# EE382V: Project Proposal

Yilin Zhang
zylime@gmail.com

C. Vic Hu
vic@cvhu.org

***Abstract*—In this project, we want to focus on the trace profiling and possible optimization to the existing JIT JavaScript engines, i.e. V8. By applying the optimization techniques we have learned in class, we want to study how modern browsers dynamically compile JavaScript, and to see if we can come up with a feasible idea to make some enhancements.**

## I. Introduction

As the popularity of web-based applications and services increases, browser performance has become one of the major competitions in the industry. Since JavaScript is what makes modern web pages dynamic and interactive, how to optimize its compilation/interpretation is the key component to building fast and robust modern browsers. Apart from its essential role in client-side interactions, JavaScript also became one of the mainstream server-side solutions in recent years [3].

Although JavaScript is traditionally translated into bytecode by an interpreter, more and more JavaScript Engines in modern browsers are designed to compile directly into machine code. Our project will mainly focus on trace profiling [1] in V8, and make constructive adjustments according to the optimization techniques we have learned in class. We will use the SunSpider JavaScript Benchmark to measure and compare the existing infrastructure with our proposed optimizations, and make a sound analysis of the results. In our project, we made these contributions:

1) We compare the runtime of JavaScript and C++, followed by an analysis of which part slows down the JavaScript.
2) We analyzed three main techniques used in V8, which boost the runtime of JavaScript.
3) We perform the profiling on one JavaScript example.
4) Based on the profiling, we identify the hot path by function counter algorithm.

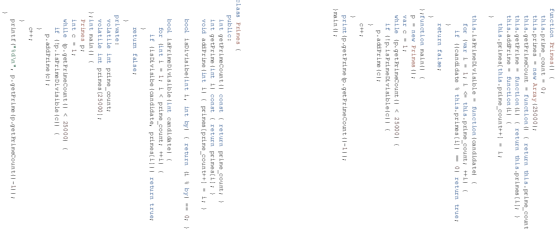
JavaScript is slow mainly due to JavaScript programs are untyped, and then compiled and run on the fly. Dynamic compilation is a great complement to static one. But completely replacing the optimized-to-death static compilation with JIT will lose the performance.

[hbt!] *Calculate the 25000th Prime Number*

**Require:**
**Ensure:** The 25000th Prime Number P
1: Prime list PL =
2: **for** P = 1 to infinity **do**
3: Flag = true
4: **for** index = 1 to PL.size() **do**
5: **if** P%PL[i] == 0 **then**
6: Flag = false
7: Continue
8: **end if**
9: **end for**
10: **if** Flag == true **then**
11: PL.push_back(P)
12: **if** PL.size() == 25000 **then**
13: **return** PL.back()
14: **end if**
15: **end if**

## II. Approach

JavaScript is slower compared with other programming languages, such as C++. Before applying the optimization to the compilation of JavaScript code, we use one example to analyze how slow JavaScript is compared with the same code in C++. The example we use here is to calculate the 25000th prime number [**?**]. The overall algorithm of of calculating the 25000th prime number is illustrated in Algorithm**??**. The corresponding C++ code is in the Fig.**??** while the JavaScript version is in the Fig.**??**.



[] []

Fig. 1. (a) is the C++ code of calculating the 25000th prime number (b) is the JavaScript code for 25000th prime number.

## References

[1] Jungwoo Ha and Mohammad R. Haghighat and Shengnan Cong, A concurrent trace-based just-in-time compiler for javascript, Tech Report, 2009
[2] John Garofalakis and Panagiotis Kappos and Dimitris Mourloukos, Web site optimization using page popularity, Internet Computing, 1999
[3] "Why Everyone Is Talking About Node." Jolie O'Dell. Mashable, Inc., Web. 10 Mar. 2011.