Client-Side Monitoring Techniques for Web Sites

Ricardo Filipe and Filipe Araujo

CISUC, Dept. of Informatics Engineering
University of Coimbra
Coimbra, Portugal
rafilipe@dei.uc.pt, filipius@uc.pt

***Abstract*—Ensuring the correct presentation and execution of web sites is a major concern for system developers and administrators. Unfortunately, only end users can determine which resources are available and working properly. For example, some internal or external addresses might be unavailable or unreachable for specific clients, while seemingly available resources, like JavaScript, might run with errors in some browsers. While standard monitoring and analytic tools certainly provide valuable information on web pages, problems might still escape such measures, to reach end web users. To demonstrate the limitations of current tools, we ran an experiment to count web page errors in a sample of 3,000 web sites, including network and JavaScript errors. Our results are significant: as many as 16% of the top 1,000 sites have errors in their own resources; less popular sites have even more. Based on these results, we make a review of three client-side monitoring approaches to mitigate such errors: stand-alone applications, browser extensions and JavaScript snippets with analytic tools. Interestingly, even the latter approach, which requires no software installation, and involves no security changes, can cover a large fraction of existing web errors.**

***Index Terms*—Web monitoring, Client-side monitoring, Analytics.**

I. Introduction

Web site monitoring plays a major role in mitigating the negative consequences of programming errors and network malfunctions. Several studies [1], [2] show the significant impact for companies, when users experience blank pages, missing items, or are unable to interact with the web page. One may consider that the consequences would be minimal, but the true costs of a web page malfunction come from disgruntled customers and the respective impact on the company's reputation. To control failures in web resources, system administrators must keep a watchful eye on a large range of system parameters, like memory occupation, network interface utilization, among an endless number of other metrics, such as page load time. Unfortunately, even with all these metrics — that add complexity and intrusiveness to the system —, clients may experience some problems, due to web page external content, and client specific conditions, such as network glitches or incompatible browser versions. In fact, even sites belonging to the top-50 of the world wide web have errors [3], thus suggesting that expensive monitoring mechanisms cannot provide a completely accurate picture of web page reliability. Another study [4] shows that an earlier detection of failures would reduce the majority of customer complaints. Indeed, customer feedback is a key aspect for web page trustworthiness since some server issues might not produce the same effects in all clients.

978-1-5090-3216-7/16/$31.00

We argue that there are still no effective means to easily detect web page problems. A possible approach is to send browsing data to some analytic tool. These tools may handle problems such as nonexistent pages in the domain, but, since they are oriented to advertising and search engine optimization, they typically neglect correct web page display.

To demonstrate that the web currently suffers from a lack of proper monitoring, in Section II, we describe an experiment with 3,000 sites of the top 1,000,000 web sites of the Alexa ranking [5]. The 3,000 samples cover sites from the range 1-1,000, 10,000-20,000 and also from 100,000-200,000. We used the Chrome web browser from two distinct locations in Europe, to ensure realistic access to web pages, and simulate real user interactions. We collected metrics, such as network errors, broken links or JavaScript problems.

We think that the results we present in Section III are noteworthy: 16% of the top 1,000 sites have errors in their web page resources, being this value higher for less popular sites. This demonstrates that no widespread monitoring tool effectively prevents these errors. Then, in Section IV, we proceed to discuss possible client-side monitoring solutions, to complement available monitoring techniques. We compare the client-side solutions, based on their level of intrusiveness for the client. Unfortunately, as we might expect, approaches that can collect more metrics are also much more intrusive. Nevertheless, even non-intrusive light-weight approaches can cover a significant fraction of web errors. These light-weight approaches have the additional benefit of not compromising client security or increasing the complexity of monitoring.

II. Experimental Settings

Before rendering and displaying an HTML (HyperText Markup Language) page, browsers must first fetch the page from a server, using a URL (Uniform Resource Locator). The browser then goes through the page, to build the DOM (Document Object Model), render the corresponding tree and

TABLE I. SOFTWARE USED AND DISTRIBUTION.

