Improving Performance and Quality of Database-Backed Software

Junwen Yang
University of Chicago, USA
junwen@uchicago.edu

Abstract

Modern web applications have stringent latency requirements while processing an ever-increasing amount of user data. To address these challenges and improve programmer productivity, Object Relational Mapping (ORM) frameworks have been developed to allow developers writing database processing code in an object-oriented manner. Despite such frameworks, prior work found that developers still struggle in developing ORM-based web applications. This paper presents a series of study and developed tools for optimizing web applications developed using the Ruby on Rails ORM. Using automated static analysis, we detect ORM related inefficiency problems and suggests fixes to developers. Our evaluation on 12 real-world applications shows that more than 1000 performance issues can be detected and fixed.

CCS Concepts • **Software and its engineering → Software performance**.

Keywords performance anti-patterns, database-backed applications

ACM Reference Format:
Junwen Yang. 2019. Improving Performance and Quality of Database-Backed Software. In *Proceedings of the 2019 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion '19), October 20–25, 2019, Athens, Greece.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3359061.3361076

1 Approach

Unlike prior approaches that tackle performance problems in web servers and database servers separately, my research treats them in tandem:

ACM ISBN 978-1-4503-6992-3/19/10…$15.00
https://doi.org/10.1145/3359061.3361076

1. I led a comprehensive study of hundreds of performance and scalability problems in real-world popular database-backed web applications, and created the first taxonomy of these problems by holistically considering application design, database design, and web-interface design.

2. I led the design and implementation of a set of tools that automatically detect and fix performance and scalability problems in database-backed web applications, through cross-stack and DB-aware analysis of web applications.

3. I participated in designing more efficient database query execution leveraging semantic information automatically extracted from web applications.

My past research has led to three first-author papers published on the 41th International Conference on Software Engineering (ICSE) in 2019, which won the ACM SIGSOFT Distinguished Paper Award [10], 40th ICSE in 2018 [9], which raised much attention in open-source community and was featured on Morning Paper blog [3], RubyWeekly [4] and Hacker News [2], and 26th Foundations of Software Engineering (FSE) in 2018 [8], which automatically identified more than 1000 performance/scalability problems in latest versions of popular open-source web applications, and a co-authored paper on the 26th Conference on Information and Knowledge Management (CIKM) 2017 [7].

Comprehensive Bug Study. Realizing that performance and scalability problems in database-backed web applications go far beyond query efficiency, I then led a project (published in ICSE'18 [9]) that conducted a cross-stack and comprehensive study about performance and scalability issues in 12 most popular open-source applications of 6 popular categories written in Ruby-on-Rails (Rails), the most popular ORM framework [9].

(1) Through thorough profiling using our carefully synthesized workload, which follows real-world user-data statistics, around one hundred new performance and scalability problems are identified by us in the latest versions of these popular applications, showing the severity and prevalance of these problems. Furthermore, these problems can be largely mitigated by simple patches designed by me — mostly less than 5 lines of code changes providing huge server-side performance speedups (2X median, and up to 39X)

(2) Studying more than 100 issues reported by real-world users, together with problems discovered by our profiling above, I summarized 9 performance anti-patterns belonging to three major categories:

- Inefficient misuse of database-related APIs;
- Performance-unaware design of web pages;
- Unsuitable database designs.

This is the first comprehensive study on real-world performance issues in database-backed web applications. This study has already raised attentions in the community [2–4], and provides guidance and benchmarks to future research along this direction.

Following this empirical study, I then built several tools to tackle the three major aspects of performance problems identified above:

(i) PowerStation (published in FSE'18 [8]), a static analysis and IDE plugin that automatically detects and fixes inefficient ***misuse of database-APIs*** in database-backed web applications;

(ii) Panorama (published in ICSE'19 [10]), a view-centric and database/server-aware development environment that helps developers trade-off functionality and performance in ***web-page design***;

(iii) an empirical study (published in CIKM'17 [7]) that explores how to customize ***database design*** to better serve a web application.

