Copyright © IEEE.

Citation for the published paper:

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of BTH's products or services Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by sending a blank email message to pubs-permissions@ieee.org.

By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

A Methodology for Evaluating JavaScript Execution Behavior in Interactive Web Applications

Jan Kasper Martinsen #,1 and Håkan Grahn #,2

School of Computing, Blekinge Institute of Technology
SE-371 39 Karlskrona, Sweden
[1] Jan.Kasper.Martinsen@bth.se
[2] Hakan.Grahn@bth.se

***Abstract*—JavaScript has gone from being a mechanism for providing dynamic web pages to an important component of many web applications. Currently one of the most popular type of web applications is so-called social networks, e.g., Facebook, Twitter, and MySpace. However, the workload and execution behavior of JavaScript in this context have not been fully explored or understood.**

In this paper we present a methodology for characterizing the JavaScript execution behavior in interactive web applications using deterministic execution of use cases. Then, we apply this methodology to evaluate a set of social network applications and compare their behavior to a set of established JavaScript benchmarks. Our results confirm previous studies that the execution behavior of social networks differ from established benchmarks. In addition, we identify one novel difference not published before, i.e., the use of anonymous functions in web applications.

I. Introduction

JavaScript was introduced primary as an interpreted prototype based scripting language for web pages, which allowed programmers to add interactivity to web pages [1], [2]. With JavaScript these web pages where given application like behavior. As a result, a number of more or less sophisticated ports of typical desktop-like applications became accessible as web pages. One typical example is Google's mail client Gmail [3]. These web pages are informally known as web applications [4].

One important advantage of web applications is the ease of application distribution. Installing a conventional application usually requires that you are careful so it gets installed onto a correct operating system and on a machine with certain specifications. In contrast, web applications can essentially be accessed and executed directly from any (reasonably modern) web browser.

Social networking [5] has become a popular type of web applications. Facebook seems to be the most popular one and is number two on the Alexa list of the most popular web sites [6], [7]. Many of the entries among top 25 web sites are social networks, e.g., Facebook [8], Twitter [9], and MySpace [10]. Several studies have confirmed the popularity of social networking web applications [11], [12], [13].

Due to the popularity and ease of distribution of web applications, JavaScript has become a very popular programming language. There have also been several approaches to improve the performance of the JavaScript interpreter [14], [15], [16].

To measure and evaluate the performance of JavaScript interpreters (and thereby measuring and quantifying the results of the interpreter optimization), a set of benchmark suites have been proposed [17], [18], [19]. Some of the critique [20], [21] against these suites, is that several of the benchmarks have been ported from benchmarks in, e.g., operating systems and numerical computation research. While these might help us to improve certain aspects of the interpreter, there is a significant risk that the execution behavior of these benchmarks might not fully reflect the JavaScript behavior of real-world web applications, such as social networking web applications.

In this paper, we make three main contributions:

- First, we propose a methodology to measure, characterize, and evaluate the JavaScript execution behavior of interactive social networking web applications such as Facebook, Twitter, and MySpace. We do this by defining a set of use cases that represent typical user operations for the selected web applications. These use cases are then deterministically executed using a scripted and controlled environment.
- Second, our measurements confirm the conclusions from several other studies, e.g., [20], [21], that there are significant differences in the execution behavior between real-world web applications and established benchmarks.
- Third, we identify one unpublished significant difference between web applications and the established benchmarks, i.e., the use of anonymous functions.

The rest of the paper is organized as follows. Section II presents some background and previous work, and then Section III presents the benchmarks and web applications that we use. In Section IV, we present our methodology to do workload characterization of interactive web applications. Section V presents our measurement results. Section VI discusses some directions for future work, and finally, we conclude our findings in Section VII.

II. Background

A. JavaScript and web applications

JavaScript [1], [2] was introduced by Netscape in 1995 as a way to allow web developers to add dynamic functionality to web pages that were executed on the client side. The purposes of the functionality were typically to validate input

forms and other user interface related tasks. JavaScript has gained momentum over the years, particularly due to its ease of deployment and the increasing popularity of certain web applications, e.g., Gmail [22]. We have found that almost all of the first 100 sites in the Alexa-top sites list [6] include some JavaScript functionality.

JavaScript is a dynamically typed, object-based scripting language with run-time evaluation. The execution of a JavaScript program is done in a JavaScript engine [23], [24], [25], i.e., an interpreter/virtual machine that parses and executes the JavaScript program. Due to the popularity of the language, there have been multiple approaches to increase the performance of the JavaScript engines, through well-known optimization techniques such as just-in-time (JIT) compilation techniques, fast property access, and efficient garbage collection [15], [23].

B. Previous work

With the increasing popularity of web applications, it has been suggested that the web browser could serve as a general platform for applications in the future. This would imply that JavaScript needs increased performance. Further, it also mean that one would need to look deeper into the workload of actual web applications. This process is in its early phases, but there are several examples of interesting work [8], [26]. Two concurrent studies [20], [21] explicitly compare the JavaScript execution behavior of web applications as compared to existing JavaScript benchmark suites.