Component	Observations	Version
Selenium	selenium-server-standalone jar	2.45.0
Chrome	browser	48.0.2564.103
Chrome	driver	2.21.371461
Xvfb	xorg-server	1.13.3

display it. The browser might need to download other resources referenced in the main page. To fetch these resources, the browser opens several TCP [6] (Transmission Control Protocol) connections to their respective server, either internal or external to the domain. However, each of these resources is subject to failures that might impact the user experience. To analyze the extension of problems, we inspected 3,000 sites, including the most popular ones, and looked for very specific metrics:

- We decomposed network errors into DNS (Domain Name System) errors, in the phase of name lookup; TCP, if the connection crashes or the server is unreachable; and *other* errors.
- For HTTP errors related to resources, we looked to the range 4xx and 5xx. In our experiments, we do not count how many of these errors exist in a single page, but only whether they exist.
- We count broken links, where the server responds with a 4xx or 5xx HTTP code. Again, we only count the number of web sites that have errors in these ranges.
- We also care for other errors related to resources: fonts, style sheets, images and JavaScript. These errors might originate in the network layer, while processing the script (if applied), or in the cancellation of a resource download, e.g., because a change in the page made it unnecessary.

For the sake of doing an online analysis of the web sites, we used Selenium [7], to emulate clients accessing web pages through browsers. We used the Xvfb virtual display emulator for the client machine. This display performs all graphical functions, without actually needing a real screen, thus allowing Selenium to run without a terminal. We wrote a program in Java that ran in the background, attached to this display emulator. This program used Selenium and Chrome, to access a list of web sites. We used a Linux machine, running in the facilities of our department in Portugal and another instance in Hungary, at MTA SZTAKI's laboratories [8]. Clients running from different locations experienced different network connectivities, thus displaying distinct perspectives for the same web page, like resources inaccessible from only one of the locations. Additionally, since programs ran autonomously, with a time lapse of several hours, they occasionally observed different page errors. Table I lists the software we used and the respective versions.

This program used as input a file that we retrieved from Alexa [5], with the top one million ranking sites. We sequentially analyzed 3,000 sites from this file: from pages 1 to 1,000; then from rank 10,000 to 20,000 with steps of 10 (e.g. rank 10,000; 10,010; 10,020;...) and finally, from 100,000 to 200,000 with steps of 100.

Additionally, and one of the most relevant aspects of our work is that we parse the main HTML page, to get all links accessible to the users through web page interaction. We follow and invoke these links, to check if any HTTP error occurs, with error codes 4xx or 5xx, related to client or server errors, respectively. This information is important, because the availability of the links is tightly connected to the utility of the web page. As we shall see the number of broken links is surprisingly high, even in top web sites.

III. RESULTS

In this section, we present the results of our experiment. The experiment took several days to finish, mostly due to the invocation of links associated with each web page. Tables IIa and IIb present the most significant results we got. To conserve space, we only display the results of our client in Portugal, as the client in Hungary got similar results.

In Table IIa, we analyze the number of web pages with network or HTTP errors. This table displays problems with page resources (HTTP 4xx and 5xx), e.g., some image; connection errors (DNS, TCP and other) and broken links, i.e., links that point to resources outside the page and that exhibit some problem. Connection errors are all related to the main HTML page or one of its resources. As we mentioned before, the numbers in the table refer to the total number of sites where we could observe the problem. This means that, for example, in the first line, first column of Table IIa, the number of HTTP 4xx errors in the top 1,000 sites is 161. I.e., 161 sites have one or more resources that are not accessible and return a 4xx error code.

The number of errors is quite high in general, especially in lower raking sites. Differences between the first and the other two rows of the table are blatant, for most metrics. This is true for internal problems and for external links, including network error conditions, which are also much less frequent in the top ranking sites. Most problems come from the external links that tend to break quite often, either with a 4xx or a 5xx error code (right side of the table). However, internal problems (left-side of the table) are arguably more important, as they might result in visible problems in the page layout. Interestingly, as much as 16% of the top-tier sites may suffer from some form of internal problem. This number is even higher for the lower rankings. The same is true for connectivity errors (center of the table). DNS, TCP and other forms of errors are less frequent in major sites. The HTTP 5xx error codes are the only ones where the frequency of problems seems stable across all rankings. This might be due to an inverse relation between complexity and ranking positions (i.e., more complex pages correspond to lower ranking numbers), but a clear demonstration of such hypothesis requires further study. Overall, these results suggest that top-tier sites either have better network connections, or more server resources, or both. We might say the same about the contents themselves, most

TABLE II. NUMBER OF SITES WITH ERRORS - PORTUGAL

(a) Network and HTTP errors

	HTTP 4xx errors	HTTP 5xx errors	DNS errors	TCP errors	Other Network errors	Broken Links 4xx	Broken Links 5xx
$range_{1000}$	161	62	68	27	96	115	65
$range_{10000}$	251	47	122	38	111	182	42
$range_{100000}$	291	51	113	37	114	193	43

(b) Resource errors and Event averages

	Resource Font errors	Resource style sheet error	Resource image error	Resource JS error	Resource JS External	Resource JS Internal	Resource JS Both	DOM Event	Load Event
$range_{1000}$	11	15	131	153	136	13	4	4517	33
$range_{10000}$	16	16	134	189	136	39	14	6097	24
$range_{100000}$	27	27	143	174	103	62	9	6963	19

likely due to significant advantages in the lifecycle of the web pages (one or more among design, development, testing, deployment, and maintenance).

