REALISTIC BENCHMARKING OF SHARED WEB HOSTING ENVIRONMENT

Matija Jekovec[1] and Jaka Sodnik[2]
[1]Domovanje d.o.o., Slovenia
[2]Faculty of Electrical Engineering, University of Ljubljana, Slovenia

Abstract – In this paper we focus on several issues related to slow responsiveness of web sites. Page response time directly correlates to user abandonment, which consequently means a loss of profit. We analyze the whole round trip time (RTT) for each web page, excluding the factors that are not vital for optimization. We concentrate on the main factors behind slow response times and focus on the indicators that can be improved by web hosting providers running a shared hosting environment. In addition, we also describe a fresh approach to benchmarking. It is far better suited for a shared hosting environment using real log files that are already collected in the course of regular traffic. These log files are then used to recreate log traffic in order to perform various tests with different setups. The insights gained from these tests represent the basis for business decisions on infrastructure for the next 3-5 year period.

1. INTRODUCTION

One of the most important features of the web is its enormous growth and expansion. More than two billion users search and browse through over 300 million web sites, with approximately 20 million new web sites being added every month [1]. A huge set of various tools and systems enable the development of new simple websites in only a couple of minutes. Many new sites focus very little or no attention to UX (user experience) which is strongly connected to interface design, page structure, page loading time, etc. We believe this field is particularly important in case of web applications and sites. Namely, web users are very often less experienced than the users of desktop applications, since the former often include children on the one hand and elderly on the other. Each website should therefore be tested and exposed to various types of users before going online. Several areas need to be focused on when performing these tests, including site organization, design, problems related to user search, negative effects of page loading delay, trade-offs between site depth and breadth, problems with users' subjective perception, etc.

In this paper, we focus on page load time and its impact on user behavior and decisions. The delay imposed when moving from one site to another is one of the most annoying aspects of web navigation. It strongly influences the final user experience and the usability of the site. In addition, search engines such as Google started to include page loading time among the most important factors when ranking their final search results. When it comes to e-commerce, every second of delay directly correlates to customer dissatisfaction and business loss.

2. SUBJECTIVE PERCEPTION OF WEB PAGE LOADING TIME

Web users tend to use and revisit a certain web site as long as it serves their needs. Usually they do not switch to a competitive site unless something disappoints them on the original page, and the most common reason for their disappointment are unexpected delays and unresponsiveness of the site. It is therefore important to consider the users' expectations. If they are accustomed to good performance of a certain web page and its performance suddenly changes, they could get disappointed rather quickly and they could switch to a new web site and stick with it until new performance problems arise.

The total response time is the most important factor from the end-user perspective. Still, it is almost impossible to measure it or predict it, as it relies on multiple factors. A certain web site works perfectly when tested in a local area network with bandwidths of 100Mb/s or even 1Gb/ and thus offers an excellent user experience. When the same site is accessed through a local Internet Service Provider (ISP), with bandwidths around 1Mb/s or even less, the user experience drops significantly. The users mostly tend to blame designers and developers for their bad experience, no matter the real reason for longer response times.

Customer conversion rate

The term "customer conversion rate" represents a quantitative measurement of the success of an e-commerce site [2]. It is a ratio between the total number of people who actually buy something from the site in a certain period of time, divided by the total number of website users. If, for example, 12 people bought something from the e-shop that was visited by 200 people, the conversion rate is 6%.

Several studies focused on finding the correlation between delay or page loading time and user patience and potential business loss. They all established that page loading time is one of the most important factors for page abandonment [3][4]. Every second can be directly converted into the percentage of users leaving the page due to long waiting time, as shown in Figure 1.

A delay of one second in page-load time results in 11% less page views, a 16% decrease of user satisfaction and a 7% decrease in customer conversion ratio [5]. It is also important to point out that almost 50% of the users expect a web page to load in two seconds or less. 40% of the users abandon a website if it takes more than 3 seconds to load, and 44% of them would tell a friend if they had a poor experience while shopping online. Large delays are

thus not only preventing the conversion of current visitors on the site, but also potential conversions of their friends and colleagues.