Powerstation: detecting & fixing database-API misuses PowerStation provides a set of database-aware optimization to Ruby-on-Rails applications. Specifically, PowerStation uses its DB-aware static analysis to identify loop invariant queries (automatically fixed by loop query motion), dead store queries (automatically fixed by removing the queries), unused data retrieval (automatically fixed by revising queries), common query subexpression, ORM API misuses (automatically fixed by API refactoring), etc. Previous work can only detect half of the anti-patterns covered by PowerStation [6, 7, 9], and we are unaware of any previous tools that can automatically fix any of them.

PowerStation has been integrated into a very popular Rails IDE, RubyMine [5], so that Rails developers can easily benefit from PowerStation to improve the efficiency of their applications. It can be freely downloaded from the IntelliJ Plugin Repository [1]. Many performance-related issues in real-world applications have been identified and fixed.

Panorama: synthesizing efficient web-page designs Panorama helps developers understand the data-processing as well as server-communication cost of generating every web-page/user-interface components in order to explore and pick the interface design with the best trade-off between performance and functionality. In this environment, we use cross-stack database-aware program analysis and novel IDE design to provide developers with intuitive information about the cost and the performance-enhancing opportunities behind every interface component, as well as suggesting various cross-stack code refactorings that enable developers to easily explore a wide spectrum of performance and functionality trade-offs.

Tackling database design problems. Finally, I participated in a project (published in CIKM'17 [7]) that applies static program analysis and profiling to identify database inefficiencies in real-world web applications. This project shows inefficiency patterns in database query processing, such as poor physical database design, the lack of caching that results in redundant query processing, etc. We also proposed ways to optimize database processing leveraging program semantic information automatically identified through static program analysis.

2 Evaluation Methology

2.1 Application Selection

We evaluate Panorama and PowerStation using a suite of 12 open-source Ruby on Rails applications, including top 2 most popular Ruby applications from 6 major categories of web applications on GitHub: Discourse (Ds) and Lobster (Lo) are forums; Gitlab (Gi) and Redmine (Re) are collaboration applications; Spree (Sp) and Ror_ecommerce (Ro) are E-commerce applications; Fulcrum (Fu) and Tracks (Tr) are Task-management applications; Diaspora (Da) and Onebody (On) are social network applications; OpenStreetmap (OS) and FallingFruit (FF) are map applications. They have all been actively developed for years, with hundreds to tens of hundreds of code commits.

2.2 PowerStation

We evaluated PowerStation using the latest versions of 12 open-source Rails applications. As shown in Table **??**, PowerStation can automatically identify 1221 inefficiency issues and generate patches for 730 of them (i.e., all but the common sub-expression pattern). We randomly sampled and examined half of the reported issues and the suggested fixes, and found no false positives. Due to the limited resource and time, we reported 433 issues with 57 of them already confirmed by developers (none has been denied). PowerStation static analysis is fast, taking only 12–625 seconds to analyze the entire application that ranges from 4k to 145k lines of code in our experiments on a Chameleon instance with 128GB RAM and 2 CPUs. Developers can also choose to analyze one action at a time, which usually takes less than 10 seconds in our experiments.

2.3 Panorama

To evaluate Panorama, we focuses on four research questions: **RQ1**: Can Panorama identify view-aware optimization opportunities from latest versions of popular web applications? **RQ2**: How much performance benefits can view-aware optimization provide? **RQ3**: Is the performance-functionality trade-off space exposed by Panorama worthwhile for developers to explore? **RQ4**: Does Panorama estimator estimate the per-tag data-processing cost accurately?

For **RQ1**, Panorama has been applied on 12 applications and can indeed identify many view-aware optimization opportunities . Specifically, as shown in Table **??** Panorama static analysis identifies 149 performance-enhancing opportunities from the current versions of our benchmark applications. Every type of optimization opportunities is identified from at least 8 applications.

For **RQ2**, to quantitatively measure the performance benefits of these alternative view designs, we randomly sampled 15 optimization opportunities identified, with 6, 2, 4, and 3 cases from Pagination, Asynchronous (loading), Approximation, and Content Removal respectively. For each application, before and after optimization, we run a Chrome-based crawler that visits links randomly for 2 hours and measure the average end-to-end-latency and server-cost of every action. We then compute speedup accordingly.