The study by Ratanaworabhan et al. [20] is one of the first studies that compares JavaScript benchmarks with real-world web applications. They instrumented the Internet Explorer 8 JavaScript runtime in order to get their measurements. Their measurements were focused on two areas of the JavaScript execution behavior, i.e., (i) functions and code, and (ii) events and handlers. Based on the results, they conclude that existing JavaScript benchmarks are not representative of many real-world web applications and that conclusions from benchmark measurements can be misleading. Examples of important differences include different code sizes, web applications are often event-driven, no clear hotspot function in the web applications, and that many functions are short-lived in web applications. They also studied memory allocation and object lifetimes in their study.

The study by Richards et al. [21] also compares the execution behavior of JavaScript benchmarks with real-world web applications. In their study, they focus on the dynamic behavior and how different dynamic features are used. Examples of dynamic features evaluated are prototype hierarchy, program size, object properties, and hot loops (hotspots). They conclude that the behavior of existing JavaScript benchmarks differ on several of these issues from the behavior of real web applications.

III. Benchmarks and web applications

A. JavaScript benchmarks

There exist three established JavaScript benchmark suites: V8 [17], Dromaeo [18], and Sunspider [19]. The applications in these benchmark suites generally fall into two different categories: (i) testing of a specific functionality, e.g., string manipulation or bit operations, and (ii) ports of already existing benchmarks that are used extensively for other programming environments [27]. The different applications found in the three benchmark suites are listed in Table I.

TABLE I
A summary of the benchmark suites used in this paper.

Benchmark suite	Applications
Dromaeo	3d-cube, core-eval, object-array, object-regexp, object-string, string-base64
V8	crypto, deltablue, earley-boyer, raytrace, richards
SunSpider	3d-morph, 3d-raytrace access-binary-trees, access-fannkuch, access-nbody, access-nsieve bitops-3bit-bits-in-byte, bitops-bits-in-byte, bitops-bitwise-and, bitops-nsieve-bits controlflow-recursive crypto-aes, crypto-md5, crypto-sha1 date-format-tofte, date-format-xparb math-cordic, math-partial-sums, math-spectral-norm regexp-dna string-fasta, string-tagcloud, string-unpack-code, string-validate-input

For example, the benchmarks Raytrace, Richards, Deltablue, and Earley-Boyer are included in the V8 benchmark suite. Raytrace is a well-known computational intensive graphical algorithm for rendering scenes [28]. Richards simulates an operating system task dispatcher, Deltablue is a constraint solver, and Earley-Boyer is a type theorem prover benchmark. In contrast, the Dromaeo benchmarks test specific JavaScript language features.

Typical for the established benchmarks is that they often are problem oriented, meaning that the purpose of the benchmark is to accept a problem input, solve this certain problem, and then end the computation. This eases measurements, gives the developer full control over the benchmarks, and increases the reproducibility.

B. Social networking web applications

There exists many so-called social networking web applications [29], where Facebook [8] is the most popular one [6], [12]. There are even examples of countries where half of the population use Facebook to some extent during the week [11]. The purpose and usage of social web applications might have many facets. However, the key element for a social networking application to be successful is to have a certain critical mass of users.

The users of a social networking web application can locate and keep track of friends or people that share the same interests. This set of friends represents each user's private

network, and to maintain and expand a user's network, a set of functionalities is defined. For example, users can create petitions to vote for a certain cause, while other users can play video games where the final 'score' is compared with other friends in their own networks.

In this paper we study the social networking web applications Facebook, Twitter [9], and MySpace [10]. In a sense, Facebook seems to be a general purpose social networking web application, with a wide range of different functionality. Further, Facebook also has the largest number of users.

Twitter is for writing small messages, so called "tweets", which are restricted to 160 characters (giving a clear association to SMS). The users of Twitter are able to follow other people's tweets, and for instance add comments in form of tweets to their posts.

MySpace seems to be especially coined at musicians, that wish to share or obtain music. Through MySpace the users can upload music, which they in turn distribute to other MySpace users. Users are also able to write comments and search for other users with similar music taste.

IV. A METHODOLOGY FOR EVALUATING JAVASCRIPT EXECUTION BEHAVIOR

While the benchmarks have a clear purpose, with a clearly defined start and end state, interactive social networking web applications behave more like operating system applications, where the user can perform a selected number of tasks. As long as the web application is viewed by the user, it remains active and performs a set of underlying tasks.

When measuring and evaluating application or system behavior, as well as when defining benchmarks, two of the most important things are: (i) the application/benchmark should be *representative* and (ii) the measurements should be *reproducible*. How representative existing JavaScripts benchmark suites are for real-world web applications have been addressed in, e.g., [20], [21], and in this paper we identify some additional differences. However, the issue of reproducibility of web application behavior measurements have not been addressed in previous studies.

A. Representative behavior

In order to mimic a representative use and behavior of social network web applications, we have defined a set of use cases. Each use case has a clear start and end state. These use cases are intended to give a realistic idea of the actual workload in web applications and also provide repeatability of the measurements. The use cases that we have designed represent common user behavior in Facebook, Twitter, and Myspace, rather than exhausting JavaScript execution.

Figure 1 shows the different use cases that we have defined for Facebook, Twitter, and MySpace, respectively. All use cases start with the user login. Then, the user has multiple options.