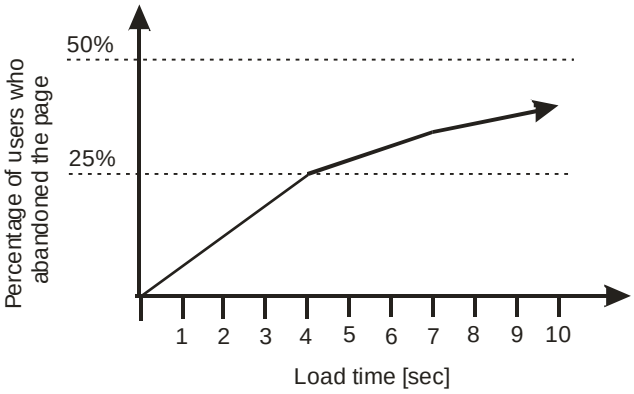

Figure 1. Abandonment rate in relation to page-load time [3].

An additional study was performed in order to establish the exact correlation between page loading time and money value or the likelihood of a user converting to a customer. The experiment was based on monitoring page performance and measuring the final revenue. The final results are summarized in the following pie chart:

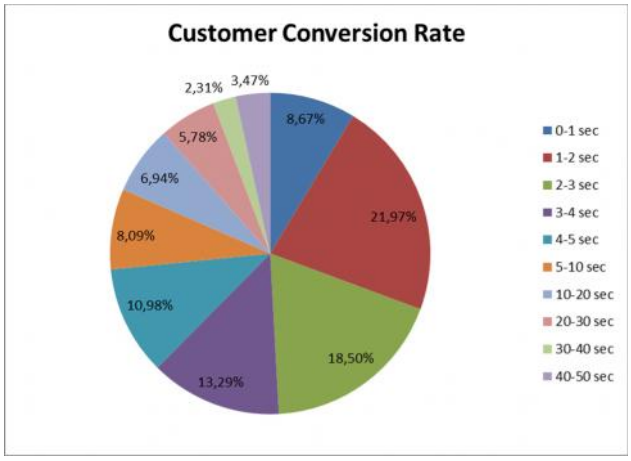

Figure 2. Customer conversion rate in relation to page response time.

The data in Fig. 2 clearly show peak conversion ratio at a delay of two seconds, dropping by 6.7% percent for each additional second of waiting time. The user who abandoned the page had to wait in average three to four times longer than the users who converted to customers [5]. The authors calculated the loss of revenue for each additional second of page loading time. As an example, they imagined an e-commerce website with an average purchase amount of 75$, a conversion rate of 5% and 400.000 unique visits per month. In such a case, every additional second of page loading time would results in an annual revenue loss of $1.3 million. In the case of an e-commerce site making $100.000 per day, an extension of page delay for just one second would result in a $2.5 million loss in a year.

3. THEORETICAL APPROXIMATIONS OF PAGE RESPONSE TIME

The final page response time is measured from the moment when the user inputs (selects, clicks etc.) and confirms the required url until the moment when the web page completely renders in a browser. The basic equation for the estimation of page response time is the following [6][7]:

$$\text{Page Response Time} = \frac{\text{Page Size}}{\text{Minimum Badwidth}} + \left(\text{Round Trip Time} \times \text{Turns}\right) + \text{Server Processing Time} + \text{Client Processing Time} \quad (1)$$

Equation 1 is a very rough approximation of the problems connected with page load time and relies on several assumptions and generalizations. However, it successfully represents the most important variables that impact the final response time. In the following sub chapters we will briefly explain each of the contributing factors.

Page size and minimum bandwidth

The total size of a web page is relatively easy to measure and is usually specified in Kbytes or Mbytes. A bigger size relates to longer load time and vice versa. When measuring or estimating the total page size it is important to include all internal and external resources: images, videos, style sheets, JavaScript files and libraries, etc.

Minimum bandwidth is the slowest part or the bottleneck of the connection between a client and a server. In most cases, the minimum bandwidth corresponds to the "last mile" or to the user's ISP connections. Nowadays, broadband connections with transfer rates of 1Mbs or higher have already replaced the majority of old 56Kbs or even slower modem connections, which contributes significantly to a better performance and lower page loading time.

