Poster: Automatically Fixing Real-World JavaScript Performance Bugs

Marija Selakovic and Michael Pradel
Department of Computer Science
TU Darmstadt, Germany
m.selakovic89@gmail.com, michael@binaervarianz.de

***Abstract*—Programs often suffer from poor performance that can be fixed by relatively simple changes. Currently, developers either manually identify and fix such performance problems, or they rely on compilers to optimize their code. Unfortunately, manually fixing performance bugs is non-trivial, and compilers are limited to a predefined set of optimizations. This paper presents an approach for automatically finding and fixing performance bugs in JavaScript programs. To focus our work on relevant problems, we study 37 real-world performance bug fixes from eleven popular JavaScript projects and identify several recurring fix patterns. Based on the results of the study, we present a static analysis that identifies occurrences of common fix patterns and a fix generation technique that proposes to transform a given program into a more efficient program. Applying the fix generation technique to three libraries with known performance bugs yields fixes that are equal or equivalent to those proposed by the developers, and that lead to speedups between 10% and 25%.**

I. Introduction

Many programs suffer from performance bugs, i.e., source code locations where a relatively simple change can significantly speedup the program [1]. Addressing such problems requires to identify performance bottlenecks, carefully study potential root causes, and to experiment with code transformations until one is found that leads to a significant performance improvement. Because this process is time-consuming and cumbersome, developers often rely on compiler optimizations, which however, are limited to a predefined set of transformations that can certainly be applied in a semantics-preserving way. This situation suggests that techniques for automatic detection and repair of performance bugs will be of considerable benefit.

This paper focuses on performance bugs in JavaScript, which has become one of the most popular programming languages. We address the problem in two steps: First, we perform an empirical study of real-world performance bug fixes to identify recurring fix patterns. Second, we present a static analysis and code transformation technique that proposes performance-improving code changes. The analysis searches for occurrences of performance bug patterns identified in the study and tries to apply the fix pattern. To focus on worthwhile fixes, the approach measures the performance improvement for given inputs and proposes a fix only if it leads to a statistically significant performance improvement. Our approach identifies performance problems missed by compilers because the approach does not guarantee to preserve the semantics, but leaves the final decision whether to apply a fix to the developer.

II. Study of Real-World Performance Bug Fixes

To focus our efforts on kinds of performance problems that occur in the wild, we search for performance issues reported in the bug repositories of popular JavaScript projects available on Github. We select projects and bug reports based on the following criteria:

Project type: We focus on projects that target the node.js platform, as its underlying JavaScript engine, V8, is highly tuned for performance.

Number of performance-related bugs: We focus on repositories with a high number of pull requests ($\geq$100) to increase the probability to find performance bugs.

Keyword-based search: We focus on bug reports that contain any of the following keywords: *'performance'*, *'slow'*, *'fast'* and *'responsive'*.

Resolved bugs: We consider only bugs with a fix that has been merged into the main project, assuming that the developers recognize the performance improvement of these changes.

In total, we identify and reproduce 37 performance bugs from eleven projects (underscore, Q, request, loadsh, mocha, mhalk, less, moment, cheerio, backbone, and ejs). For each of them, we reproduce the problem by creating a test based on the bug report and the documentation of the project, and we validate that the change indeed improves performance.

Through careful inspection of these bugs and their fixes, we identify 23 recurring fix patterns. Each fix pattern consists of a description of code before and after fixing the problem. 16 of these fix patterns are specific to JavaScript or its API, whereas the others are programming language-independent. We further assign the fix patterns into one or more of the following categories:

- Faster built-in function (13 bugs)
- More efficient data structure (5 bugs)
- Better reflection usage (5 bugs)
- Avoid functional programming style (2 bugs)
- Use new language feature (4 bugs)
- Avoid deprecated language feature (2 bugs)
- Manual application of traditional compiler optimization, e.g, function inlining (4 bugs)

Figure 1 shows three representative examples of JavaScript-specific performance bugs along with their fixes.