For Facebook, the users login to the system, then the user searches for an old friend, which the user in turn finds. When the user finds this old friend, the user marks him as a "friend", an operation where the user needs to ask for confirmation from the friend to make sure that he actually is the same person. This operation is a typical example of an use case, which in turn is composed of several sub use cases: 0 `-login/home`, 0.3 `-find friend`, 0.3.1 `-add friend`, and 0.3.1.0 `-send request`, as shown in Figure 1a.

All use cases start with the login case, and we recognize an individual operation, such as 0.3.1 `-add friend` as a sub use case, though it must complete previous use cases. Further, we do allow use cases that goes back and forth between use cases. For example in Figure 1b, if we want to both choose the option 0.1.0 `-follow` and 0.1.1 `-mention`, then we would need to visit the following sub use cases: 0 `-login/home`, 0.1 `-find person`, 0.1.0 `-follow`, 0.1 `-find person`, and 0.1.1 `-mention`.

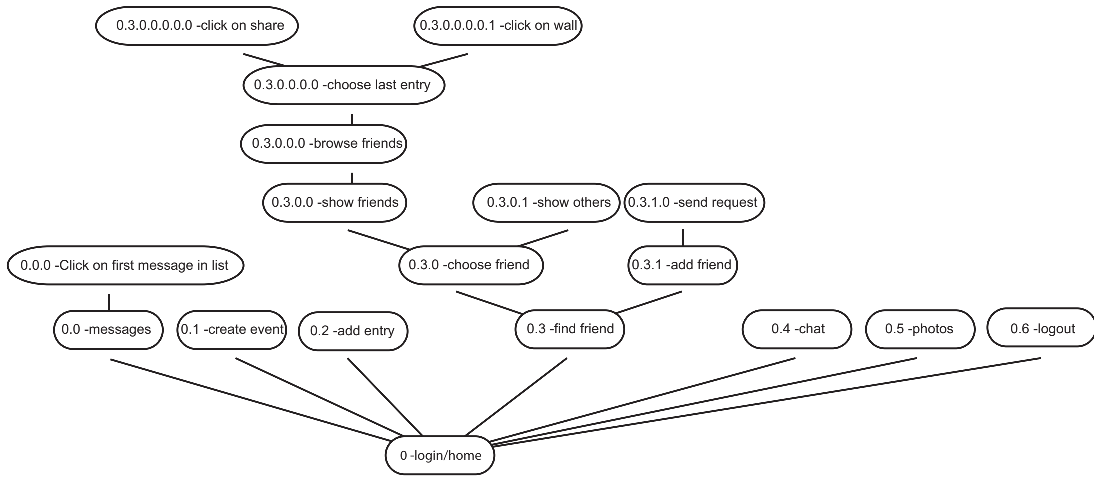
B. Reproducible behavior

To enhance reproducibility, we use the AutoIt scripting environment [30] to automatically execute the various use cases in a controlled fashion. As a result, we can make sure that we spend the same amount of time on the same or similar operations, such as to type in a password or click on certain buttons. This is suitable for the selected use cases. However for certain operations, several social networks employ various web crawling countermeasures, e.g., through CAPTCHA [31] or restricts the number of login attempts.

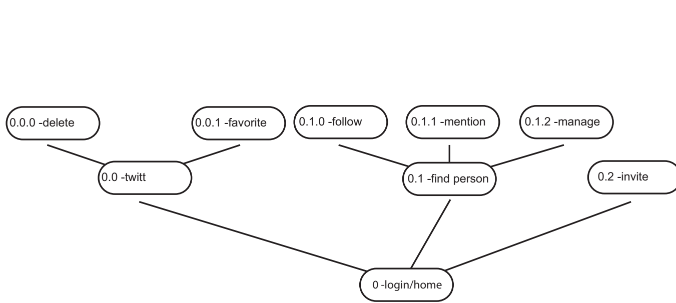
We discovered by successively executing the same use case (the 0 `-login/home` Facebook use case) 10 times, that there is no guarantee that the executed JavaScript code would be identical in all the cases, even though the usage would be identical. Since JavaScript has a function such as `eval`, we can easily create script that dynamically generates JavaScript code. We have found that a certain fraction of the function names is unique for repetitions of identical cases, which suggests that changes occur between reloads or as a result of session specific code through AJAX calls [32]. We also found that the number of function calls, and the number of functions that are called vary for identical cases as shown in Figure 2.

A large fraction of these function calls is anonymous function calls (we will come back this issue in Section V-B). Thus, we could argue that many of them were dependent on the input data, which could potentially change through AJAX calls [32]. However, at the same time, not all of them are anonymous calls for all of the 10 successive calls with functions that had unique function names.

