A Comparative Evaluation of JavaScript Execution Behavior

Jan Kasper Martinsen[1], Håkan Grahn[1], and Anders Isberg[2]

[1] Blekinge Institute of Technology, Karlskrona, Sweden,
{jan.kasper.martinsen,hakan.grahn}@bth.se
[2] Sony Ericsson Mobile Communications AB, Lund, Sweden,
Anders.Isberg@sonyericsson.com

Abstract. JavaScript is a dynamically typed, object-based scripting language with runtime evaluation. It has emerged as an important language for client-side computation of web applications. Previous studies indicate some differences in execution behavior between established benchmarks and real-world web applications.
Our study extends previous studies by showing some consequences of these differences. We compare the execution behavior of four application classes, i.e., four JavaScript benchmark suites, the first pages of the Alexa top-100 web sites, 22 use cases for three social networks, and demo applications for the emerging HTML5 standard. Our results indicate that just-in-time compilation often increases the execution time for web applications, and that there are large differences in the execution behavior between benchmarks and web applications at the bytecode level.

1 Introduction

The World Wide Web is an important platform for many applications and application domains, e.g., social networking and electronic commerce. These type of applications are often referred to as web applications. Social networking web applications, such as Facebook, Twitter, and Blogger, have turned out to be popular, being in the top-25 web sites on the Alexa list [1]. All these three applications use JavaScript extensively. Further, we have found that 98 of the top-100 web sites use JavaScript to some extent.

JavaScript is a dynamically typed, object-based scripting language with runtime evaluation. The execution of a JavaScript program is done in a JavaScript engine [6], and several benchmarks have been proposed to evaluate its performance. However, previous studies show that the execution behavior differs between benchmarks and real-world web applications in several aspects [4, 5].

We compare the execution behavior of four different application classes, i.e., (i) four established JavaScript benchmark suites, (ii) the start pages for 100 most visited web applications, (iii) 22 different use cases for popular social networks, and (iv) 109 demo applications for the emerging HTML5 standard. We extend previous studies with *three main contributions*: An extension of the execution behavior analysis with reproducible use cases of social network applications and

HTML5 applications, we show that just-in-time compilation often *increases* the execution time for web applications, and we provide a detailed instruction mix measurement and analysis. A more comprehensive set of results is found in [3].

2 Experimental methodology

The experimental methodology is thoroughly described in [2]. We have selected a set of 4 application classes consisting of the first page of the 100 most popular web sites, 109 HTML5 demos from the JS1K competition, 22 use cases from three popular social networks (Facebook, Twitter, and Blogger), and a set of 4 benchmarks for measurements. We have measured and evaluated two aspects: the execution time with and without just-in-time compilation, and the bytecode instruction mix for different application classes. The measurements are made on modified versions of the GTK branch of WebKit (r69918) and Mozilla Firefox with the FireBug profiler.

Web applications are highly dynamic and the JavaScript code might change from time to time. We improve the reproducibility by modifying the test environment to download and re-execute the associated JavaScript locally (if possible). For each test an initial phase is performed 10 times to reduce the chances of execution of external JavaScript code.

Another challenge is the comparison between the social networking web applications and the benchmarks, since the web applications have no clear start and end state. To address this, we defined a set of use cases based on the behavior of friends and colleagues, and from this we created instrumented executions with the Autoit tool.

We modified our test environment in order to enable or disable just-in-time compilation. During the measurements, we executed each test case and application with just-in-time compilation disabled and enabled 10 times each, and selected the best one for comparison. We used the following relative execution time metric to compare the difference between just-in-time-compilation (JIT) and no-just-in-time-compilation (NOJIT):

$$T_{exe}(JIT)/T_{exe}(NOJIT) \geq 1$$

3 Experimental results

3.1 Comparison of the effect of just-in-time compilation

In Figure 1 we present the relative execution time for the Alexa top-100 web sites, the first 109 JS1K demos, 24 SunSpider benchmarks, 6 Dromaeo benchmarks, and 10 JSBenchmarks. The results show that for 58 out of the top-100 web sites and for 50 out of 109 JS1K demos, JIT *increases* the execution time. When JIT fails, it increases the execution time by a factor of up to 75. In contrast, just-in-time compilation decreases the execution time for almost all benchmarks. In general, the penalty of a unsuccessful JIT compilation is larger in real-world web applications. However, the gain is also much larger for the largest decrease in execution time with a JIT compilation.

