BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

PERFORMANCE ANALYSIS OF WEB APPLICATIONS

ANALÝZA VÝKONU WEBOVÝCH APLIKACÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **TOMÁŠ VALENT**
AUTOR PRÁCE

SUPERVISOR **Ing. JIŘÍ PAVELA,**
VEDOUCÍ PRÁCE

BRNO 2024

Abstract

Abstrakt

Keywords
Typescript, profiling, Perun, continuous monitoring

Klíčová slova
Typescript, profilovanie, Perun, nepretržié monitorovanie

Reference
VALENT, Tomáš. *Performance Analysis of Web Applications*. Brno, 2024. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jiří Pavela,

Performance Analysis of Web Applications

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Jiří Pavela. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Tomáš Valent
October 30, 2023

Acknowledgements

I would like to thank the supervisor of this thesis, Ing. Jiří Pavela, and the technical assistant, Ing. Tomáš Fiedor Ph.D., for their supervision and patience.

Contents

List of Figures

Chapter 1

Introduction

Every programmer wants to have their application perfect. We usually encounter many different problems and bugs that are in our way to do so. One of the most common issues is *performance* of the program.

> Performance is part of the non-functional requirements of software. It can be defined as how efficiently a software can accomplish its tasks. A system is never more performative than its slowest part. And that part is what we call a *bottleneck* or *contention*. If you want to improve the performance of your system, you have to improve the performance of the slowest part. As all your processing is queueing in there, the rest of your system hasn't reached its peak yet. [8]

There are a lot of tools that can help us identify performance issues – profilers. Profiling is one of the software's forms of *dynamic analysis*. I will explain the types of program analysis in more detail in Chapter 2. The main goal of profiling is to identify performance issues such as throughput (either operations or data volume per second), IOPS (input/output operations per second), utilization (how busy a resource is, as a percentage), latency (operation time, as an average or percentile), CPU load and visualize them. Common ways to visualize the results of profiling are graphs and heat maps.

The main objective of this thesis is to develop a new module that will extend the system for versioning performance profiles *Perun*, which is developed by the *VeriFIT* group at the Faculty of Information Technology BUT, by a profiler that can profile web applications programmed in *Typescript*. TypeScript is a free and open-source high-level programming language developed by Microsoft. TypeScript is a syntactic superset of JavaScript which adds static typing. This means that TypeScript adds syntax on top of JavaScript, allowing developers to add types [10]. Unlike JavaScript, TypeScript supports object-oriented programming concepts in the vein of classes, interfaces, and inheritance. [9]

The following chapters will deal with these topics: Introduction of program analysis and profiling in Chapter 2. Introduction of version control system Perun in Chapter 3. Comparison of existing open-source Typescript profilers in Chapter 4. Design of the module in Chapter 5. All information about the implementation is in Chapter 6.

Chapter 2

Performance Analysis

This chapter will closely explain profiling as a whole, the types of software analysis, and some profiling concepts.

Performance analysis is a critical aspect of software development and system optimization. Whether you are a developer, a system administrator, or an IT manager, understanding and improving the performance of your applications is necessary. In a time when user expectations are higher than ever, and a fraction of a second can make a significant difference, it is essential to have the knowledge and tools to identify and remove performance bottlenecks.

2.1 Web application profiling

Profiling builds a picture of a target that can be studied and understood. In the field of computing performance, profiling is typically performed by *sampling* the state of the system at timed intervals and then studying the set of samples. The use of sampling provides a coarse view of the target's activity. How coarse depends on the rate of sampling. As an example of profiling, CPU usage can be understood in reasonable detail by sampling the CPU instruction pointer or stack trace at frequent intervals to gather statistics on the code paths that are consuming CPU resources. [3]

2.1.1 Static analysis

Static Testing is a type of software testing method which is performed to check the defects in software without actually executing the code of the software application. Static analysis is performed in the early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily. The errors that can't not be found using dynamic analysis, can be easily found by static analysis. [2]

Static analysis is for example code review. In code review, developers look at the code of each other with the goal of finding possible mistakes while they do not run the code to test it.

2.1.2 Dynamic analysis

Dynamic analysis is a type of software testing which is performed to analyze the dynamic behavior of the code. It includes the testing of the software for the input values and output values that are analyzed. [2]

Dynamic analysis of software can be for example profiling or unit testing. The point is to test the code with various inputs during the runtime.