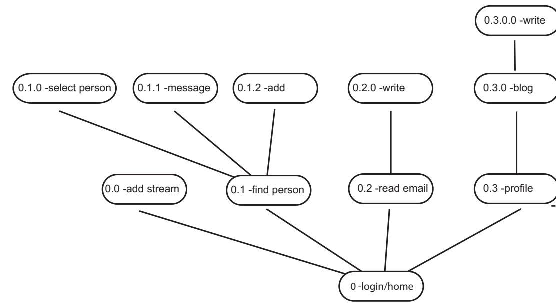
To remedy this problem and simplify later analysis, we have done the following. For example, the two cases in Figure 1c (0 `-login/home`, 0.1 `-find person`, 0.1.1 `-message`) and (0 `-login/home`, 0.1 `-find person`, 0.1.1 `-add`) both share the same actions (0 `-login/home`, 0.1 `-find person`), which we from now on denote as a sub case. However, as we saw above, the JavaScript execution for this sub use case might be radically different for the two cases. To simplify later analysis we have created two



a) Facebook



b) Twitter



c) MySpace

Fig. 1. Use cases to characterize the JavaScript workload of a) Facebook, b) Twitter, and c) MySpace.

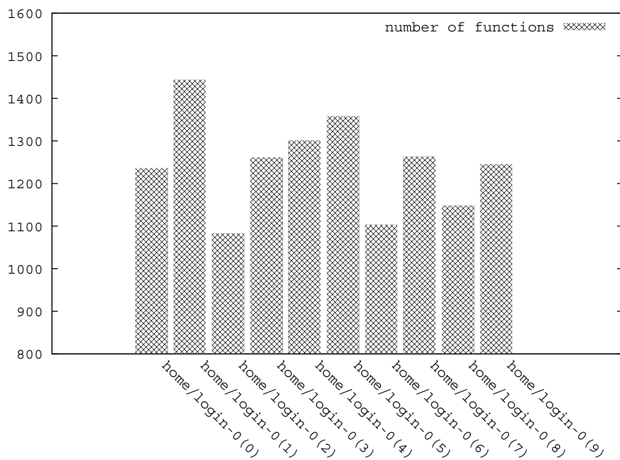


Fig. 2. Total number of functions for each of the 10 repetitions of the same case.

countermeasures, to make sure that the executed JavaScript code will be less different between the two cases.

By using the WebKit [33] environment, we have extracted and created a local copy of certain sub use cases. In more detail, when these parts are extracted, we first login to, e.g., Facebook, where there seems to be some so-called session variables that needs to be set. Then we open up the local copy, and profile this until we reach the point where we can select two different paths (e.g., `0.1.0 -add` or `0.1.1 -message`). We have found that this approach works well for some of the cases, but not for all of them. If this first approach does not work, we have used the following mechanism. We have instrumented and repeatedly executed the common use cases 10 times (e.g., `0 -login/home`, `0.1 -find person`) and then used the average of the common JavaScript execution profile for the measurements.

C. Experimental environment

To do the actual profiling we have used the Firebug v1.5.4 profiling tool [34] running on a freshly installed Windows XP. Firebug runs on a custom compiled version of Firefox v3.6, which is able to automatically record executed JavaScript code as well as some simple instrumentation. Firebug reports a number of issues, and for JavaScript code it reports, e.g., the name of the JavaScript functions called, the amount of time each function is executed, the percentage of the total execution time the function uses, and the amount of time the function uses for execution. To extract use cases we have used a custom Ubuntu installation with WebKit [25].

V. MEASUREMENT RESULTS

A. Distribution of function calls and execution time

In order to understand the relative impact on the execution time of each function call, we have collected execution statistics of how many times each function is called and how much it contributes to the total execution time.

We have normalized the execution time for all the function call entries. To understand how these relates to functions that accounts for most of the applications execution time, we created a histogram for the execution time for both the benchmarks and the first 100 sites on the Alexa top sites list (Figure 3 and Figure 4). This histogram is divided into 10 categories, where each category accounts for the number of function calls that contributes to either $0-9\%$, $10-19\%$, $20-29\%$, $30-39\%$, $40-49\%$, $50-59\%$, $60-69\%$, $70-79\%$, $80-89\%$, or $90-99\%$ of the execution time.

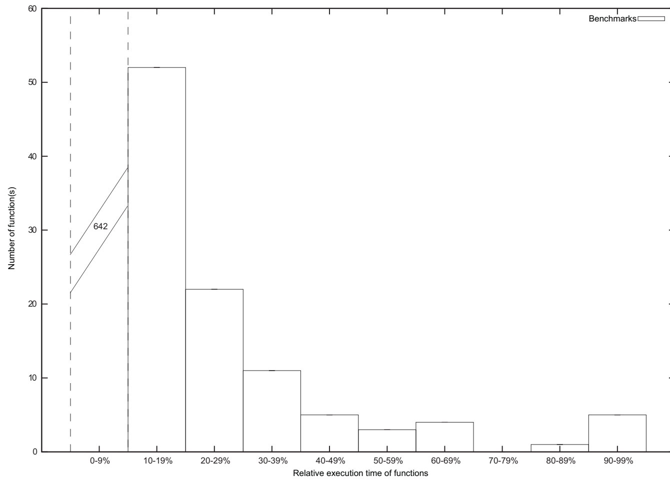


Fig. 3. Histogram over the number of functions that contributes to a certain percentage of the total execution time for the benchmarks.