The final estimation of the impact of page size and minimum bandwidth is a ratio between the two called the download time (DT):

$$DT\,[\text{sec}] = \frac{\text{Page size}\ [\text{MB}]}{\text{Minimum bandwidth}\ [\text{MB/sec}]} \quad (2)$$

Round trip time and number of turns

Round trip time (RTT) refers to the time difference between the moment of sending the request to the server and the moment when the user starts receiving the response. A typical web page requires multiple requests and responses even for a very simple page. The simplest way to measure the RTT is with the aid of a ping tool that sends 32 bytes of data to the designated IP address and measures the time it takes for the server to respond. The request/response cycle is repeated several times and the minimum, the maximum and the average RTTs are provided.

The number of turns refers to the number of requests that have to be sent to the server when downloading the page. Each request requires the establishment of the separate TCP connection and a DNS address resolution. The process of opening the TCP connection is a three-way handshake protocol, in which three packets of information and sequence numbers are exchanged. The external objects of the web page are not transmitted along with the base page. The base page provides the client with the instructions where extra material can be acquired. Therefore, each additional part of the webpage (image, css file, javascript file, etc.) requires additional communication. The minimum number of turns required to download a simple web page is four. The first turn represents the DNS lookup, while the other three turns request and transfer the basic content. All modern browsers enable the establishment of multiple concurrent TCP connections (from 6 to 15), which enables several pieces of content to be downloaded simultaneously [8]. Servers also often limit this number for an average client. The final estimation of the impact of TSS and the number of turns is calculated by Eq. 3:

$$DT[\text{sec}] = RTT[\text{sec}] \times \left(4 + \frac{3 \times \text{Number of Objects}}{\text{Number of simultanous connections}}\right) \quad (3)$$

Processing time

The final part of Eq. 1 is the processing time required on the server to process the request and on the client to render the result in the browser. The processing time on the server depends strongly on the type of the requested web page. It could be a simple static page requiring no processing or a complex dynamic script which requires the server to run it and create the response. Sometimes various further backend scripts and complex programs are included, which further extends the processing time. There is a similar situation on the client side as well. A simple html page can be processed and rendered very fast, while a web page with a lot of javascript and other dynamic content requires more processing and rendering time. The final processing time on both, i.e. the server and the client, depends on hardware capabilities and availabilities and is therefore impossible to estimate. Especially on the server side, the processing time depends strongly on the current occupancy of the server and the total number of clients being served simultaneously. The best estimations of processing times can only be made based on measurements and extensive testing.

4. PERFORMANCE TESTING

Performance analysis of web-based applications is conducted on two different levels. The first level (we will call it application level) is focused on processing a single request (usually HTTP request) and is conducted for each part of the application separately. The second level of performance analysis (the so-called infrastructure level) has to do with “real life” loads and focuses on testing the infrastructure (physical and virtualized hardware, operating system, software services). The first level is about taking apart each request and dissecting it, while the second level deals with observing the infrastructure while being flooded with large quantities of data.

Application level testing and optimization

Application level testing is a development practice that does not require any special setup or hardware and should be done by every developer. Application testing is more about relative gains in performance (reduce # of requests for every visit, reduce relative memory usage for every request) and can not always be precisely extrapolated to gains in production environment. The tools used in performance testing are freely available, some are even included in browsers or available on line.

Most of the application level optimization in web application can be done by reducing the number of HTTP requests and reducing the round trip time of each request.

The requests can be divided into static and dynamic. Static requests require the web server to simply return the requested resource without (much) processing, while dynamic requests are always processed on the server side. In average, the ratio between static and dynamic requests in web applications (e.g. web sites, web based application) can range from 1:10 to 1:100 in favor of static content. Static consists of resources like images, sounds, style sheets, Javascript source files and other documents. Performance optimization techniques with static content include:

- caching on the client (by using HTTP headers like “etag” etc.),
- switching to a fast and lightweight web server for static content while handing over dynamic requests to another service,
- re-compressing images (most of the images are not optimally compressed for the web; the compression could sometimes be improved for up to 50% without or with minimal loss of quality),
- image spriting to reduce requests (combining images into a single image and using style sheets to crop different parts of the image to get the desired effect),
- inlining images, style sheets and scripts (small images can be included in dynamic content to reduce the number of requests),
- style sheet and script outlining (separate requests for each resources but can be cached on client),
- usage of content delivery networks (allows static resources to be served from location geographically or topologically closer to the end user).