2.1.3 Continuous monitoring

Continuous profiling is a real-time monitoring technique used to analyze production applications to identify and resolve performance issues that impact application user experience. It involves gathering data on execution-related factors like CPU usage, disk I/O consumption, and duration of function calls to point out impediments and areas for improvement.

With continuous profiling, developers get more information about executed functions. Engineers, on the other hand, are able to collect CPU utilizations and note measuring usage for performance improvements. [4]

Continuous monitoring can be used to profile web pages, in cybersecurity, or in hospitals it can be „vital signs monitor“

2.2 Sampling

Sampling collects statistical data about the work that is performed by an application during profiling and it's a good starting place to find areas to speed up your application. At specified intervals, the Sampling method collects information about the functions that are executing in your application. Data collection is done by collecting information on the application at a regular interval or sampling frequency such as every millisecond. The collected data is analyzed to create a model of where the time was spent in the application. If you need accurate measurements of call times or are looking for performance issues in an application for the first time, then you may want to use sampling.

Sampling has less accuracy in the number of calls but is low cost to the profiler and has little effect on the execution of the application being profiled. [5]

2.3 Instrumentation

Instrumentation profiling collects detailed information about the work that is performed by an application during a profiling run. Data collection is done by tools that either inject code into a binary file that captures timing information or by using callback hooks to collect and emit exact timing and call count information while an application runs. The instrumentation method has a high overhead when compared to sampling-based approaches. [5]

2.4 Sampling vs. Instrumentation

The value of sampling is that it has less overhead and for this reason is more likely to be statistically representative of the application running in production. The value of instrumentation profiling is that you can get exact call counts on how many times your functions were called. This gives you much more detailed information than normal sampling, which can distort the time taken in some scenarios. For example, functions that don't do much, but are called frequently, will show up more than they would in a real-world scenario.

With instrumentation, every function call selected in your application is annotated and instrumented so that when it gets invoked it's added to the trace along with information

about the caller. With sampling, the current call stack executing is polled from the CPU at an interval and then each frame is added to the trace. [5]

2.5 Tracing

Tracing provides better information on how often a method was executed. If you need accurate measures of call numbers, use tracing. Tracing can have a larger impact on the performance of your code during collection, but sampling has only a small overhead. Additionally, tracing can be slower to analyze because it takes longer to view the data after collection. [5]

2.6 Profiling metrics

In profiling, we can measure different *metrics*. Metrics show us how effectively the is program written and we can use them to visualize graphs, heat maps, and so on. Those metrics and visualizations can be used to analyze the performance of the program. Now I am going to list the most common performance metrics.

- **Total time** – The total time of a program is the easiest metric to measure while profiling. Profiling time involves measuring the amount of time for a specific piece of code or a program to execute. This can help identify bottlenecks and performance issues in the code.
  - **Wall time** – is time of a specific function. It is a difference between the start and the end of the function.
- **Functions calls** – The number of calls for each function in the program can be helpful in finding non-optimal parts of the code. simple solution can be the cached result of the function so that the developer does not have to call it more than it is needed. For example, if the function is loading data from a database it can save many function calls but also total performance time.
- **CPU** – CPU profiling focuses on monitoring the usage of the central processing unit (CPU). It helps identify how much CPU time is spent on various functions or code sections, aiding in optimizing code for efficiency.
- **Memory** – Memory profiling assesses the usage of system memory (RAM) by a program. It helps find memory leaks, inefficient memory allocation, and excessive memory usage, which can lead to performance problems or crashes.
- **Instructions** – Instruction profiling involves counting the number of machine instructions executed by a program. This can be useful for identifying code segments that may be expensive.

Chapter 3

Perun

In this chapter, I am going to describe what is Perun. **Per**formance **Un**der Control is an open source light-weight Performance Version System, which works as a wrapper over existing Version Control Systems and in parallel manages performance profiles corresponding to different versions of projects [7]. Moreover, it offers a tool suite allowing one to automate the performance regression test runs, post-process existing profiles or interpret the results [6, Chapter 1.1]

3.1 Overview

Perun has the following advantages over databases and sole Version Control Systems:

1. **Context** – each performance profile is assigned to a concrete minor version adding the missing context to your profiles—what was changed in the code base, when it was changed, who made the changes, etc. The profiles themselves contain collected data and additional information about the performance regression run or application configurations.