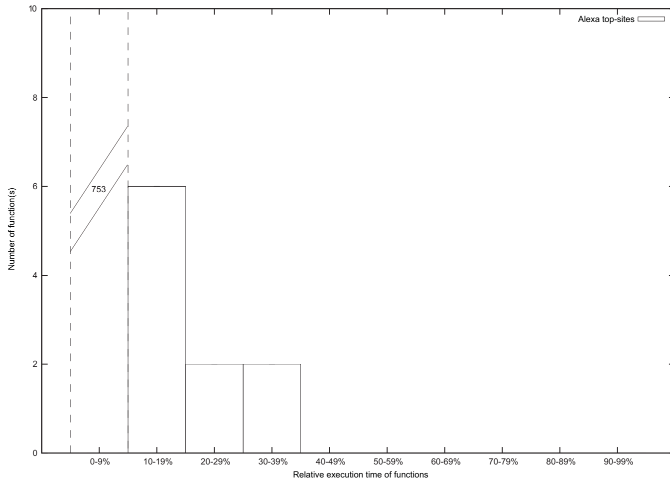


Fig. 4. Histogram over the number of functions that contributes to a certain percentage of the total execution time for the Web Applications.

We see in Figure 3 and Figure 4 that both the benchmarks and the Web Applications have a large number of functions in the $0-9\%$ category, which indicate that there is a very large number of small functions executed. In Figure 3 we see that for the benchmarks, the workload is divided into most of the columns in the histogram. Especially, we find that there are a number of functions that account for more than 80% of the execution time, i.e., a clear hot spot function exists. In contrast, we see in Figure 4 that the execution time of the Web Applications only uses the first four categories. This means that no function dominates the execution time in the web applications, i.e., no hot spot exists in the code. In web applications, the workload seems to be more evenly distributed, and *no* JavaScript function contributes to more than at most 39% of the total execution time.

In order to analyze the relative execution time for social network web applications, we show the relative fraction of execution time per function and the relative number of function calls per function for Facebook, Twitter, and MySpace in Figure 5, Figure 6, and Figure 7, respectively.

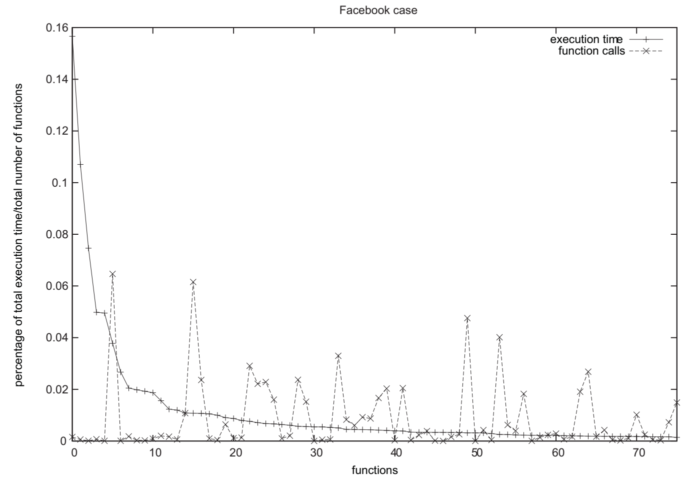


Fig. 5. Relative number of function calls and the relative amount of execution time spent in each function for the Facebook use cases.

Our results show a high variance between the number of times a function is called and its contribution to the execution time for Facebook, Twitter, and MySpace. This indicates that there is a high variance in the execution times of individual functions. For example, we found that for the Facebook use case, only 14 out of 75 function calls have the same relative number of function calls as the relative fraction of the execution time.

B. Anonymous function behavior

A previous study of Facebook reveals that a large number of anonymous function calls are made [35]. However, the same study reveals that these functions do not account for a large fraction of the total execution time. In Figure 8, Figure 9, and Figure 10, respectively, we have measured (i) the number of unique anonymous functions relative to the total number of

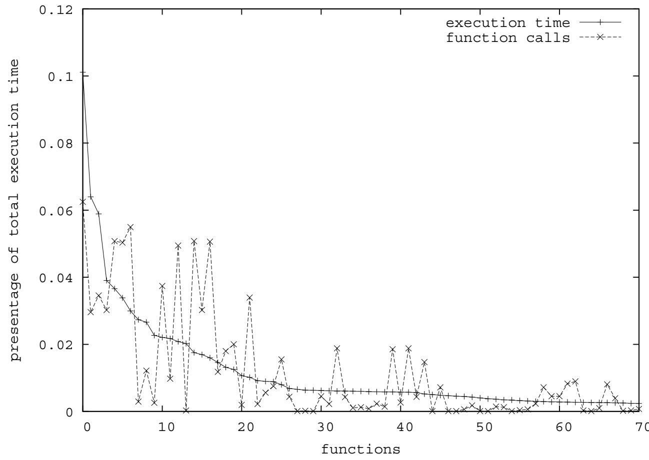


Fig. 6. Relative number of function calls and the relative amount of execution time spent on each function for the Twitter use cases.

