BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

UNIFIED REPORTING FOR PERFORMANCE TESTING
UNIFIED REPORTING FOR PERFORMANCE TESTING

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. MARTINA KŮROVÁ**
AUTOR PRÁCE

SUPERVISOR — **Mgr. Bc. HANA PLUHÁČKOVÁ**
VEDOUCÍ PRÁCE

BRNO 2017

Brno University of Technology - Faculty of Information Technology

Department of Intelligent Systems Academic year 2016/2017

Master's Thesis Specification

For:	**Kůrová Martina, Bc.**
Branch of study:	Information Systems
Title:	**Unified Reporting for Performance Testing**
Category:	Software analysis and testing

Instructions for project work:

1. Analyze typical performance issues in complex systems, the best set of attributes to observe and the mapping between measured values and the performance issues.
2. Design a unified reporting component (i.e. destination) for the open-source performance testing tool PerfCake that would compile all the measured values into a single portable document.
3. Implement the reporting component where the document will identify potential performance issues. Try to minimize time to analyze the system and identify what is the problem.
4. Test the developed component by unit and integration tests.
5. Using a simulated broken service, it will be verified that the reporting component can successfully discover the performance issues.

Basic references:

- Java Performance: The Definitive Guide by Scott Oaks, Link: http://a.co/jahOI7N
- Effective Java (Java Series) by Joshua Bloch, Link: http://a.co/btIyXYq
- JavaTM Puzzlers: Traps, Pitfalls, and Corner Cases by Joshua Bloch et al., Link: http://a.co /aCdNZLU
- Clean Code: A Handbook of Agile Software Craftsmanship by Robert C. Martin, Link: http://a.co/dyiYPxi
- Pro Java EE 5 Performance Management and Optimization by Steven Haines, Link: https://amzn.com/1590596102

Requirements for the semestral defense:
Items 1 and 2.

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

Supervisor:	**Pluháčková Hana, Mgr. Bc.**, DITS FIT BUT
Beginning of work:	November 1, 2016
Date of delivery:	May 24, 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

[illegible]

Petr Hanáček
Associate Professor and Head of Department

Abstract

Modern advances in software technologies for today's application development have allowed for developers to concentrate less on issues, such as performance and resource management, and instead spend more time on developing the application functionality such that the time to market is reduced. Consequently, performance analysis and optimization become more difficult and create a need for advanced performance tools that should provide a clear report of the application in terms of its performance and allow a fast interpretation of these results. This work investigates typical performance problems of today's applications and offers approaches on how to automatically detect them. Using statistical methods like regression and correlation analysis, investigation of measured values is performed in order to detect performance deviations that possibly occurred in an application under test. The proposed approach has been implemented as a new Reporter component into an open source performance testing tool PerfCake, developed by QE engineers from Red Hat Czech s.r.o. The developed component is capable of detecting and reporting possible issues and their probability. A unified report from all pre-specified measurements is created in such a way that all detected performance issues are immediately visible. The aim is to improve an end-user experience and usability when reading the report from performance testing.

Abstrakt

Moderní pokrok v oblasti technologií pro vývoj dnešních softwarových aplikací umožnil vývojářům více se soustředit na vývoj funkčnosti aplikace na úkor sledování jejího výkonu a správy zdrojů. V důsledku toho se zvýšily požadavky na nástroje pro výkonnostní testování, které by měly poskytovat vývojářům jasný a srozumitelný přehled o stavu systému z hlediska jeho výkonu a umožnit rychlou interpretaci naměřených výsledků. Tato práce zkoumá typické výkonnostní problémy dnešních aplikací a navrhuje přístupy, pomocí kterých je možné tyto anomálie automaticky rozpoznat. Pomocí statistických metod, jako je regresní a korelační analýza, je provedena analýza dat naměřených během výkonnostního testování s cílem rozpoznat ve výsledcích odchylky od normálního chování a z nich identifikovat výkonnostní problémy. Výsledkem je report o celkovém stavu systému z hlediska jeho výkonu. Implementací regresní analýzy je možné detekovat výkonnostní problémy jako je například zhoršující se reakční čas odpovědi, nízká propustnost systému či odhalit únik paměti. Navrhovaný přístup byl implementován v podobě nové komponenty v open-source nástroji pro výkonnostní testování PerfCake. Vyvinutá komponenta je schopna detekovat a reportovat potenciální výkonnostní problémy a jejich pravděpodobnost.

Keywords

anomalies detection, regression analysis, performance testing, perfcake, java

Klíčová slova

detekce anomálií, regresní analýza, výkonnostní testování, perfcake, java

Reference

KŮROVÁ, Martina. *Unified Reporting for Performance Testing.* Brno, 2017. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Pluháčková Hana.

Unified Reporting for Performance Testing

Declaration

I hereby declare that this master's thesis was prepared as an original author's work under the supervision of Mgr. Bc. Hana Pluháčková. Mgr. Martin Večeřa provided the supplementary information. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
Martina Kůrová
May 24, 2017

Acknowledgements

I would like to thank my academic supervisor, Hana Pluháčková for her guidance and patience. By sharing her experience and giving me advice, she helped me a lot in shaping this thesis into its current form. I would also like to thank my consultant, Martin Večeřa for giving me the opportunity to work on this topic, positive attitude, and lots of valuable suggestions throughout the work in this thesis.

