

# Instrumenting V8 to Measure the Efficacy of Dynamic Optimizations on Production Code

Michael Maass and Ilari Shafer  
{mmaass, ishafer}@cs.cmu.edu

March 20, 2012

## 1 Summary

We plan to implement an instrumentation strategy for Google’s V8 JavaScript engine. This will allow us to produce evidence to either support or reject the hypothesis that the optimizations utilized in industrial scripting engines produce speed gains for production code of the same magnitude as they do for current benchmarks.

Project Page: <http://www.cs.cmu.edu/~ishafer/compilers/>

## 2 Project Description

Throughout 15-745 we have focused on optimizations for static languages within the highly adaptable LLVM research platform. This framework has provided the ability to study optimizations and how they improve the performance of code. In this project, we plan to focus on evaluating optimization effectiveness for *dynamic* languages using a *production* compiler. This is an important research topic because the impact of recent optimizing compilers for dynamic languages on production code has not been studied in depth.

We have chosen to investigate JavaScript, which is a particularly important dynamic language as it has emerged as “the language of the Web” and more. Modern web applications are built in JavaScript or language layers built atop the language. JavaScript is emerging on more platforms: as the framework for writing desktop applications (Windows 8 Metro applications), mobile applications (PhoneGap and kin), server side code (node.js), databases (MongoDB), and even for systems programming (HP WebOS). We will instrument V8, a popular compiler that we can modify.

Our work will analyze production compilers and web applications. Many existing studies (see 5) have found that existing applications used to evaluate compiler effectiveness do not represent real usage. Using a production compiler affords us the opportunity to test optimizations that are used in practice as opposed to using an easier-to-instrument trivial runtime. We believe this work will help identify a few ineffective optimizations and contribute to a better understanding of optimizing dynamic languages. It will also primarily involve work within a compiler for a dynamic language, and grant us insight into its operation.

### 3 Plan of Attack and Schedule

Week Beginning	Plan for the Week
March 26	Learn where and how to instrument V8, produce a simple modification (such as emitting information from within an optimization)
April 2	Turn off all runtime optimizations, and build framework for selectively enabling single optimizations. Test one optimization. Identify metrics for evaluation.
April 9	Identify set of real-world applications and apply selective optimization to a few of them. Begin instrumenting metrics for evaluation (e.g. number of instructions executed) to determine benefit of optimizations.
April 16	Continue running set of real-world applications with instrumentation. Finish instrumenting evaluation metrics. If time, begin experimenting on another platform (see 6).
April 23	Transform raw data into condensed results, establish results. If time, finish experimenting with other platform.
April 30	Complete project writeup and poster

### 4 Milestones

For the project milestone on Thursday, April 19, we plan to have completed the framework for evaluating optimizations, and have experimented with one or two real-world applications.

### 5 Related Work

There is quite a bit of evidence in previous work that strongly suggests current benchmarks are not representative of production JavaScript [5, 7]. Several of the studies that examined the behavior of JavaScript at runtime also produced some evidence that existing optimizations actually make production code slower [4, 3]. To address the problem of poor benchmarks, work has been done to create better research compilers to aid development and experimentation [1] and towards automatically producing benchmarks using traces from executions of production scripts [6]. Unfortunately for optimizing compilers, it turns out that production JavaScript, in addition to being very dynamic, makes heavy use higher-order operations such as `document.write` and `eval` [2]. Currently there seems to be no published work that attempts to conclusively measure whether optimization suites or specific optimizations are effective or actually harmful for speeding up production scripts.

### 6 Resources Needed

We will need a copy of the V8 compiler framework. V8 is open-source, and has been integrated with many of the environments we are targeting. We will begin our work on standard (i.e. our own) machines. If we extend our work to other platforms, we will need one or more of these tools:

- node.js: a server-side JavaScript implementation, which is open-source (and, in fact, is by default a source install). It uses V8 as its engine.
- Android: we would experiment within an Android emulator, which is freely available.
- MongoDB: MongoDB is open-source; to modify the V8 portion its JavaScript engine can be switched to a modified version of V8.

### 7 Getting Started

Thus far, we have read seven recent papers from the area. We have also taken some initial steps towards the initial implementation aspects of the project:

- We have installed node.js from source and tested basic (hand-written) applications
- We have installed the Android emulator
- We have begun identifying relevant areas within the V8 source

Our next step is to begin modifying V8. Though we anticipate encountering obstacles as we delve further, we do not have any constraints that prevent us from starting immediately.

## References

- [1] M. Chevalier-Boisvert, E. Lavoie, M. Feeley, and B. Dufour. Bootstrapping a self-hosted research virtual machine for JavaScript: an experience report. In *DLS*, pages 61–72, New York, NY, USA, 2011. ACM.
- [2] H. Kikuchi, D. Yu, A. Chander, H. Inamura, and I. Serikov. JavaScript instrumentation in practice. In *APLAS*, pages 326–341, Bangalore, India, 2008. Springer-Verlag Berlin / Heidelberg.
- [3] J. Martinsen. A methodology for evaluating JavaScript execution behavior in interactive web applications. In *AICCSA*, pages 241–248, Sharm El-Sheikh, Egypt, 2011.
- [4] J. K. Martinsen, H. Grahm, and A. Isberg. Evaluating four aspects of JavaScript execution behavior in benchmarks and web applications. In *ICWE*, pages 399–402, Paphos, Cyprus, 2011. Springer.
- [5] P. Ratanaworabhan, B. Livshits, and B. G. Zorn. JSMeter: comparing the behavior of JavaScript benchmarks with real web applications. In *WebApps*, page 3, Berkeley, CA, USA, 2010. USENIX Association.
- [6] G. Richards, A. Gal, B. Eich, and J. Vitek. Automated construction of JavaScript benchmarks. In *OOP-SLA*, pages 677—694, Portland, OR, 2011. ACM.
- [7] G. Richards, S. Lebresne, B. Burg, and J. Vitek. An analysis of the dynamic behavior of JavaScript programs. In *Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation, PLDI '10*, pages 1–12, New York, NY, USA, 2010. ACM.