2. **Automation** – Perun allows one to easily automate the process of profile collection, eventually reducing the whole process to a single command and can be hence hooked, e.g. when one commits new changes, in the supported version control system to make sure one never misses to generate new profiles for each new minor or major version of the project.

3. **Genericity** – supported format of the performance profiles is based on JSON notation and has just minor requirements and restrictions. Perun tool suite contains a basic set of generic (and several specific) visualizations, postprocessing, and collection modules which can be used as building blocks for automating jobs and interpreting results. Perun itself poses only a minor requirements for creating and registering new modules

4. **Easy to use** – the workflow, interface, and storage of Perun is heavily inspired by the git systems aiming at natural use (at least for the majority of potential users). The current version has a Command Line Interface consisting of commands similar to git (e.g., add, status, and log). The Interactive Graphical User Interface is currently in development.

[6, Chapter 1.1]

Figure 3.1 shows a comparison of the git and Perun systems.

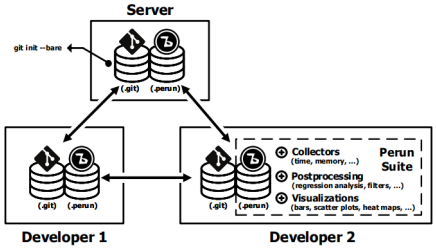

Figure 3.1: Comparison of the git and Perun systems [6, Chapter 1.1]

3.2 Architecture

PERUN consists of a tool suite and a wrapper over Version Control Systems (VCS), such as git, that keep track of performance profiles for different project versions. Figure 3.2 shows the workflow of the Tracer profiler is divided into four steps. [1]

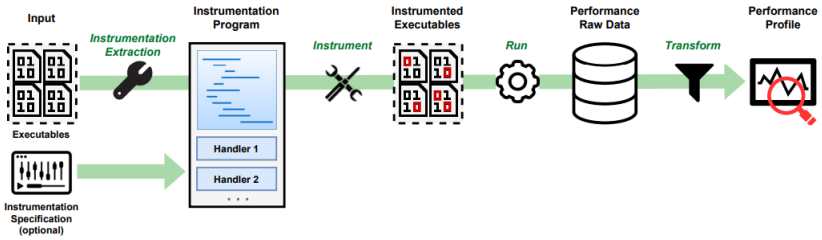

Figure 3.2: Comparison of the git and Perun systems [1]

- The input executables are first used to identify potential instrumentation points (unless the user specifies them manually) and an instrumentation program and instrumentation handlers are generated. The instrumentation program instructs the instrumentation tool on how to perform the instrumentation, and the handlers say what should be done at various instrumentation points.
- The input executables are instrumented using the generated instrumentation program and handlers
- The obtained executables are started (in a way specified by the user) and generate raw performance data
- Finally, the raw performance data are transformed into a performance profile.

[1]

The internal architecture of Perun can be divided into several units—logic (commands, jobs, runners, store), data (vcs and profile), and the tool suite (collectors, postprocessors, and visualizers). Data includes the core of the Perun—the profile manipulation and supported wrappers (currently git and simple custom vcs) over the existing version control systems. The logic is in charge of automation, and higher-logic manipulations and takes care of the actual generation of the profiles. Moreover, the whole Perun suite contains a set of collectors for the generation of profiles, a set of postprocessors for transformation, and

various visualization techniques and wrappers for graphical and command-line interfaces. [6, Chapter 1.4]