Contents

1 Introduction **3**

2 Performance of Software Systems **5**
2.1 Performance Measurements . . . 5
2.2 Performance Issues . . . 6
2.3 Performance Metrics and Problem Indicators . . . 7
2.3.1 Response Time . . . 7
2.3.2 Throughput . . . 10
2.3.3 Start-up Time . . . 10
2.3.4 Resource Usage . . . 11
2.3.5 Memory Problem . . . 11

3 Anomaly Detection **12**
3.1 Mathematical Background . . . 12
3.1.1 Measuring the Central Tendency . . . 12
3.1.2 Measuring Spread . . . 13
3.1.3 Understanding Numeric Data - Distributions . . . 14
3.2 Statistical Methods . . . 15
3.2.1 Regression Analysis . . . 16
3.2.2 Correlation Analysis . . . 17
3.2.3 Basic Statistical Tests . . . 19
3.3 Methods Based on Machine Learning . . . 20
3.3.1 Neural Networks . . . 20
3.4 Methods Based on Data Mining . . . 21
3.4.1 Classification - Findings Patterns . . . 21
3.4.2 Clustering - Finding Natural Groupings of Data . . . 21

4 PerfCake Testing Tool **23**
4.1 Architecture . . . 23
4.1.1 Reporting Component . . . 25
4.2 Monitored Performance Issues . . . 26
4.3 Reporting Capabilities . . . 27
4.4 Performance Test Configuration . . . 27

5 Design of Anomaly Detection Mechanism **29**
5.1 Suspicious Results Detection . . . 29
5.2 Performance Issues Identification . . . 30

6 Implementing the Destination Component **31**
6.1 Crystal Destination Component 31
6.2 Definition in a Test Scenario 32
6.3 The Main Procedure 33
6.4 Results Analysis 34
6.5 Report Generation 36
6.5.1 Results Evaluation 36

7 Testing **38**
7.1 Experimenting and Verification 38
7.1.1 Verification by Simulated Broken Service 38
7.1.2 Weaver Configurations 38
7.1.3 PerfCake Scenario Definition 39
7.2 Achieved Results 40

8 Conclusion **44**

Bibliography **47**

Appendices **49**

A Content on DVD **50**

Chapter 1

Introduction

Performance is a major issue during the development of large scale, multi user enterprise applications. It must be said from the outset that a performance testing is not a cheap affair and therefore, many small companies with a limited budget simply do not test their applications and rely primarily on feedback from users. Very few software projects are delivered early, so there are usually significant time pressures. However, performance testing can draw attention to what needs to be improved before going into production. The wrong decision about going live with a website or application could damage the financial results, brand, or even the viability of the company. More importantly, software performance testing is an extremely significant issue for many large industrial projects and essential in critical applications such as space programs or emergency medical facilities. In such cases, it is necessary to ensure that they work properly for a long time and without deviations. In the past, software developers had to be extremely careful when developing their applications as resources were often scarce and the management of such scarce resources was a complex issue. Modern advances in software technologies, however, have allowed for developers to concentrate less on issues such as performance and resource management, and instead spend more time developing the functionality of applications. Today's application development revolves around enterprise level component frameworks, like Java Enterprise Edition, whereby the framework can be expected to handle complex underlying issues such as security, persistence, performance, and concurrency. The idea is to allow developers to concentrate more on the application functionality such that the time to market is reduced. Consequently, developers do not have enough knowledge about application performance testing, thus performance analysis and optimization become more difficult and create a need for advanced performance tools to provide more sophisticated outputs. Fast interpretation of the measured values and detection of possibly occurred performance issues are also a significant feature that helps developers to evaluate an application performance in a short time. This enables bottlenecks to be identified quickly and provides their resolution with plenty of time left in the release cycle before everything is rolled into production. This thesis investigates typical performance problems of today's applications (e.g. spikes in response time, limited throughput, etc.) and offers approaches on how to automatically detect them. Using statistical methods like regression and correlation analysis, the diagnosis of measured values is performed with the aim to recognize suspicious results and identify frequently occurring performance issues. The developed approach is based on statistical methods, which have the advantage that they do not need to go through the training phase and they can analyze the results immediately. This work was created with the cooperation of QE engineers from the Red Hat company. A new reporting component was created and implemented into a lightweight performance tool and a load generator called PerfCake. The developed component is responsible for detecting and reporting possible issues and their probability. A unified report from all pre-specified measurements is created in such a way that all detected performance issues are immediately visible. The main contributions of this work are, (a) a proposed approach for automatic detection of often occurring performance issues in software applications, (b) a new reporting component integrated into a PerfCake performance testing tool, implemented this approach and (c) an option to generate a unified report, providing developers with a clear overview of the status of the system in terms of performance – a list of possibly occurred issues and their probability. The thesis is structured as follows: Chapter 2 gives an overview of today's application performance issues, how they can be recognized and what are their indicators. Chapter 3 provides an overview of a key mathematical principle which are crucial for anomaly detection in large datasets. It outlines different methods for anomalous profiles detection, with particular focus on statistical methods. In this chapter, we also give an overview of related work. Chapter 4 provides information about a performance testing tool PerfCake, to which the approach is implemented. It introduces a performance testing tool PerfCake, its architecture with a description of each component, and possibilities of a test configuration and execution as well as its testing and reporting capabilities. Chapter 5 gives a more detailed overview of our approach. It shows how regression and correlation analysis can be used to identify deviations in performance results (measured values during a performance test). Chapter 6 describes the implementation details about a new reporting component integrated into PerfCake testing tool, which serves to detect and report possible issues and their probability. Chapter 7 discusses our methodology and outlines a number of criteria that we use to validate our work. It presents different sets of results from a range of tests that we have performed to validate our approach. Finally, Chapter 8 gives our conclusions and ideas for future work in this area.

Chapter 2

Performance of Software Systems

Performance testing is done to make sure an application runs fast enough to keep a user's attention and interest. A slow running application is at risk of losing potential users. Most performance issues revolve around speed, response time, load time and poor scalability. Speed is often one of the most important attributes of an application and therefore also a significant factor of many common performance problems. This chapter summarizes the most frequently occurred performance problems in server applications. The main focus is to recognize specific patterns through which the problems are expressed. Using this knowledge, we are able to identify deviations differing from normal behaviour.

2.1 Performance Measurements

When we talk about performance testing, we think about a testing technique that is performed to determine a system responsiveness and stability. During a test, a system's behaviour is monitored under various workloads. Based on the results of the test, it is possible to determine whether the system is stable and efficient, or some performance issues are occurring in the system. The main goal of performance testing is to ensure system's reliability, responsiveness, and scalability. Following, is an explanation of their significance to clarify what they mean in the context of performance testing:

· ***Reliability, Stability*** – *The ability of a system to perform its functions under certain conditions for an acceptable period of time.*

· ***Responsiveness, Speed*** – *The ability of a system to react quickly, carry out its operations without undue delay and provide an appropriate response.*

· ***Scalability*** – *The ability of a system to behave properly under varying loads and handle a growing amount of work - increased and expanding workload.*

Software is created for various purposes and has different requirements for performance regarding quality and quantity, but we usually monitor the following performance characteristics, base on: the throughput requirements of the system, its response in various situations, recovery rate, resource usage (memory, CPU, disk, etc.), and also the time of the first run of the application.