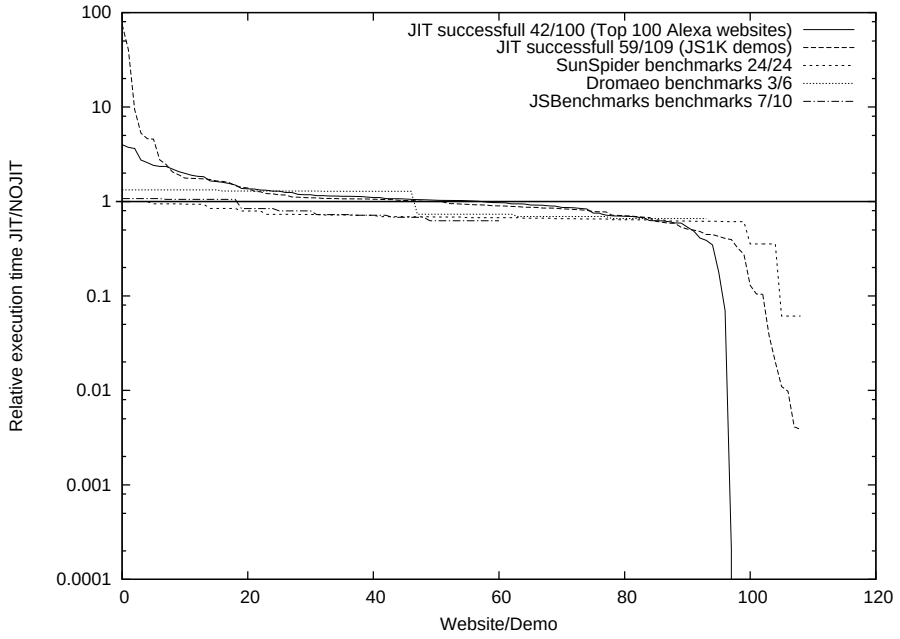

Fig. 1. Relative execution time $T_{exe}(JIT)$ / $T_{exe}(NOJIT)$ for the top-100 Alexa web sites, the first 109 JS1K demos, 24 Sunspider benchmarks, 6 Dromaeo benchmarks, and 10 JSBenchmarks. A value larger than 1 means that JIT compilation *increases* the execution time.

3.2 Comparison of bytecode instruction usage

We have measured the bytecode instruction mix for the selected benchmarks and for the Alexa top-100 sites. Figure 2 shows the results for the top-100 sites and the SunSpider benchmarks since they differ the most.

Our results show that the arithmetic/logical instructions and bit operations are used significantly more in the SunSpider benchmarks than in the web applications. We also find that general branch/jump instructions are more common in web applications, while loop instructions are more common in the benchmarks. The large number of `jmp` instructions indicates the importance of function calls in web applications.

4 Conclusions

Our most important results are that just-in-time compilation often *increases* the execution time of web applications and that the execution behavior of the benchmarks differs significantly from the web applications. The instruction mix gives an indication why loop-based optimization techniques often fails for web applications. These results call for alternative optimization techniques for web applications as well as benchmarks that better represents their workload.

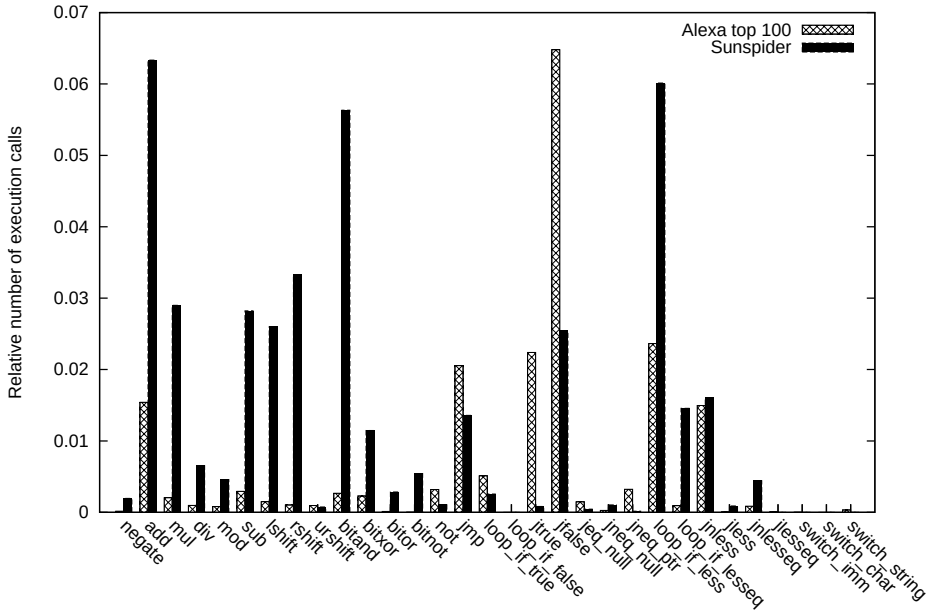

Fig. 2. Branch, jump, and arithmetic/logical related bytecode instructions for the Alexa top-100 web sites and the SunSpider benchmarks.

Acknowledgments

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

References

1. Alexa. Top 500 sites on the web, 2010. `http://www.alexa.com/topsites`.
2. J.K. Martinsen and H. Grahn. A methodology for evaluating JavaScript execution behavior in interactive web applications. In *Proc. of the 9th ACS/IEEE Int'l Conf. on Computer Systems And Applications*, pages XX–YY, December 2011.
3. J.K. Martinsen, H. Grahn, and A. Isberg. Evaluating four aspects of JavaScript execution behavior in benchmarks and web applications. Technical Report No. 2011:01, Blekinge Institute of Technology, Sweden, 2011.
4. 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*, pages 3–3, 2010.
5. 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*, pages 1–12, 2010.
6. WebKit. The WebKit open source project, 2010. `http://www.webkit.org/`.