Performance bug	Bottleneck code	Optimized code	Pattern
Underscore issue 39	`str.split("'").join("\\'");`	`str.replace(/'/g, "\\'");`	Faster built-in function
Chalk issue 28	`return applyStyle._styles.reduce(function ( str, name) { var code = ansiStyles[name]; return code.open + str.replace(code.closeRe, code.open) + code.close; }, str) ;`	`var styles = applyStyle._styles; for (var i = 0; i < styles.length; i++) { var code = ansiStyles[styles[i]]; str = code.open + str.replace(code.closeRe, code.open) + code.close; } return str;`	Avoid functional programming style
Mocha issue 701	`if (toString.call(err) === "[object Error]") { .. }`	`if (err instanceof Error || toString.call(err) === "[object Error]") { .. }`	Use new language features

Fig. 1: Examples of JavaScript performance bugs and their fixes.

III. Automatic Repair with Fix Patterns

To help developers find and fix common performance bugs, we present an approach that automatically detects code that matches one of the fix patterns from Section II and that proposes a change to the developer. Given a program with some input, the approach consists of two parts: First, attempt to apply a fix pattern to the program. Second, validate whether the fix improves performance by measuring the execution time of the program before and after the change.

To apply fix patterns to a program, the approach traverses the AST of the program, and once it finds a subtree that matches the AST structure of the pattern specification, the approach attempts to change the AST as specified by the fix pattern. If and only if the approach can successfully transform the AST, it generates a transformed JavaScript program and passes it on to the second part of the approach.

After applying a fix pattern to the source code, the approach runs the old and the transformed version of the program with the given input, and it measures the execution time. We follow the methodology of Georges et al. [2], i.e., we repeatedly execute the program and apply statistical significance tests to obtain reliable measurements. If the difference in execution times is statistically significant, and if the improvement exceeds a user-provided minimum (we use 5%), then the transformation is reported as an optimization opportunity to the developer.

IV. Preliminary Results

We implement the automatic repair technique for JavaScript and apply it to several of the libraries from the study in Section II. Our prototype implementation builds upon existing parser and code generation libraries.[1] The implementation successfully identifies and fixes the three JavaScript-specific performance bugs from Figure 1. The generated fixes are equal or semantically equivalent to the fixes that the developers have found. Applying the automatically generated fixes gives speedups of 25%, 12% and 10 % for first, second, and third example in Figure 1, respectively.

[1] http://esprima.org/ and http://github.com/estools/escodegen

V. Related Work

Previous studies of real-world performance bugs show that these bugs are common and non-trivial to fix [1], [3], [4]. Our work contributes by studying JavaScript-specific performance bugs and by exploiting the results of this study for automatic fix generation. Several recent approaches generate fixes for correctness problems [5], [6]. These approaches search for fixes that change the semantics of a program, whereas our work aims at preserving the semantics while improving performance.

VI. Conclusion

This paper presents ongoing work on studying and automatically fixing real-world JavaScript performance bugs. Our preliminary results show that the approach successfully detects three types of recurring performance problems and proposes fixes that improve performance and that are acceptable by the developers. Our work is guided by bug fix patterns revealed in an empirical study, and therefore, focuses on relevant kinds of performance problems. Encouraged by the preliminary results, we are currently working towards extending the study to gather more performance bug fix patterns and towards generalizing the automatic fix generation technique based on a generic description of fix patterns.

Acknowledgments

This research is supported by the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE and by the German Research Foundation (DFG) within the Emmy Noether Project "ConcSys" (PR 1537/1-1).

References

[1] G. Jin, L. Song, X. Shi, J. Scherpelz, and S. Lu, "Understanding and detecting real-world performance bugs," in *PLDI*, 2012, pp. 77–88.
[2] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous Java performance evaluation," in *OOPSLA*, 2007, pp. 57–76.
[3] S. Zaman, B. Adams, and A. E. Hassan, "A qualitative study on performance bugs," in *MSR*, 2012, pp. 199–208.
[4] Y. Liu, C. Xu, and S. Cheung, "Characterizing and detecting performance bugs for smartphone applications," in *ICSE*, 2014, pp. 1013–1024.
[5] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest, "Automatically finding patches using genetic programming," in *ICSE*, 2009, pp. 363–374.
[6] D. Kim, J. Nam, J. Song, and S. Kim, "Automatic patch generation learned from human-written patches." in *ICSE*, 2013, pp. 802–811.