For **RQ3**, we conducted a user study by recruiting 100 participants from Amazon Mechanical Turk. Our benchmark suite includes 12 web pages from 5 web applications. For each of these 12 baseline pages, Panorama automatically generates a new page with exactly one HTML tag changed. We refer to the original page as *Base* and the one optimized by Panorama as *New*.

Each participant is assigned 8 tasks. In each task, they are asked to click two links one by one, and then answer questions about (1) which page they think is faster ("Performance"); (2) which page they think delivers more or better organized content ("Functionality"); and (3) which page do they like more with everything considered ("Overall"). These two links are the *Base* and *New* versions of one benchmark, with random ordering between them.

For **RQ4**, we used a webpage as a case study, and calculate and compare the performance ranking through both dynamic workload (200, 2000, and 20000 records database) and static analysis to see whether the performance estimator is accurate.

2.4 Threats to Validity

Threats to the validity of our study could come from multiple sources. Applications beyond these 12 applications may not share the same problems as these 12 applications. The profiling workload synthesized by us may not accurately represent the real-world workload. The machine and network settings of our profiling may be different from real users' setting. Our study of each application's bug-tracking system does not consider bug reports that are not fixed or not clearly explained. Despite these aspects, we have made our best effort in conducting a comprehensive and unbiased study, and we believe our results are general enough to guide future research on improving performance of ORM applications.

3 Future Work

My future research will continue to look at such cross-stack and cross-server problems in big data management and processing systems.

First, still along the performance direction, I plan to look beyond web applications and investigate other types of big-data systems, including those with non-SQL back-ends and those with mobile front-ends. New application set needs to be explored.

Second, going beyond performance concerns, I plan to also look at correctness issues that are related to data maintenance in these database-backed systems. Specifically, I plan to look at what type of assumptions developers put on persistent data, how these assumptions are expressed and enforced in the system, and how these assumptions evolve and get maintained, effectively or poorly, when software evolves. I plan to similarly conduct empirical study first and then design tools to help improve data-maintenance correctness accordingly. To evaluate, we will continue to use the same set of applications, dig into their issue tracking system, and develop tools to see whether we can find problems previously unknown.

In summary, I expect my thesis research to make the development and maintenance of correct and efficient big-data software easy!

References

[1] 2018. *Download PowerStation.* https://bit.ly/2NYFRs3.
[2] 2018. *Hacker News.* https://news.ycombinator.com/item?id=17414383.
[3] 2018. The morning paper. https://bit.ly/2Ixpkl4.
[4] 2018. RubyWeekly. https://rubyweekly.com/issues/406.
[5] 2019. *RubyMine.* https://www.jetbrains.com/ruby/.
[6] Tse-Hsun Chen, Weiyi Shang, Zhen Ming Jiang, Ahmed E Hassan, Mohamed Nasser, and Parminder Flora. 2016. Finding and evaluating the performance impact of redundant data access for applications that are developed using object-relational mapping frameworks. *Transactions on Software Engineering* (2016).
[7] Cong Yan, Junwen Yang, Alvin Cheung, and Shan Lu. 2017. Understanding Database Performance Inefficiencies in Real-world Web Applications. In *26th Conference on Information and Knowledge Management (CIKM).*
[8] Junwen Yang, Pranav Subramaniam, Shan Lu, Cong Yan, and Alvin Cheung. 2018. PowerStation: Automatically detecting and fixing inefficiencies of database-backed web applications in IDE. In *26th Foundations of Software Engineering (FSE'18 Demostration Track).*
[9] Junwen Yang, Cong Yan, Pranav Subramaniam, Shan Lu, and Alvin Cheung. 2018. How not to structure your database-backed web applications: a study of performance bugs in the wild. In *IEEE/ACM 40th International Conference on Software Engineering (ICSE).* IEEE, 800–810.
[10] Junwen Yang, Cong Yan, Chengcheng Wan, Shan Lu, and Alvin Cheung. 2019. View-Centric Performance Optimization for Database-Backed Web Applications. In *IEEE/ACM 41th International Conference on Software Engineering (ICSE).* IEEE.