The tools used to test and optimize performance at this level can be found in all modern browsers (search for development tools) and are usually represented as a waterfall model of resources that is downloaded along with all the data about each request (see Figure 3). Most of these tools are “real time”, which means they can also be used to test and optimize web applications that use dynamic (asynchronous) resource loading and execution.

Performance testing and optimization is measurable; however, the effects of perceived improvements must not be ignored. Deferring execution of scripts until the page is loaded is not a performance technique per se, but is perceived as such because the user can already look at (if not yet interact with) the web site even though it is not completely loaded yet.

Static content might outnumber the dynamic one in the number of requests, but is much "easier" on the server. This is why dynamic content requires a different approach to performance analysis. This is usually done by profiling a request that gives insight into the application on a method call level. Each method call is measured and a "call tree" is constructed, which (depending on the tool) allows a graphical analysis of bottlenecks in the application. These tools are usually run on the server and analyzed on the client (Xdebug + kcachegrind combination for PHP). Optimization techniques include all standard software optimization techniques that are not going to be discussed in detail.

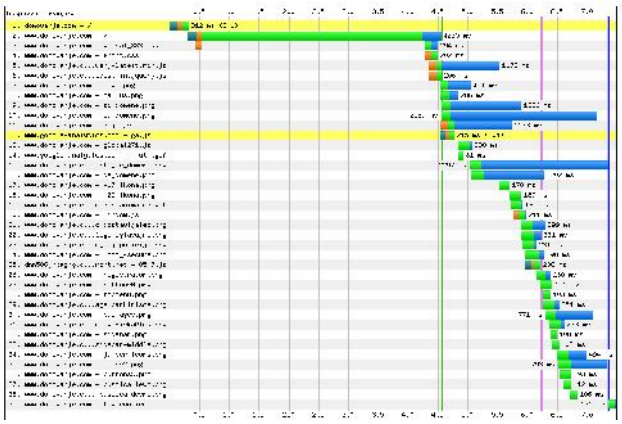

Figure 3: Example of a waterfall diagram for network exchange for a single web page

Infrastructure level testing

While application level testing is done in a development environment, infrastructure performance level testing should be done in an environment as close to production as possible. As it is usually too expensive to duplicate the whole environment, a production version of the application is used on a virtual server setup similar to the production environment but on a much less powerful physical hardware. Infrastructure performance testing is done with benchmarking, e.g. flooding the web application with traffic and monitoring the response of the system under test (SUT).

Compared to application level testing, this type of testing is more difficult to setup. To begin with, hardware requirements, virtual server setups (which can be quite complex when using clusters of dedicated servers) and application deployment (different servers, different parts of applications) must be dealt with. While application level testing is done in real time, benchmarking requires one to leave the system up and running for a longer time (minutes or hours) and can only then analyze the data. This means that iterations of build, deploy, analyze and fix are much longer. As a consequence, this type of testing is avoided by teams and is rarely included in regular development processes. Also, as it is a type of a black box testing, only indirect observations of the application can be made through the measurements of the system. These measurements are "operating system" level measurements, such as memory and bandwidth usage, different input/output parameters, process count and lifetime. Another problem with this type of testing is in the manner in which the load for the system is generated.

The available tools that are mostly used in the web community usually generate "synthetic" traffic with the aid of different parameters. For example, the ab [9] or siege [10] tools use two options for the number of requests and number of concurrent requests. These parameters allow complete control over the requests, including "advanced" usage like simulating an interchange of user credentials or TCP window size. Still, all of this is only a representation of an "average" user and is not close to real traffic.

Another tool called Jmeter [11] (and others like Selenium, Watir etc.) allows a developer to "record" a typical usage scenario and then "replay" it on the server. Again, this is just a representation of a user, not real traffic. Another problem with this approach is specific to web hosting industry. Most of these tools are directed at high traffic websites where a single domain is flooded with traffic. In a shared hosting environment, a single server can serve a few thousand web sites, each one hosting a different application with different usage scenarios and different performance issues.

