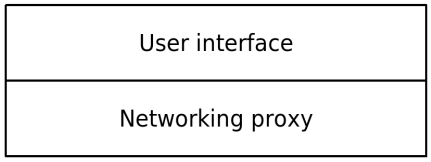

Figure 1: CZProxy architecture

The bottom layer specifies interface and its implementation can be based on different approaches following the interface. We implement 5 different approaches. Besides thread pool and thread per connection approach or NIO/selector, we also implement new Java 7 approaches. These new approaches Callback and Future are new ways to handle sockets and files in asynchronous way. This takes the advantage of underlaying operating system facilities [4]. This layer is responsible for data communication between client and server. Client request a page to load, this layer receives the request and forwards it to the server, from here are captured timings and other statistics, so there are three main responsibilities: implementation independence, request forwarding and statistics.

Taking the advantage of strict layered system we can build more upper layers independent of each other. We can build standalone client or web-based. This user interface layer is not bound to some decision made below and we just implement user interface we want. Many related projects[1, 2] chosen standalone Swing based user interface we rather use web based. We mention more about related projects later. The user interface uses HTML. To archive dynamic content generation there is also a need for an application server. Or we can build a proprietary dynamic content engine, which seems as a better solution for this light weight tool. We split our upper layer in another two sub-layers. The bottom one is a web engine and other one is the concrete view. Web engine takes care of dynamic content generation a also about form submission. The build web engine is a lite web framework that offers necessary features. The engine provides an application and request scopes for the tool, which is enough for the purpose we use, the engine is capable to request all data gathered by lower layer. The upper view is built in an engine language which makes it trivial to rearrange layout and elements like forms or tables. This provides an advantage for the end user to define his own user interface without any code recompilation. Our engine language is a mix of JSP and PHP which makes it simple to use.

2.3 Using CZProxy

This section describes our tool[6] from the user perspective. Since we use Java the tool runs under every operating system and will cooperate with every web browser. The tool is delivered in form of a Java Archive (JAR) file and a web directory. User can specify settings like proxy port, or network proxy implementation in a configuration file. We start the tool, set the proxy at the web browser and navigate to URL `czproxy/config.icpc`.

In the main view we can set bandwidth, latency or filters as denoted by Figure 2.

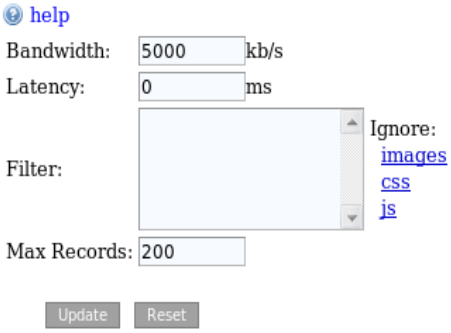

Figure 2: CZProxy settings

Every lookup of the browser is now recorded by the proxy. We can hit our web page and then return to the configuration page. The page shows recorded lookups in a table with detailed information about paths, sizes and times (Figure 3).

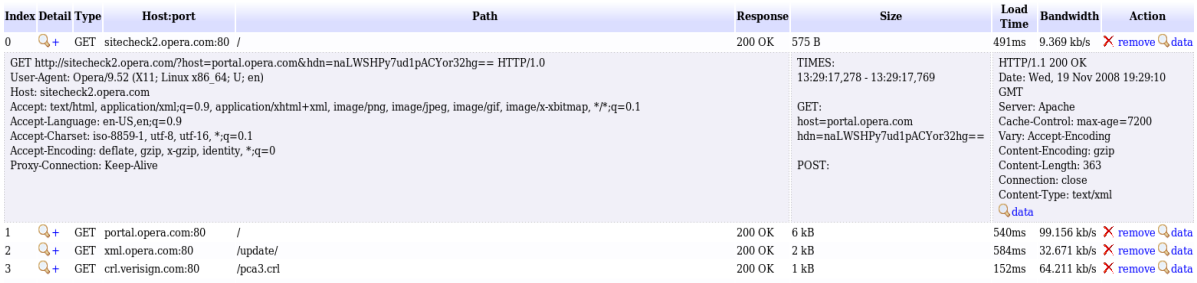

Figure 3: Detail information

We get more information by toggling row detail. Row detail contains parameters and headers for request and for response. If we want to see a detailed information about the data we select a given record and redirect to the detail page. That shows all information about the record and shows delivered content in uncompressed form with proper encoding. The configuration page has a lot of information, but it may be useful to see a graph of timing. For this reason the tool provides another view that has such a time graph (Figure 4).