In Table IIb, we show the number of sites that returned at least one resource error, for different types of resources. Resource problems include errors getting fonts, images or processing JavaScript (left side). Regarding JavaScript errors, we split data into external or internal to the web page domain, or both, if errors exist in internal and external resources. We can see that fonts and style sheet resources do not pose major problems for sites. The main offenders to web page reliability are images and JavaScript. Another interesting result is that top-tier sites have more errors in external JavaScript resources. This is an indication that top pages rely more on standard libraries, normally hosted in another domain.

IV. CLIENT-SIDE MONITORING

In this section, we discuss some solutions that might serve to improve web page reliability. We suggest three different options involving different levels of transparency to the client: a stand-alone approach, a browser extension and a JavaScript snippet.

For a stand-alone application (similar to the one we used in Section II), the possibilities are endless. By having total control of the browser and resorting to a testing framework such as Selenium, developers can test pretty much anything. The major disadvantage of this approach is that a customized stand-alone application is not practical or reasonable to install on the clients. Monitoring a site in this way, would therefore be limited to a handful probes controlled by the site owners.

A second approach would be to install a browser extension to get the most important metrics from the web page interaction and send them to a central monitoring site (we refer to some work using browser plugins in Section V). Extensions could bypass some of the security constraints associated with JavaScript. For example, with extensions, it would be possible to have access to the browser APIs and therefore to network logs. This would enable network error collection (DNS, TCP and others). Additionally, the extension could invoke links associated with a web page. Unfortunately, extensions have two setbacks: firstly, it does not look feasible to convince hundreds or thousands of users to install some browser extension that could raise issues, concerning security and privacy; secondly, different extensions should be developed for each different browser, thus entailing a great effort and cost.

As a third approach, site owners might use JavaScript and AJAX in the web pages they serve, to collect error information. By precluding the need for special software, this would rule out the shortcomings of the previous approaches. Furthermore, this would allow for a very simple integration with analytic tools, like Google Analytics [9]. Naturally, this can only work for resources inside the main page, once the browser loads the JavaScript. Collecting different kinds of errors with JavaScript and AJAX raises a number of challenges, but it is still possible, as we can see in the following list:

1) Networking Errors: with JavaScript, one can only infer DNS or TCP errors, using the Resource Timing API [10], as some browsers add entries in the PerformanceResourceTiming array, for resources with network issues.

2) Internal 4xx Errors: it is possible to customize an HTTP 4xx page for this range of errors. As the user is redirected to this page, administrators will receive an alert.

3) Internal 5xx Errors: the browser may analyze the connection and the response times. If the former is different from zero, while the latter is zero, the browser has an indication that it could not retrieve the resource from the server.

4) External Broken Links: we might invoke links from a proxy, using some AJAX solution [11]. This proxy will then invoke the URL and return the request in JSON, thus not breaking cross-domain security.

5) Resource Errors: regarding JavaScript exceptions and console logs, it is possible to use the `window.addEventListener` for *error* events with the `useCapture` argument set to `true` or use `window.onerror` event. This will retrieve the element or script that originated the error, and not the specific error message. We briefly compare the alternatives in Table III.

TABLE III. COMPARISON OF METHODOLOGIES

	Stand-alone Application	Browser Extension	JavaScript Code
Network Problems	Y	Y	Y Indirectly for resources
Broken Links	Y	Y	Y Proxy
JavaScript Errors	Y	Y	Y
Real-world application	Hard to deploy	Security constraints	Easier to scale and deploy

V. RELATED WORK

We divide previous work on web sites reliability into two categories: (i) methods or platforms that collect client metrics; (ii) studies regarding top sites reliability.

Regarding monitoring platforms, Dasu [12] is a client-based software that gathers metrics from different locations. It is limited by the number of hosts that are online, and discards HTML objects from third-party resources or JavaScript errors. In [13], Flach *et al.* use a browser plugin to analyze sites based on rules. This work only focuses on connectivity issues. In [14], authors propose a collaborative approach to detect performance problems. They use a web browser extension on each client and send all information to a central point, for processing. In [15], authors aim to detect user-visible failures, by analyzing Web logs and users' browsing patterns. However, the client may not react to the visible failure (e.g., by leaving the page, or not refreshing it) making this a major concern.