2.2 Performance Issues

Performance issues occurring in some software can be viewed from two perspectives. We can either talk about problems which occur only under certain circumstances – mostly under heavy unexpected load or those that are visible only after a certain time running the application – called long-term problems. From the survey of frequently occurring performance issues in server applications, we have selected those that are characterized by the manifestation of specific patterns, and where we are likely to be able to identify deviations that differ from normal behaviors.

Traffic spike

An example that directly affects system performance is the sudden degradation of the metric – whether it's response time, bandwidth, or resource usage, caused by a large number of users in the operating spikes. Increased traffic is a good thing from a business point of view, but bad user experience can discourage customers from using the service and have a bad impact on the reputation of the application in general. A few examples of major traffic spikes from the life are, for example, after a successful marketing promotion, discount events, viral video knows, or we can pinpoint cases directly from the academic environment such as registration of subjects, sports, or the start of selling tickets to the highly anticipated upcoming events – whether it's a ball or a university's football tournament.

Sudden shortage may occur in several performance metrics and can culminate in the following problems:

- long response time,
- bad throughput (number of requests per second),
- limited concurrency (the maximum number of concurrent connections)

Spike testing can be used for simulating spikes by suddenly increasing or decreasing the load, perform dramatic changes in order to determine whether there will be degradation of a system. Another frequently used technique is *stress testing* with the objective, to find how a system behaves in extreme conditions. It involves testing an application under extreme workloads to see how it handles high traffic or data processing. The objective is to identify the breaking point of an application using any set of extreme conditions – whether doubling the number of users, using a database server with much less memory, or a server with a weaker CPU, and to inspect what will the user experience be like. Whether the system starts throwing out errors, the response time will double, or the entire system will get stuck and crash. When testing To continue with the previous example, Instead of stopping at a certain number of users, when the page time exceeds the goal, we would continue to increase the user load on the system with the aim to discover the upper limit when the system still functions and when it starts to deteriorate.

Performance Degradation

When we talk about system performance degradation, we mean such problems in performance, that surface only during a long duration of time of running service. This group includes performance problems such as memory leaks or inability of a software to handle the expected load over a long period of time.

These problems can be exposed by testing techniques called soak testing, which measures the system's stability over time by placing it under load for an extended period. Quite often, the 'standard' load testing, which is run for a short limited time will not succeed uncovering all problems. A production system typically runs for days, weeks and months. For example, an application must be available 24h, and a stock exchange application will run continuously on work days. The duration of a soak test should have some correlation to the operational mode of the system under test. Typical problem scenarios that may occur are the following [3]:

- a constant degradation in response time when the system is run over time,
- any degradation's in system resources that are not apparent during short runs, but will surface when a test is run for a longer time
  - memory consumption
  - free disk space
  - machines handles
- any periodical process that may affect the performance of the system, but which can only be detected during a prolonged system run
  - a backup process that runs once a day
  - exporting of data into a 3rd party system.

2.3 Performance Metrics and Problem Indicators

Several important metrics should be monitored when we measure the performance of a system. This section deals with the description of common metrics that are monitored when it comes to system performance monitoring, including considering and comparing applications written in different languages. There are several indicators through which some performance issue can be reflected. To detect the performance issue, we investigate these system's metrics:

- response time,
- throughput (number of processed requests per unit of time),
- the maximum number of concurrent connections,
- utilization of system resources (CPU, memory, network, disk).

2.3.1 Response Time

Response time is the most common and useful performance measures, easy to log, and monitor for potential performance issues. Response time is the amount of time from when a user enters a request until the first character of the response is received. Generally, this should be very quick. Again if a user has to wait too long, they lose interest.

Spikes in Response Time

Regular fluctuations in response time, as shown in Figure 2.1, can be caused by the garbage collector (GC). This is not a problem if GC runs only for a few seconds, such as every minute, as is typical of applications that often store data from databases (disk) to the JVM (memory). The problem arises when the GC takes longer than just a few seconds. In this case, the application may appear slow to the end user.

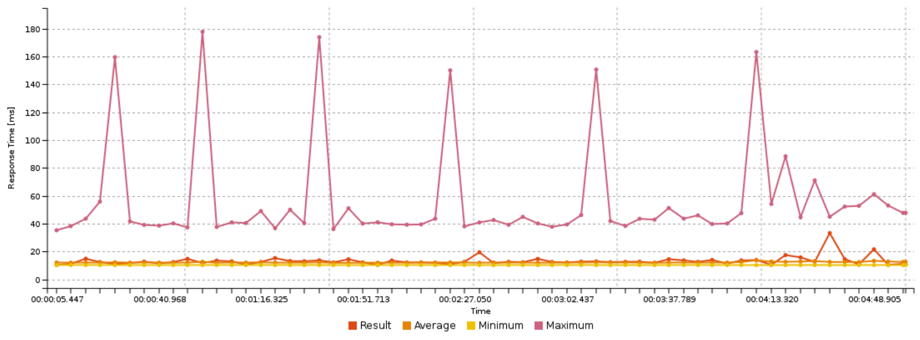


Figure 2.1: Regular spikes in response time

Means vs. Percentile

There are two ways of measuring response time [11, p. 26]. Response time can be reported as an average: the individual times are added together and divided by the number of requests. Response time can also be reported as a percentile request, for example the 90th% response time, as shown in Figure 2.2. If 90% of responses are less than 4 seconds and 10% of responses are greater than 4 seconds, then 4 seconds is the 90th% response time. One difference between the two numbers is in the way outliers affect the calculation