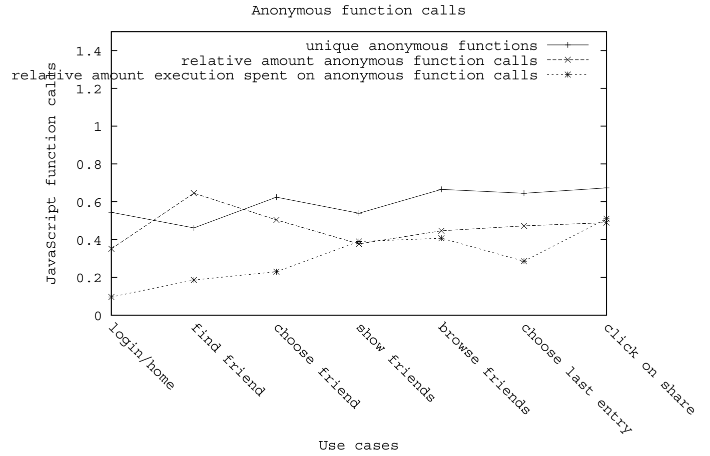


Fig. 8. Anonymous function calls for Facebook.

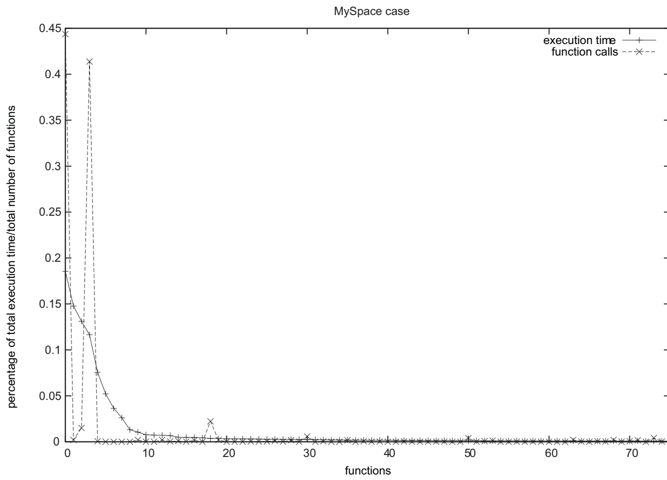


Fig. 7. Relative number of function calls and the relative amount of execution time spent in each function for the MySpace use cases.

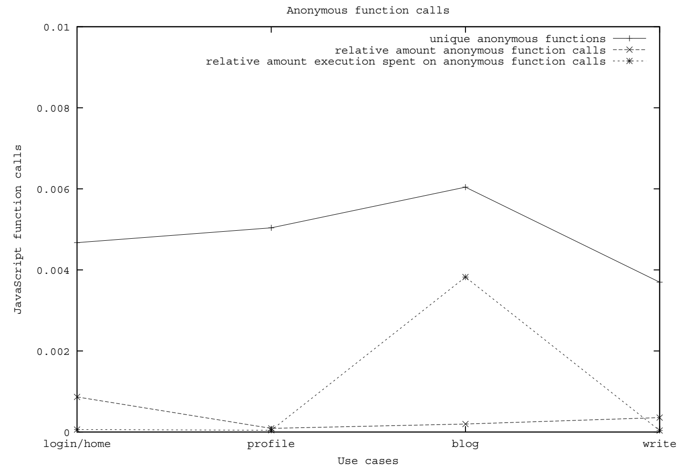


Fig. 9. Anonymous function calls for Twitter

unique functions, (ii) the total number of anonymous function calls relative to the total number of function calls, and (iii) the total execution time spent in anonymous functions relative to the total execution time for the use cases defined in Figure 1.

We see in Figure 8 that the number of unique anonymous function calls as well as the number of calls increase slightly as we complete the use case (25% and 31%). However, the execution time increases with a factor 5 between the login sub use case and the final use case. At the final sub use case, the anonymous function workload accounts for over half of the total workload.

However, from Figure 9 and Figure 10 we see that both Twitter and MySpace use fewer anonymous function calls than Facebook does. They have only a small number of unique anonymous functions, a small number of anonymous function calls and those functions that are called do only account for a minor part of the execution time.

In comparison, our results show that anonymous functions are used to a very little degree in the benchmarks. For instance, both the V8 benchmarks Raytracer and Earley-Boyer had both over 40000 function calls, but only 3 of them were anonymous functions. In comparison, in Figure 8, for the use case where we search for friends, over 40% of the function calls were anonymous.

VI. Discussion and future work

As pointed out in [20], [21] one could argue that the workload of JavaScript in a web application setting is not well represented by the established benchmarks. These studies and our own results suggest that the behavior of web applications is significantly different than for traditional programs, e.g., by being more event-driven and by utilizing dynamic updates of code at runtime. The lack of iterative constructs also could suggest that traditional JIT like optimization could be less effective for web applications than for the established benchmarks. The results in [36] show that just-in-time compilation often increases the execution time for many common web applications.