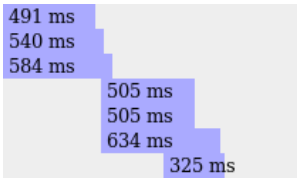

Figure 4: Timing of requests

The tool also provides a global information for the entire load time which might be one debugging criteria we look at.

3 Related work

We started to write this project one year ago and there was not many related applications for the network simulation, although among these we found three interesting projects. The closes project Charles [1] provides web debugging via proxy similar to our project. The first difference is that Charles is a standalone plug-in type application with a graphical user interface, while our interface operates via a browser, which allows easier operation, since browsers are virtually universal, and remote access which is useful for sharing among developers. Users specify a list of requests, and Charles displays detailed information based on the specification. Charles provides two additional features: CSV export and Secure Sockets Layer (SSL) support. Unfortunately, Charles is neither open-source nor free to use. In addition, the user documentation mostly focuses on debugging and provides no guidelines for performance optimization.

Another related project is Fiddler[2]. Similar to the previous one also Fiddler uses standalone user interface and provides debugging functionality. Besides the traffic monitoring Fiddler also offers extensions to highlight syntax for the replied content that is in HTML, XML or in JavaScript. The extensions might be useful when we want to look into the content of the server replay. What we believe is one significant issue is its compatibility only with Windows. As well as Charles neither Fiddler user documentation does not focus on performance optimization.

Little different tool is Firefox plug-in Firebug[3]. Firebug is embedded in the browser that allows web debugging, script debugging, style sheets debugging, code to view hot rendering or network activity monitoring. Firebug focuses mainly on page content and its rendering. The network monitoring has very nice design but is not very credible and it does not allow to simulation of latency or bandwidth. We consider Firebug plug-in a very useful complement to our tool (or vice versa).

Nowadays there came similar projects to our tool where our tool predates these projects. One of them is Page Speed[8], which is an open-source add on for Firefox and Firebug. The user documentation provides very rich guidelines for optimizations. Where they also focus on browser rendering. The disadvantage for this project is its compatibility only with one of many web browsers and that developer cannot adjust network conditions.

Another very interesting idea comes from Google labs, and is called Webmaster Tools[9]. Developer can add his public web project URL and these tools help you with search result performance evaluation in the way Google sees it, where all hacker attack can be identified. This tool also provides developer with timing and latency information about their site and also with the average page load time. The user documentation provides reasonable set of recommendations for the developers and references also Page Speed. The disadvantage is that this tools work only for publicly accessible web applications and forces user to use Google services which my add an extra time to the debugging.

Among the related projects we believe that our open-source tool brings a useful way to evaluate web applications. We have build the tool independent of a web browser when we leave the web browser option on the user. Our tool allows to setup network conditions and remote access that might be useful for multi-user use. We see that many similar project are exists but many of them have some disadvantages. We suggest to complement our project with Firebug where the developers receive strong evaluation environments with capability to debug the rendered page.

4 Conclusion and future work

In this paper we identified the need for a tool that helps with simulation of remote locations. We described requirements and design for such a tool and we provide an open-source tool implemented in Java. We believe that many developers find this tool useful and will apply it in the tests for their developed products. All clients appreciate faster page loads with connection to less congested Internet.

The tool CZProxy can be improved in various ways, a standalone user interface could be implemented. Functionality could be improved with searches, secure socket layer, syntax highlighting or break points. Although all proposed features are not necessary for the web page load time evaluation and debugging.

References

1. Charles Web Debugging Proxy, http://www.charlesproxy.com
2. Fiddler – Web debugging proxy, http://www.fiddler2.com/fiddler2
3. FireBug – Firefox plug-in, http://getfirebug.com/
4. Open JDK project NIO, http://openjdk.java.net/projects/nio
5. Kenneth L. Calvert and Michael J. Donahoo, TCP/IP Sockets in Java™: Practical Guide for Programmers, ISBN: 1-55860-686-8
6. CZProxy binary version, http://cz-proxy.wiki.sourceforge.net
7. Request for comments RFC, http://www.ietf.org/rfc/rfc2616.txt
8. Page Speed, http://code.google.com/speed/page-speed/
9. Webmaster Tools, http://www.google.com/webmasters/tools
10. JQuery, JavaScript Library, http://jquery.com