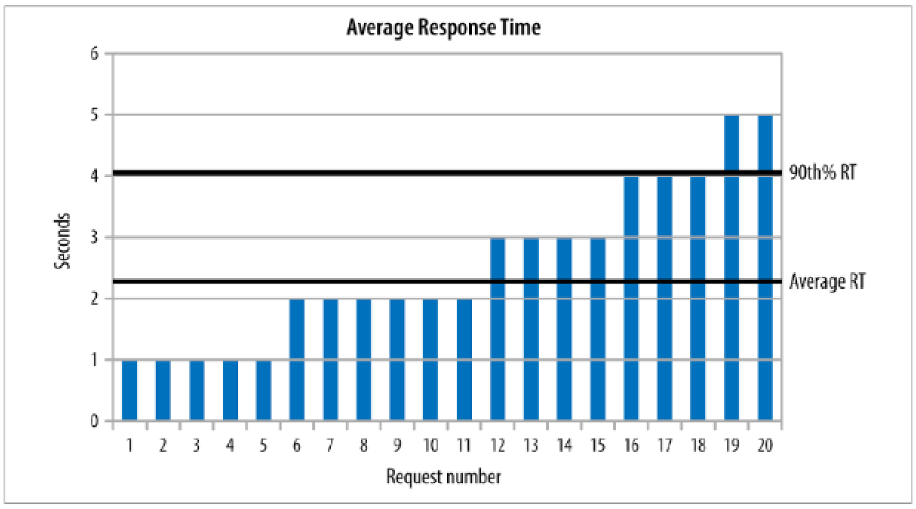


Figure 2.2: Typical set of response times (taken from [11])

of the average: since they are included as part of the average, large outliers will have a large effect on the average response time. Latencies are often captured using HDR Histogram, which observes the complete latency distribution and allows us to look, for example, at 'six nines' latency.

Service Time vs. Response Time

The best way to describe service time vs. response time is to think of a cash register. The cashier might be able to ring up a customer in, under 30 seconds 99% of the time, but 1% of the time, it takes three minutes. The time it takes to ring up a customer is the service time, while the response time consists of the service time plus the time the customer waited in line [2]. Thus, the response time is dependent upon the variation in both service time and the rate of arrival. When we measure latency, we want to measure response time rather than service time, because that's what our users experience. This is a common problem with a lot of benchmarks, that measure service time rather than response time and it creates a so-called coordinated omission problem. A common tool for benchmarking

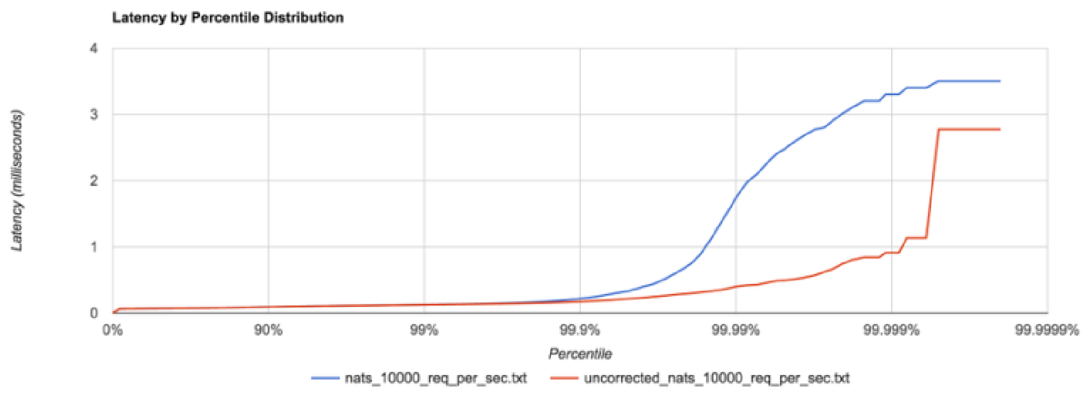


Figure 2.3: Coordinate omission problem (taken from [2])

usually sends a requests continually and records and logs the response time for each request, and builds a histogram out of the recorded response time. This approach works well, but makes an assumption, it does not take into account the stalling of the system under test, (all systems will stall at some point). Suppose our test tool sends 10 requests/sec, and our system process each request within 1 millisecond, but after sometime, say our system stalls for 10 seconds (for the sake of simplicity, say due to garbage collection), during this '10 seconds stall' time, our test tool has only sent one request, as it is also stalled waiting for the response. This long operation has only been recorded once. But in actuality, during this 10 seconds stall, the system would have delayed 10 req/sec x 10 sec = 100 requests, but this was not recorded by the testing tool as the testing tool was waiting for the first request.

HDR Histogram attempts to correct the coordinated omission by filling in additional samples when a request falls outside of its expected interval [2]. The result of the correction of the coordinated omission problem is shown in Figure 2.3, where we see the differences between the corrected and uncorrected values. We can also deal with the coordinated omission by simply avoiding it altogether—always issue requests according to the schedule.

2.3.2 Throughput

Throughput is a number of requests per second that a system can handle. Throughput measurements are almost always taken after a suitable warm-up period, particularly since what is being measured is not a fixed set of work [11, p. 25]. To properly test throughput, we should measure the performance of a system under a specified level of load and examine how a system behaves with a large number of users and what the response time received for pages is, under different scenarios. A production environment typically has a varying number of active users throughout the day. *Quality testing* ensures the application performs well under small loads and peak (for example, Black Friday) loads.

Concurrency Problems

A software product suffers from poor scalability when it cannot handle the expected number of users or when it does not accommodate a wide enough range of users, as shown in Figure 2.4. As mentioned in [3], *scalability testing* should be done to be certain that the application can handle the anticipated number of users and helps identify the maximum capacity of users the system can support, while not exceeding a max page time you defined. Basic indicators, which are investigated, are:

- Maximum active sessions (the maximum number of sessions that can be active at once).
- Thread counts (an application's health can be measured by the number of threads that are running and currently active).