On the other hand, most of the web servers are by default configured to log every access to a specific web site. Logs are used as a debugging or an incident analysis tool; however, they are invaluable when assessing web traffic.

For these reasons we decided to build a tool that would analyze performance at the infrastructure level using the existing data already collected.

"mlogreplay" script

"nodejs-logreplay" script was originally created by Adam Lundrigan as node.js program with the exact idea we had in mind. It was a starting point that we extended to support multiple processes (and thus taking advantage of the underlying hardware), support a range of outgoing IPs (to avoid DOS counter measures by target host) and allow targeting multiple domains (HTTP host names) on a single IP (shared hosting environment). Our server environment uses "one log file per web site" setup, so we also had to develop a script that combined all of the log files into one unified log file.

The resulting application first parses the Apache access log. Then it uses date and time information in the log file to combine requests in one-second bursts where all the requests are made in parallel. Due to the sequential (synchronous) nature of node.js, only a smaller number of requests are made in parallel (currently limited to the number of cores on the executing machine). However, the requests

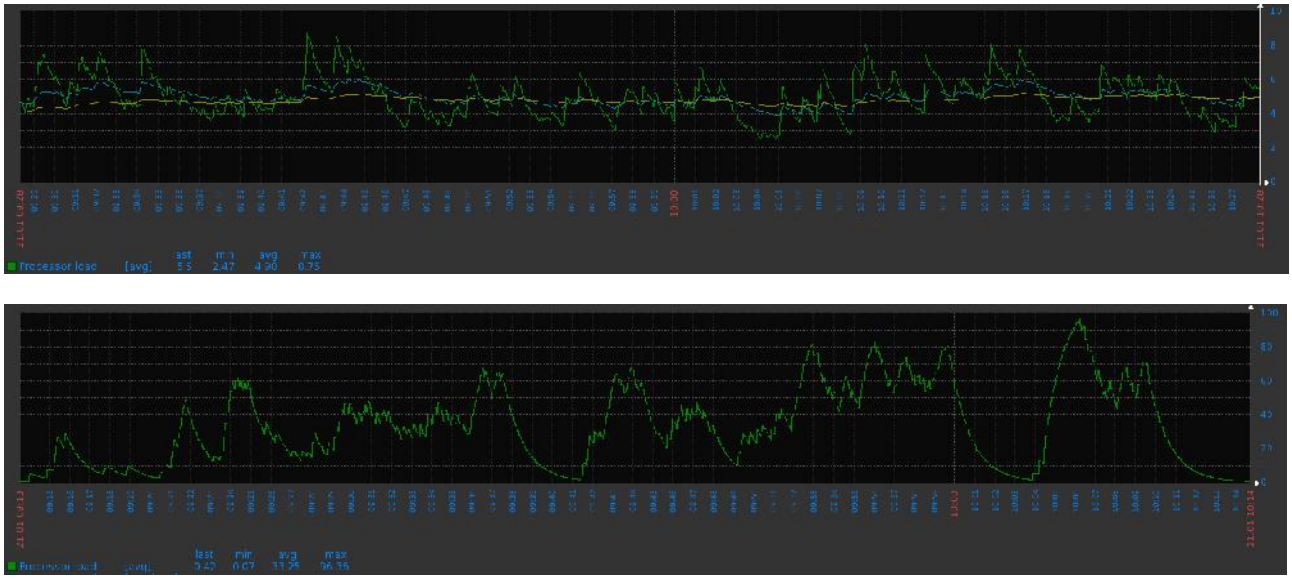

Figure 4: CPU load measurements (in %) made with Zabixx tool, comparing normal traffic conditions (upper image) and high traffic conditions (lower image)

made by the program do not block while waiting for a response, because the asynchronous callback model is used. This allows for a partially parallel method that works reasonably well in practice.

Currently, the program uses a random strategy to select an IP from a pool of assigned IPs when connecting to a target host, but additional strategies can be implemented and plugged in the script.

Destination host is defined either by the IP address or domain name. This host name is not actually used in a HTTP request header. Instead, the host name is read from a log file, which allows the simulation of requests to a shared hosting infrastructure.

