Improving JavaScript performance by deconstructing the type system

Paper review

Max Trunnikov 2024-01-30

Meta data

- **Authors:** Wonsun Ahn, Jiho Choi, Thomas Shull, María J. Garzarán, and Josep Torrellas
- **Conference:** PLDI '14: Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation
- **Year:** 2014
- **Citations:** 55
- **References:** 23
- **Downloads:** 822
- **Pages:** 12

What is this study about?

- Increased focus on JavaScript performance has resulted in vast performance improvements for many benchmarks.
- However, for actual code used in websites, the attained improvements often lag far behind those for popular benchmarks.
- The main reason behind this shortfall is how the compiler understands types.
- JavaScript has no concept of types, but the compiler assigns types to objects anyway for ease of code generation.
- Chrome V8 compiler defines types, and identify two design decisions that are the main reasons for the lack of improvement: (1) the inherited prototype object is part of the current object's type definition, and (2) method bindings are also part of the type definition.
- These requirements make types very unpredictable

Table of contents

Problem statement 👍

- Only recently has it come to light that the behaviour of JavaScript code in real websites is more dynamic than in the benchmarks
- V8 is unable to substantially improve the performance of JavaScript in JSBench
- Type dynamism stems from the way the compiler understands types
- V8 defines **three aspects** of objects to be part of the type description:
  - The object's structure
  - The object's inherited prototype
  - The object's method bindings.
- The expectation is that, although JavaScript allows for more dynamism, it will follow the behaviour of static languages in actual execution. This assumption, while true for popular benchmarks, often turns out to be inaccurate for real website code

Innovation 👍

Restructuring the type system

- Decoupling prototypes from types
  - Optimizing function creation
  - Optimizing Built-in object creation
- Decoupling method bindings from bindings
  - Complete decoupling
  - Partial decoupling

Contribution 👍

- It identifies type unpredictability as the main source of performance shortfall for website code. Further, it singles out frequent changes to prototypes and method bindings as the cause of the unpredictability, as well as scenarios that trigger them.
- It proposes three enhancements to the compiler that effectively decouple prototypes and method bindings from the type definition, and eliminate most type unpredictability.
- It implements these enhancements in V8 and evaluates them with JSBench. The results show that, on average, our enhancements reduce the execution time by 36%, and the dynamic instruction count by 49%. Moreover, the reduction in type dynamism leads to a reduction in bookkeeping information in the compiler, leading to a savings of 20% in heap memory allocation. Finally, the performance of popular JavaScript benchmarks is largely unchanged.

Readability 👍

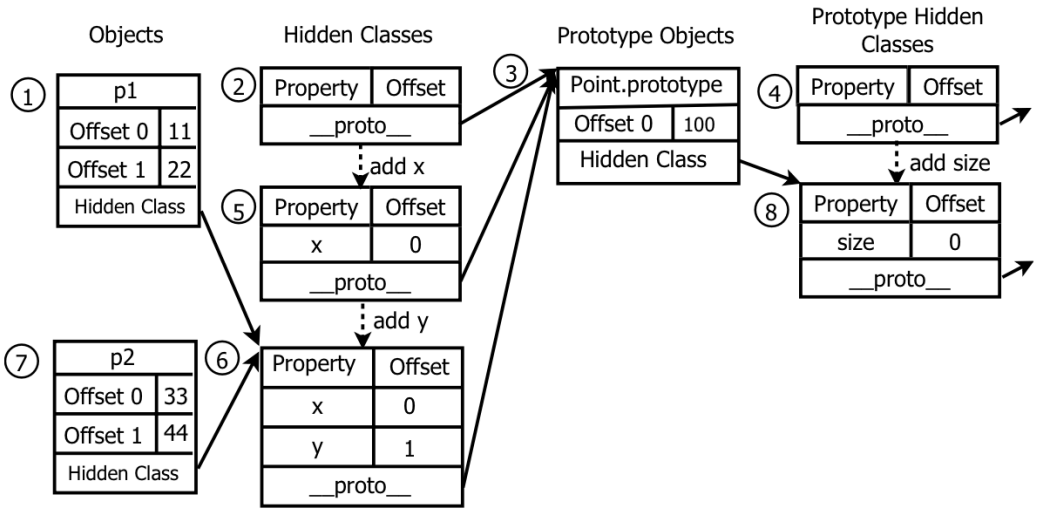
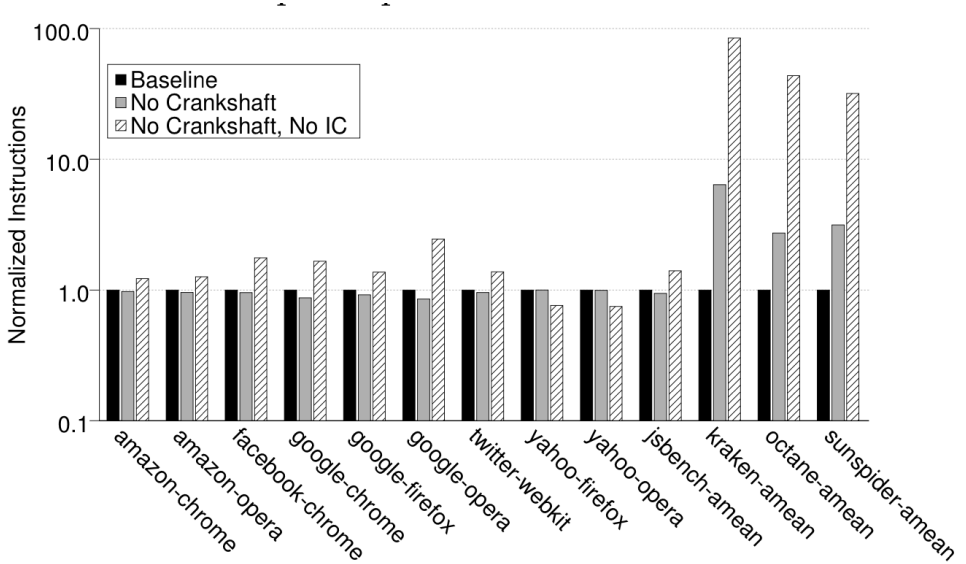


Figure 2: Example of hidden classes.

```
1 var initAmznJQ = function() {
2   window.amznJQ = new function() {
3     var me = this;
4     me.addLogical = function(...) {...};
5     me.declareAvailable = function(...) {...};
6     ...
7   }();
8   window.amznJQ.declareAvailable("JQuery");
9 }
```



An exemplary article