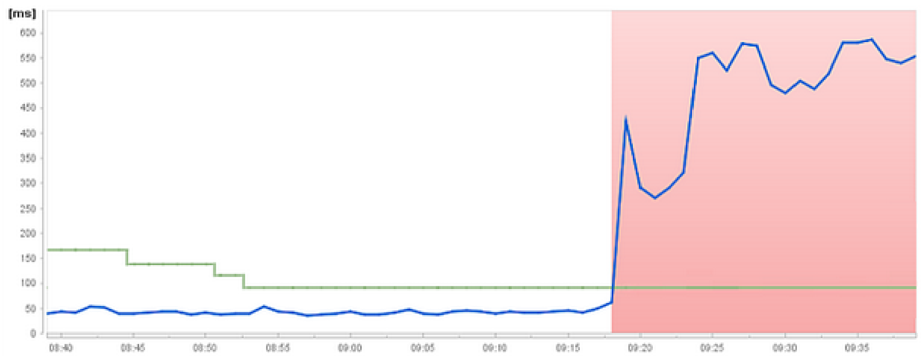


Figure 2.4: Concurrency problem

2.3.3 Start-up Time

The load time is normally the initial time it takes an application to start. This should generally be kept to a minimum. While some applications are impossible to make load in under a minute, load time should be kept for a few seconds if possible.

Just-in-time compilation

In the non-Java world, this testing is straightforward: the application is written, and the time of its execution is measured. Java start-up time is often much slower than many languages, including C, C++, Perl, or Python, because many classes (and first of all classes from the platform Class libraries) must be loaded before being used. In the Java world, there is one wrinkle to this: just-in-time compilation. Essentially, it means that it takes a few minutes (or longer) for the code to be fully optimized and operate at peak performance. For that (and other) reasons, performance studies of Java are quite concerned about warm-up periods: performance is most often measured after the code in question has been executed long enough for it to have been compiled and optimized [11].

2.3.4 Resource Usage

On servers, there is a limited amount of resources available for use at any one given time, for all users. Common monitored system resources regarding their utilization are: CPU utilization, Memory utilization, Network utilization, Operating System limitations, Disk usage.

2.3.5 Memory Problem

Java memory management is challenging and can lead to all kinds of performance issues. In most cases, a C++ application will consume less memory than an equivalent Java application due to the large overhead of Java's virtual machine, class loading, and automatic memory resizing [11]. Anyway, the two most common memory issues are garbage collection configuration and memory leaks.

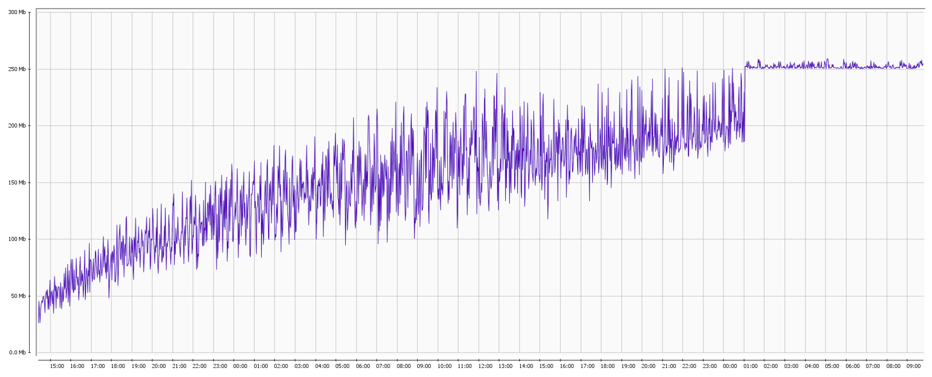


Figure 2.5: Memory usage

Garbage Collection Configuration

Garbage collection statistics have to do with returning unused memory back to the system. Garbage collection needs to be monitored for efficiency. Garbage collector should not be invoked more frequently than every 30 seconds. In other words, the percentage of the time an application is stopped because garbage collection is occurring should be as low as possible, ideally less than 2-3% [11].

Memory Leaks

The cause of the performance problem may be the case where no unused memory is released. Memory leakage occurs due to bugs in the program, whether it is a service written in a language with automatic memory management or not. The graph of Figure 2.5 serves as an example of a gradually increasing amount of memory used.

Chapter 3

Anomaly Detection

Detection of anomalies consists of defining the normality of the behavior by using a set of selected variables, which are then compared with the newly measured ones. If they are significantly different from the expected values, anomaly is reported. The principle is to monitor the functional dependency of the measured metric on time and thus, identifying its deviations from normal behavior. From the statistical point of view, it is the search for correlations, i.e., the investigation of mutual relationships or models of data. The intent is to analyze data dependencies, identify trends, and if possible, predict future values.

Various methods from simple tabulations and visualizations to sophisticated approaches such as genetic programming are used. Methods for anomalies detection can be divided into several basic categories according to the principle on which they are based. However, the most commonly used ones are decision trees, association rules, neural networks, regression and clustering.

3.1 Mathematical Background

Performance analysis and reporting are particularly math-intensive, so it is important to know how to apply mathematics and interpret statistical data to understand the results of performance testing. This subsection describes the most commonly used key mathematical principles and often misapplied mathematical and statistical concepts in the context of performance testing. More detailed description of statistical functions can be found in [9].

3.1.1 Measuring the Central Tendency

Measuring of central tendency [6] is a class of statistics used to identify a value that falls in the middle of a set of data.

Mean

An average – also known as an *arithmetic mean* – is probably the most commonly used statistic of all. To calculate an average, it is needed to simply add up all the numbers a_i and divide the sum by the quantity of numbers n that have been added.

When it comes to performance testing, it is good to be careful to avoid misinterpretation of the results. For example, regarding application response times, we cannot take the average as meaningful quantity when it is the only one reported statistic. It is a good idea to also include the sample size, minimum value, maximum value, and standard deviation and take them into account when analyzing a report from a performance test. Looking at only the average of the measurement of the response time, it may seem that everything can look fine and that we met the goal. But looking at the data closely, however, it may show that not every value of the set meets the given threshold, and therefore, some performance anomaly has occurred. This is because the mean is affected differently by values falling at far ends of the range. In particular, the mean is highly sensitive to *outliers*, or values that are atypically high or low relative to the majrity of data [6].