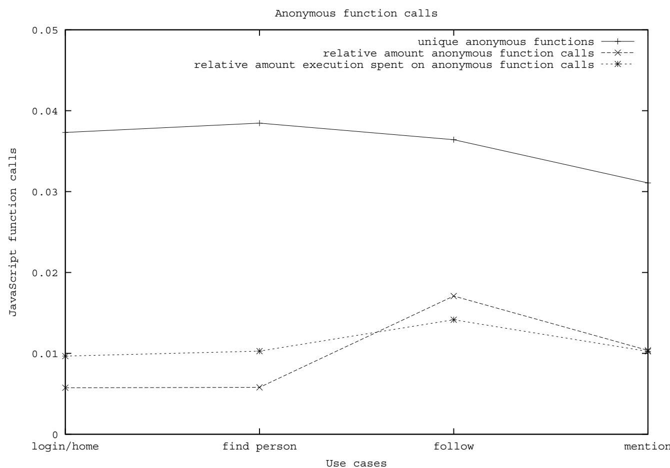


Fig. 10. Anonymous function calls for MySpace

However, we still need to be careful with our conclusions, and we will outline a couple of reasons why such a care is needed. Web applications is a fairly new concept that came together with the popularization of a set techniques known as Web2.0 and the possibility for dynamic updates. Our tests reveal the importance of such technologies in, e.g., Facebook. However, at the same time, there is a trend where richer multimedia possibilities are starting to be offered to the web through technologies such as for instance WebGL. For some of the workloads found in multimedia applications using 3D graphics, some of the current benchmarks would be more relevant.

Our study as well as other similar studies [20] are based on web applications. However, JavaScript has turned out to be a popular embedded language for multiple applications, e.g., an embedded language in the FireFox web browser, so its usage is in no way restricted to only web applications. Further, some of the workloads in web applications are spawned from functionalities that are not strictly a part of the JavaScript specification, but rather a part of the functionalities of the web browser.

Either way, JavaScript has some rather unique programming constructs, and as our test shows, a functionality such as anonymous functions are used extensively. These kind of functions are spawned from a different field, and are usually not available in other programming languages. That does, however, not mean that they are not powerful. The established benchmarks address this issue only to a minor extent, and future benchmarks ought to take the usage of anonymous functions into account. We also suggest that there should be put some effort into resembling event-driven programs.

VII. Concluding remarks

In this paper, we have described a methodology to characterize the workload behavior of interactive web applications that are written in JavaScript. As part of the methodology, we have defined a number of use cases for three popular social networking applications, i.e., Facebook, Twitter, and MySpace. Further, we use an automatic scripting environment in order to enhance the repeatability of the measurements.

Our characterization of the workload behavior of social networking web applications shows some interesting differences as compared to the workload behavior of established JavaScript benchmarks. First, we have found that the correlation between the relative number of function calls and the relative amount of execution time spent in each function is significantly lower for web applications than for the benchmarks. Second, the studied web applications have a significantly larger amount of anonymous functions and function calls than the established benchmarks. Finally, the established benchmarks often contain loop constructs that account for a significant portion of the total execution time, while such hot spots have not been observed in the web applications. In contrast, web applications seem to be based on event-driven programming techniques.

Acknowledgment

This work was partly funded by the Industrial Excellence Center EASE - Embedded Applications Software Engineering, (http://ease.cs.lth.se).

References

[1] D. Flanagan, *JavaScript: The Definitive Guide, 5th edition*. O'Reilly Media, 2006.

[2] JavaScript, "http://en.wikipedia.org/wiki/JavaScript," 2010.

[3] C. A. Sanders, "Coming down the e-mail mountain, blazing a trail to gmail," in *SIGUCCS '08: Proceedings of the 36th annual ACM SIGUCCS conference on User services conference*. New York, NY, USA: ACM, 2008, pp. 101–106.

[4] J. Wright and J. Dietrich, "Survey of existing languages to model interactive web applications," in *APCCM '08: Proc. of the 5th Asia-Pacific Conf. on Conceptual Modelling*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2008, pp. 113–123.

[5] R. Arjan, U. Pfeil, and P. Zaphiris, "Age differences in online social networking," in *CHI '08: CHI '08 extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM, 2008, pp. 2739–2744.

[6] Alexa, "Top 500 sites on the web," 2010, http://www.alexa.com/topsites.

[7] G. R. Notess, "Alexa: Web archive, advisor, and statistician," *Online*, vol. 22, no. 3, pp. 29–30, 1998.

[8] A. Nazir, S. Raza, and C.-N. Chuah, "Unveiling Facebook: A measurement study of social network based applications," in *IMC '08: Proc. of the 8th ACM SIGCOMM Conf. on Internet Measurement*. New York, NY, USA: ACM, 2008, pp. 43–56.

[9] B. Krishnamurthy, P. Gill, and M. Arlitt, "A few chirps about twitter," in *WOSP '08: Proc. of the First Workshop on Online Social Networks*. New York, NY, USA: ACM, 2008, pp. 19–24.

[10] U. Pfeil, R. Arjan, and P. Zaphiris, "Age differences in online social networking - a study of user profiles and the social capital divide among teenagers and older users in MySpace," *Comput. Hum. Behav.*, vol. 25, no. 3, pp. 643–654, 2009.

[11] E. Eldon, "Facebook used by the most people within Iceland, Norway, Canada, other cold places," 2009, http://www.insidefacebook.com/2009/09/25/facebook-used-by-the-most-people-within-iceland-norway-canada-other-cold-places/.