Another interesting feature of the described program is the manual setting of playback speed. This allows us to "fast forward" the replay and test the performance with the same pattern with a much larger load.

"Real life" example

Domovanje d.o.o. along with its daughter company Domenca is the largest web hosting provider in Slovenia with an approximately 30% share of the market. Currently, the company's greatest challenges lie in consolidating and upgrading the infrastructure needed to operate a large customer base and grow this customer base in the next 3-5 year period. We used the "mlogreplay" as a part of the testing suite that was used to determine the viability of using a NAS in the production environment for shared web hosting environment. We are currently using separate physical servers, but would like to consolidate storage into a single solution for easier maintenance and higher availability of the infrastructure.

A testing environment was set up that included real storage solutions from three of the more established hardware vendors and production machines that were connected to the storage solutions. These clones were complete copies of virtual servers running in production, but we disconnected them from the internet. A "control" system with hard drives in the server was used to determine the performance relative to current infrastructure.

"mlogreplay" script was then run on all different setups for a prolonged period of time and measurements were collected using the Zabbix [12] monitoring solution. Based on the comparison with the generated load ("ab" tool was used), we can compare different machines much more easily. Figure 4 compares the CPU load measurements in normal traffic conditions (upper image) and high traffic conditions (lower image). The screenshots are taken from Zabixx tool.

We were also able to approach various vendors with very specific issues. Some of them were setup issues that were easily corrected, while others were actually related to the hardware or software performance of vendor machines, which allowed us to make the correct business decision.

5. CONCLUSION

The perceived response time is one of the most important performance indicators for the end user. As a hosting provider in a shared hosting environment with thousands of web sites (and thousands of web developers) sharing their resources, it is impossible to substantially improve it. An individual provider can put focus on the education of web developers (similarly to Google's "Make the web faster" initiative) and improve the networking and hardware infrastructure. Although this is not the biggest source of optimization, it can make a big difference if approached systematically. We tested and evaluated our next generation infrastructure by developing a fresh

approach to benchmarking and applying “real life” traffic loads to the system setups. We could easily determine that mid-level NAS solutions are not suitable for our needs. This kind of setup would improve the reliability of the infrastructure by allowing us to start new virtual servers and to replace or add new servers to existing ones instantly. The “single NAS storage solution” introduces a new single point of failure in the infrastructure. Performance-wise, a mid-level NAS would barely cover the current needs of the company. However, it would not allow us to grow in the 3-5 year period that was the main goal of this selection process.

REFERENCES

[1] “*Internet 2010 in numbers*”, http://royal.pingdom.com/2011/01/12/internet-2010-in-numbers, 01/2013.

[2] “*Customer Conversion*”, http://www.thesitedoctor.com/website_evaluation_benefits/Customer_Conversion.htm, 01/2013.

[3] “*How Loading Time Affects Your Bottom Line*”, http://blog.kissmetrics.com/loading-time, 01/2013.

[4] “*Speed Is A Killer – Why Decreasing Page Load Time Can Drastically Increase Conversions*”, http://blog.kissmetrics.com/speed-is-a-killer, 01/2013.

[5] “*Just One Second Delay In Page-Load Can Cause 7% Loss In Customer Conversions*”, http://blog.tagman.com/2012/03/just-one-second-delay-in-page-load-can-cause-7-loss-in-customer-conversions, 01/2013.

[6] Savoia, A.: “*Web Page Reponse Time 101 - Understanding and measuring performance test results*”, 2001.

[7] Sevcik, P. and Bartlettm J. “*Understanding Web Perforance*”, Business Communications Review, Oct 2001.

[8] “*Browser behaviour and performance*”, http://loadimpact.com/blog/browser-behaviour-and-performance, 01/2013.

[9] “ab – Apache HTTP server benchmarking tool”, http://httpd.apache.org/docs/2.2/programs/ab.html, 01/2013.

[10] “Joe Dog Software - Siege”, http://www.joedog.org/siege-home, 01/2013.

[11] “Apache JMeter”, http://jmeter.apache.org/, 01/2013.

[12] “Zabbix: The Enterprise-class Monitoring Solution for Everyone”, http://www.zabbix.com, 01/2013.