Percentage

Percentages are used, for example, to determine what was the response time measured in the 95 percent of the cases. To find this 95th percentile value when we performed a hundred measurements at all, it is needed to sort the measurements from largest to smallest and then count down six values from the largest. The 6th value represents the 95th percentile of those measurements. For response times, this statistic is read *„95 percent of the simulated users experienced a response time of [the 6th-slowest value] or less for this test scenario“* [8].

Medians

A median is simply the middle value in a data set when sequenced from lowest to highest.

Normal Values

A normal value, also known as *modus* is the single value that occurs most often in a data set.

3.1.2 Measuring Spread

To understand and interpret large numeric data [6], it is useful to know how to measure the spread - variance and standard deviation. When interpreting the variance, larger numbers indicate that the data are spread more widely around the mean. The standard deviation indicates, on average, how much each value differs from the mean. Variance and standard deviation are sometimes called *measures of variability*, in this sense, the term variability understands distractions.

Variance

Variance [6] is a measurement of the spread between numbers in a data set. It is defined as the average of the squared differences (to make them positive) between each value and the mean value. Variance is calculated by taking the differences between each number in the set and the mean, squaring the differences (to make them positive), and dividing the sum of the squares by the number of values in the set.

$$\sigma^2 = (1/n)[(x_1 - \mu_p)^2 + (x_2 - \mu_p)^2 + ... + (x_n - \mu_p)^2] \tag{3.1}$$

Simply put, the variance measures how far each number in the set is far from the mean value. In mathematical notation, the variance of a set of n values of x is defined by the following formula.

$$Var(X) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \tag{3.2}$$

Where:

· x_i represents the individual data point,

· μ is mean of data points,

· n is total of data points.

A drawback to variance is that it gives added weight to numbers far from the mean (outliers), since squaring these numbers can skew interpretations of the data. The drawback of variance is that it is not easily interpreted and the square root of its value is usually taken to get the standard deviation of the data set in question.

Standard Deviations

By definition, a standard deviation is the amount of variance within a set of measurements that includes approximately the top 68 percent of all measurements in the data set [8]. In other words, when we know the standard deviation of the measured results, we can tell how densely the individual measurements are clustered around the average. The standard deviation [6] is the square root of the variance, and is denoted by *sigma* as shown in the following formula:

$$StdDev(X) = \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{3.3}$$

The smaller the standard deviation, the more consistent measured values we have. A common rule in this case is: *„Data with a standard deviation greater than half of its average should be treated as a suspect“* [8]. With the knowledge of this rule, we can also decide whether the collection of measured values is normally distributed or not.

3.1.3 Understanding Numeric Data - Distributions

Distributions allow us to characterize a large number of values using a smaller number of parameters. The normal distribution, which describes many types of real-world data, can be defined by just two: center and spread. The center of the normal distribution is defined by its *mean* value, mentioned in 3.1.1. The spread is measured by a statistic called the *standard deviation*, which we have defined before in 3.1.2.

Uniform Distributions

Uniform distributions [8] – also known as rectangular distributions – represent a collection of data that is more or less equivalent to a set of random numbers evenly spaced between the upper and lower bounds. In a uniform distribution, every number in the data set is represented approximately the same number of times. In other words, they have the same probability density. Regarding mean (μ) and variance (σ^2), the probability density function of the uniform distribution may be written as follows:

$$f(x) = \begin{cases} \frac{1}{2\sigma\sqrt{3}} & \text{for } -\sigma\sqrt{3} \leq x - \mu \leq \sigma\sqrt{3} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

Uniform distributions are frequently used when modeling user delays, but are not common in response time results data. In fact, uniformly distributed results in response time data may be an indication of suspect results.

Normal Distributions

Normal distributions [6], also known as Gaussian distribution, are data sets whose member data are weighted toward the center (or median value), resulting in a bell-shaped distribution of data. It means that some values are seemingly far more likely to occur than others. The probability density of the normal distribution is:

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\sigma^2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.5}$$

Where:

- μ is mean or expectation of the distribution (and also its median and mode),
- σ is standard deviation,
- σ^2 is variance.

Statistically speaking, most measurements of human variance result in data sets that are normally distributed. As it turns out, end-user response times for web applications are also frequently normally distributed [8].

3.2 Statistical Methods

In statistical methods for anomaly detection, the system observes the activity of subjects and generates profiles to represent their behavior. Statistical approaches to anomaly detection have a number of advantages. Firstly, these approaches do not require prior knowledge of the performance issues themselves. However, statistical anomaly detection schemes also have drawbacks. Statistical anomaly detection can accept the abnormal behavior as normal. It can also be difficult to determine thresholds that balance the likelihood of false positives with the likelihood of false negatives. Also, statistical methods need perfect statistical distributions, but, not all behaviors can be modeled using purely statistical methods. In fact, a majority of the proposed statistical anomaly detection techniques require the assumption of a quasi-stationary process, which cannot be assumed for most data processed by anomaly detection systems [12].

There are a large number of algorithms based on statistical methods. This is also because these methods are used in a wide range of disciplines. The work [7] discusses the use of correlation and regression analysis to detect security incidents within the Internet of Things. Similarly to performance testing, we have the most important deviations from the normal predicted scenario, as is the case with network traffic analysis to detect anomalies and thus detect Attack [15, 5]. Regression analysis is also an important method in sociological research.

3.2.1 Regression Analysis

Regression analysis methods are used for estimating the relationships among variables. The focus is on the relationship between a single numeric dependent variable (the value to be predicted) and one or more numeric independent variables (the predictors) [6]. More specifically, regression analysis helps one understand how the typical value of the dependent variable changes when any one of the independent variables is varied, while the other independent variables are held fixed. The objective of regression analysis is to describe this dependency using a suitable mathematical model. For example, an approximation of data values by a line, as noted in Figure 3.1. Because problems of this type occur so frequently in many branches of engineering and science, regression analysis is one of the most widely used statistical tools [8].