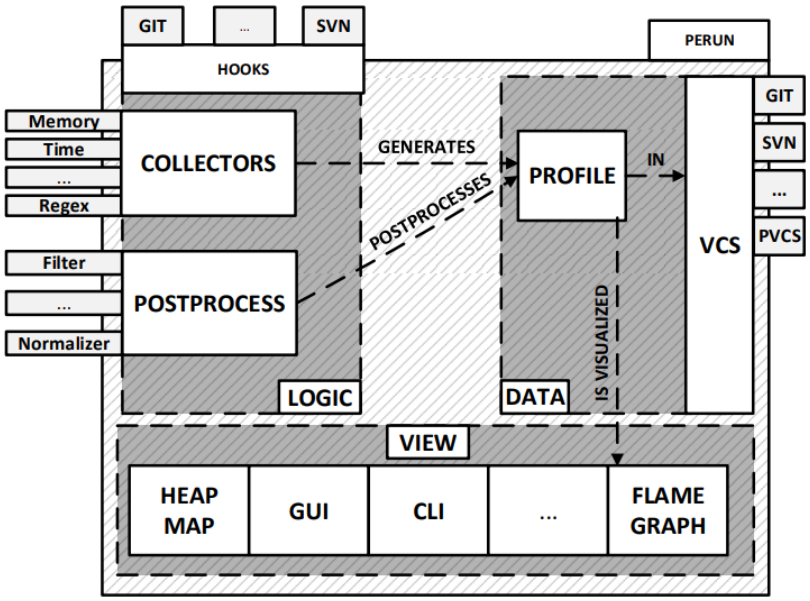

Figure 3.3: The scheme above shows the basic decomposition of Perun suite into sole units [6, Chapter 1.4]

3.3 Detecting Performance Changes

Every change of the project, and every new minor version, can cause a performance degradation of the project. And manual evaluation of whether the degradation has happened is hard. Perun allows one to automatically check the performance degradation between various minor versions within the history and protect the project against potential degradation introduced by new minor versions. Potential changes in performance are then reported for pairs of profiles, together with more precise information, such as the location, the rate, or the confidence of the detected change. The detection of a performance change is always checked between two profiles with the same configuration (i.e. collected by the same collectors, postprocessed using the same postprocessors, and collected for the same combination of command, arguments, and workload). These profiles correspond to some minor version (so-called target) and its parents (so-called baseline). But baseline profiles do not have to be necessarily the direct predecessor (i.e. the old head) of the target minor version and can be found deeper in the version hierarchy. [6, Chapter 8]

Chapter 4

Existing TypeScript profilers

Chapter 5

Architecture for Profiling

Chapter 6

About implementation

Chapter 7

Conclusion

Bibliography

[1] Fiedor, T., Pavela, J., Rogalewicz, A. and Vojnar, T. *PERUN: Performance Version System* [online]. June 2022 [cit. 2023-10-30]. Available at: https://arxiv.org/pdf/2207.12900.pdf.

[2] geeksforgeeks.org. *Difference between Static and Dynamic Testing* [online]. May 2019 [cit. 2023-10-15]. Available at: https://www.geeksforgeeks.org/difference-between-static-and-dynamic-testing.

[3] Gregg, B. System Performance: Enterprise and the Cloud. In:. 2nd ed. Pearson, 2020, chap. 2, p. 35. ISBN 9780136820154.

[4] influxdata.com. *Continuous Profiling* [online]. April 2023 [cit. 2023-10-18]. Available at: https://www.influxdata.com/glossary/continuous-profiling.

[5] microsoft.com. *Understand profiler performance collection methods* [online], 15. march 2023 [cit. 2023-10-23]. Available at: https://learn.microsoft.com/en-us/visualstudio/profiling/understanding-performance-collection-methods-perf-profiler.

[6] Pavela, J. and Fiedor, T. *Perun Documentation* [online]. June 2022 [cit. 2023-10-30]. Available at: https://raw.githubusercontent.com/tfiedor/perun/master/docs/pdf/perun.pdf.

[7] Pavela, J., Fiedor, T., Vojnar, T., Podola, R. and Míchal, O. *Perun: Lightweight Performance Version System* [online], 28. september 2023 [cit. 2023-10-24]. Available at: https://github.com/Perfexionists/perun/blob/devel/README.md.

[8] Sipriano, R. *Https://www.linkedin.com/pulse/what-actually-software-performance-rodrigo-sipriano* [online]. December 2021 [cit. 2023-10-14]. Available at: https://www.linkedin.com/pulse/what-actually-software-performance-rodrigo-sipriano.

[9] thenewstack.io. *What Is TypeScript?* [online]. July 2022 [cit. 2023-10-14]. Available at: https://thenewstack.io/what-is-typescript.

[10] w3schools.com. *TypeScript Introduction* [online]. March 2022 [cit. 2023-10-14]. Available at: https://www.w3schools.com/typescript/typescript_intro.php.