Regarding the reliability of top web sites, in [16] the goal is to collect web page evolution over time. [17] uses a different approach, implementing a web crawler that gathers HTTP, DNS and TCP connection data, to understand in which layer do most of the user-visible page failures occur. This, however, uses a customized crawler, instead of a common browser. In [18], authors gather network information from 80 sites, and analyze the source of the problems. However, recent studies suggest that this pattern of concurrent accesses can significantly change the results observed [19].

Unlike previous work, we consider a very wide range of sites, using a real web browser.

VI. CONCLUSION

The evidences we collected in this paper support the point of view that monitoring remains as a largely unsolved challenge to this day. Even large companies with vast resources fail to provide impeccable, failure-free, web sites. To mitigate this problem, we argue that web site providers must include client-side observations into their monitoring tools. Despite the trade-offs involved, the least intrusive mechanisms can still detect a large number of errors.

ACKNOWLEDGMENT

This work was partially carried out under the project PTDC/EEI-ESS/1189/2014 — Data Science for Non-Programmers, supported by COMPETE 2020, Portugal 2020-POCI, UE-FEDER and FCT.

REFERENCES

[1] "A study about online transactions, prepared for tealeaf technology inc, oct 2005," http://www-01.ibm.com/software/info/tealeaf/, retrieved: April, 2016.

[2] "Causes of failure in web applications (cmupdl-05-109), dec 2005," http://repository.cmu.edu/cgi/viewcontent.cgi?article=1047&context=pdl, retrieved: April, 2016.

[3] *No time for downtime: It managers feel the heat to prevent outages that can cost millions of dollars*, Internet Week, N.807, 3, Internet Week Std., Apr 2000.

[4] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, ser. USITS'03. Berkeley, CA, USA: USENIX Association, 2003, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251460.1251461

[5] "Alexa — top-ranked websites," https://support.alexa.com/hc/en-us/articles/200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites-, retrieved: April, 2016.

[6] J. Postel, "Transmission Control Protocol," RFC 793 (Standard), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[7] "Selenium browser automation," http://www.seleniumhq.org/, retrieved: April, 2016.

[8] Z. Balaton, P. Kacsuk, N. Podhorszki, and F. Vajda, *From Cluster Monitoring to Grid Monitoring Based on GRM*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 874–881. [Online]. Available: http://dx.doi.org/10.1007/3-540-44681-8_121

[9] "Google analytics solutions," https://analytics.google.com/, retrieved: April, 2016.

[10] "Papers — Resource Timing," https://www.w3.org/TR/2016/WD-resource-timing-20160225/, retrieved: March, 2016.

[11] "Webpage — Yahoo Query Language (YQL)," https://developer.yahoo.com/yql/, retrieved: April, 2016.

[12] M. A. Sánchez, J. S. Otto, Z. S. Bischof, D. R. Choffnes, F. E. Bustamante, B. Krishnamurthy, and W. Willinger, "Dasu: Pushing experiments to the internet's edge," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX, 2013, pp. 487–499.

[13] T. Flach, E. Katz-Bassett, and R. Govindan, "Diagnosing slow web page access at the client side," in *Proceedings of the 2013 Workshop on Student Workhop*, ser. CoNEXT Student Workhop '13. New York, NY, USA: ACM, 2013, pp. 59–62. [Online]. Available: http://doi.acm.org/10.1145/2537148.2537160

[14] S. Agarwal, N. Liogkas, P. Mohan, and V. Padmanabhan, "Webprofiler: Cooperative diagnosis of web failures," in *Communication Systems and Networks (COMSNETS), 2010 Second International Conference on*, Jan 2010, pp. 1–11.

[15] W. Li and I. Gorton, "Analyzing web logs to detect user-visible failures," in *Proceedings of the 2010 Workshop on Managing Systems via Log Analysis and Machine Learning Techniques*, ser. SLAML'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1928991.1929000

[16] "Http archive," http://httparchive.org/, retrieved: April, 2016.

[17] C. Vaz, L. Silva, and A. Dourado, "Detecting user-visible failures in web-sites by using end-to-end fine-grained monitoring: An experimental study," in *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, Aug 2011, pp. 338–341.

[18] V. N. Padmanabhan, S. Ramabhadran, S. Agarwal, and J. Padhye, "A study of end-to-end web access failures," in *Proceedings of CoNEXT*, Lisboa, Portugal, December 2006.

[19] J. Sommers and P. Barford, "An active measurement system for shared environments," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 303–314. [Online]. Available: http://doi.acm.org/10.1145/1298306.1298348