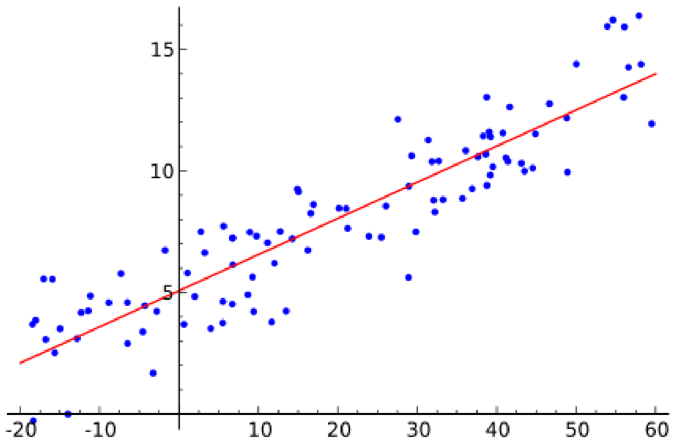


Figure 3.1: Regression analysis

Depending on the number of independent variables, we distinguish models of simple regression and multiple regressions. Simple regression describes the dependency of the explaining variable on one regressor. Instead, multiple regression solves the situation where the dependent variable depends on more than one regressor. This work deals with a simple regression when the explanatory variable is dependent on only one regressor.

Simple linear regression

Simple linear regression [6] defines the relationship between a dependant variable and a single independent predictor variable, using a line denoted by an equation in the following form:

$$y = b_0 + b_1 x + e_i \tag{3.6}$$

Where y (dependent variable) is the measured variable and x (independant variable – regressor). The parameter that determines the position of the line is denoted as b_0, the slope of the line as b_1. e_i represents a random model error.

Least squares method

The least squares method is a method for estimating the regression function parameters, and particularly for those models that are linear in the parameters. The essence is the approximation of the (measured) values by some function from the prescribed space. The simplest example is data interleaving (approximation) with linear - therefore linear function, as noted in Figure 3.2. The goal is to find a line so that the sum of the squares of errors e_i is minimal.

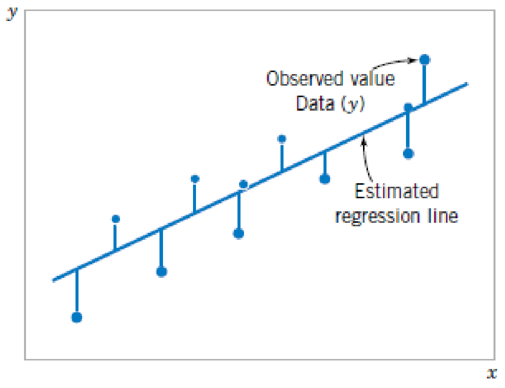


Figure 3.2: Deviations of the data from the estimated regression model (taken from [9])

This means that the parameters of $y'_i = b_0 + b_1 x_i$ (values b_0 and b_1) are searched so that the sum of squares of observed values Y_i from values $\bar{Y}_i$ is the smallest. For a given regression function, this sum is called the residual sum of squares (S_{rez}).

$$S_{rez} = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2 \tag{3.7}$$

From the minimum squares condition, normal equations are derived, from which unknown parameters are calculated by their solution b_0 a b_1.

$$b_0 = \bar{y} - b_1 \bar{x} \tag{3.8}$$

$$b_1 = \frac{S_{xy}}{S_{xx}} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{3.9}$$

3.2.2 Correlation Analysis

Correlation analysis is used to express the dependence of two or more numerical variables. The correlation between two variables [6] is a number that indicates how closely their relationship follows a straight line. From the statistical point of view, it can answer questions like, 'How strong is the dependence between the variables?' or 'How much does the model correspond with reality?'.

Pearson's correlation coefficient r

The most commonly used for measuring dependence is the **Pearson correlation coefficient** r, which measures the linear dependence of two random variables x and y with a two dimensional normal distribution.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{3.10}$$

Where:

x_i:	is the x-coordinate of the data point (Independent variable)
$\bar{x}$:	is the average value of the x values
y_i:	is the y coordinate of the data point (Dependent variable)
$\bar{y}$:	is the average value of y values
N:	is the number of values

It takes values from -1 to 1, which denotes the perfect linear relation (negative or positive) as shown in the following Figure 3.3.

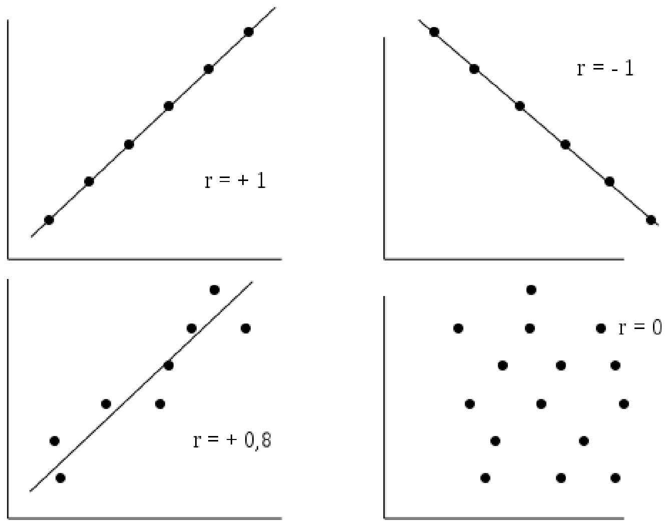


Figure 3.3: An example of the relationship of the data distribution to the value of the correlation coefficient

In the case of a positive correlation, $r > 0$ the values of both variables are also rising. In the case of a negative correlation $r < 0$, the value of one variable increases and the second decreases. In the absence of a linear relationship, $r = 0$.

The Determination coefficient R^2

The square of the correlation coefficient R^2 is called **determination coefficient**. It expresses the extent of which variance of the dependent variable is explained by changes in the variable independently. It is usually multiplied by one hundred, which is expressed as a percentage. The determinant is the quantity

$$R^2 = 1 - \frac{S_{rez}}{S_{yy}} \tag{3.11}$$

Where:

S_{yy}: is a total sum of squares of data deviations from the mean value.

$$S_{yy} = \sum_{i=1}^{n} (y_i - \bar{y})^2 \tag{3.12}$$

In the linear regression model, with an absolute member is the value R^2 in the interval $[0, 1]$ and indicates what proportion of scattering in observation of the dependent variable was managed by the regression. It indicates the match of the data model. Higher values mean greater regression success.