[12] Facebook, "Facebook statistics," 2010, http://www.facebook.com/press/info.php?statistics.

[13] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *IMC '07: Proc. of the 7th ACM SIGCOMM Conf. on Internet Measurement*. New York, NY, USA: ACM, 2007, pp. 29–42.

[14] A. Gal, "TraceMonkey vs V8," 2008, http://andreasgal.wordpress.com/2008/09/03/tracemonkey-vs-v8/.

[15] A. Gal, B. Eich, M. Shaver, D. Anderson, D. Mandelin, M. R. Haghighat, B. Kaplan, G. Hoare, B. Zbarsky, J. Orendorff, J. Ruderman, E. W. Smith, R. Reitmaier, M. Bebenita, M. Chang, and M. Franz, "Trace-based just-in-time type specialization for dynamic languages," in *PLDI '09: Proc. of the 2009 ACM SIGPLAN Conf. on Programming Language Design and Implementation*. New York, NY, USA: ACM, 2009, pp. 465–478.

[16] J. Resig, "http://ejohn.org/blog/tracemonkey/," 2008.

[17] Google, "V8 benchmark suite - version 5," 2010, http://v8.googlecode.com/svn/data/benchmarks/v5/run.html.

[18] Mozilla, "Dromaeo: JavaScript performance testing," 2010, http://dromaeo.com/.

[19] WebKit, "SunSpider JavaScript Benchmark," 2010, http://www2.webkit.org/perf/sunspider-0.9/sunspider.html.

[20] P. Ratanaworabhan, B. Livshits, and B. G. Zorn, "JSMeter: Comparing the behavior of JavaScript benchmarks with real web applications," in *WebApps'10: Proc. of the 2010 USENIX Conf. on Web Application Development*. Berkeley, CA, USA: USENIX Association, 2010, pp. 3–3. [Online]. Available: http://portal.acm.org/citation.cfm?id=1863166.1863169

[21] G. Richards, S. Lebresne, B. Burg, and J. Vitek, "An analysis of the dynamic behavior of JavaScript programs," in *PLDI '10: Proc. of the 2010 ACM SIGPLAN Conf. on Programming Language Design and Implementation*. New York, NY, USA: ACM, 2010, pp. 1–12.

[22] E. Schonfeld, "Gmail grew 43 percent last year. AOL mail and Hotmail need to start worrying," 2009, http://techcrunch.com/2009/01/14/gmail-grew-43-percent-last-year-aol-mail-and-hotmail-need-to-start-worrying/.

[23] Google, "V8 JavaScript Engine," 2010, http://code.google.com/p/v8/.

[24] Mozilla, "What is SpiderMonkey?" 2010, http://www.mozilla.org/js/spidermonkey/.

[25] WebKit, "The WebKit open source project," 2010, http://www.webkit.org/.

[26] A. A. Andrews, J. Offutt, C. Dyreson, C. J. Mallery, K. Jerath, and R. Alexander, "Scalability issues with using FSMWeb to test web applications," *Inf. Softw. Technol.*, vol. 52, no. 1, pp. 52–66, 2010.

[27] O. Agesen, "GC points in a threaded environment," Sun Microsystems, Inc., Mountain View, CA, USA, Tech. Rep., 1998.

[28] A. Watt, *3D Computer Graphics*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1993.

[29] Wikipedia, "List of social networking websites," 2010, http://en.wikipedia.org/wiki/List_of_social_networking_websites.

[30] J. Brand and J. Balvanz, "Automation is a breeze with autoit," in *SIGUCCS '05: Proc. of the 33rd Annual ACM SIGUCCS Conf. on User services*. New York, NY, USA: ACM, 2005, pp. 12–15.

[31] P. Golle, "Machine learning attacks against the Asirra CAPTCHA," in *CCS '08: Proc. of the 15th ACM Conf. on Computer and Communications Security*. New York, NY, USA: ACM, 2008, pp. 535–542.

[32] A. A. Dahlan and T. Nishimura, "Implementation of asynchronous predictive fetch to improve the performance of Ajax-enabled web applications," in *IIWAS '08: Proc. of the 10th Int'l Conf. on Information Integration and Web-based Applications & Services*. New York, NY, USA: ACM, 2008, pp. 345–350.

[33] M. Stringer, J. A. Rode, E. F. Toye, A. F. Blackwell, and A. R. Simpson, "The WebKit tangible user interface: A case study of iterative prototyping," *IEEE Pervasive Computing*, vol. 4, no. 4, pp. 35–41, 2005.

[34] FireBug, "FireBug, JavaScript profiler," 2010, http://getfirebug.com.

[35] J. K. Martinsen and H. Grahn, "A comparative evaluation of the execution behavior of JavaScript benchmarks and real-world web applications," in *Poster proc. of the 28th Int'l Symp. on Computer Performance, Modeling, Measurements and Evaluation (Performance-2010)*, Nov. 2010, pp. 27–28.

[36] J. K. Martinsen, H. Grahn, and A. Isberg, "A comparative evaluation of JavaScript execution behavior," in *Proc. of the 11th Int'l Conference on Web Engineering (ICWE 2011)*, June 2011.