3.2.3 Basic Statistical Tests

An important part of assessing the adequacy of a linear regression model is testing statistical hypotheses about the model parameters [9]. For linear regression models, we can test either the whole model (using the F test) or the influence of the individual predictors (t test). The simplest zero hypothesis is the equality of any of the regression coefficients as null.

$$H_0 : b_i = 0 \tag{3.13}$$

We test the zero hypothesis that the model does not explain anything (the variables are independent). These hypotheses relate to the significance of the regression. Failure to reject $H_0 : b_i = 0$ is equivalent to concluding that there is no linear relationship between x and Y [9, p. 385]. If the parameter is insignificant, then $b_i = 0$ applies. This situation is illustrated in Figure 3.4.

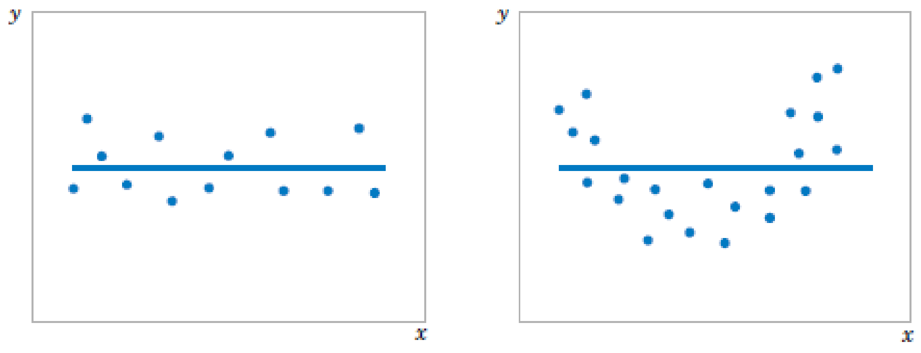


Figure 3.4: The hypothesis $H_0 : b_i = 0$ is not rejected (taken from [9])

Alternatively, if $H_0 : b_i = 0$ is rejected, this implies that x is of value in explaining the variability in Y, as shown in Figure 3.5. Rejecting $H_0 : b_i = 0$ could mean either that the straight-line model is adequate or that, although there is a linear effect of x [9, p. 385].

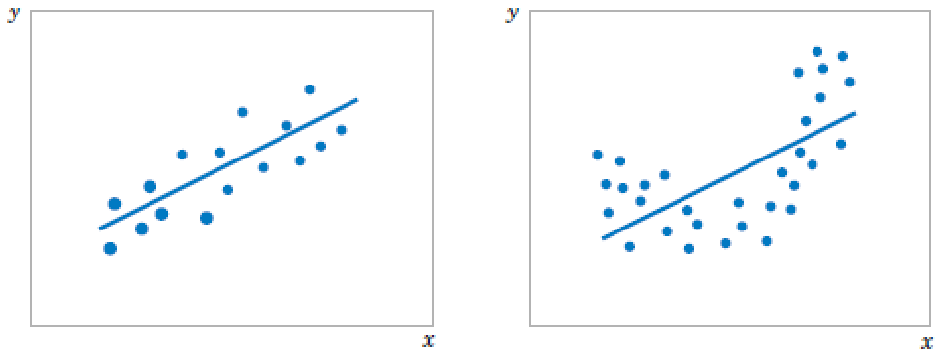


Figure 3.5: The hypothesis $H_0 : b_i = 0$ is rejected (taken from [9])

For tests of the significance of the equalization line coefficients, we refer to partial t-tests.

$$t = \frac{estimated\ parameter\ value}{standard\ error\ of\ the\ mean} \tag{3.14}$$

$$t_i = \frac{b_i}{S_{bi}} \tag{3.15}$$

We compare t_i in absolute value with $t_{krit}(1-\alpha/2)$ Student's distribution for (n-m) degrees of freedom where n is the number of values and m is the number of parameters b_i.

$$t_i = \frac{b_i}{S_{bi}}\ T(n-m) \tag{3.16}$$

The α number is called the level of statistical significance of the test. It specifies the probability that the test characteristic falls outside the scope of acceptance. Typically, it takes values from 0.001 to 0.3 depending on the nature of the problem being investigated (the recommended value is 0.05). If the value obtained is less than 0.05 - we denounce H_0, i.e., we cannot release the b_i coefficient from the model.

3.3 Methods Based on Machine Learning

Machine learning can be defined as the ability of a program or system to learn over a time, enhance their performance on the job. It transforms data into intelligent action. It is not even strictly necessary to understand the theoretical basis of machine learning before using it [6]. Unlike statistical methods, machine learning is not trying to understand the origin of the processes generating data, but instead creates systems that are continually improving based on previous results.

3.3.1 Neural Networks

An Artificial Neural Network (ANN) [6] models the relationship between a set of input signals and an output signal using a model derived from our understanding of how a biological brain responds to stimuli from sensory inputs. Just as a brain uses a network of interconnected cells called *neurons* to create a massive parallel processor, the ANN uses a network of artificial neurons or *nodes* to solve learning problems. Neural network tries to provide the solution to any given problem by learning trends and recognizing problems in the data.

Strengths	**Weaknesses**
· Can be adapted to the classification or numeric prediction problems. · Among the most accurate modeling approaches. · Makes few assumptions about the data's underlying relationships.	· Reputation of being computationally intensive and slow to train, particularly if the network topology is complex. · Easy to overfit or underfit training data. · Results in a complex black box model that is difficult if not impossible to interpret.

Table 3.1: The evaluation of the method neural networks (taken from [6])

3.4 Methods Based on Data Mining

Data mining, sometimes also referred to as knowledge discovery in databases, is the search of useful and interesting information and relationships in large amounts of data. In other words, it's the ability to take an input data and obtain information from them, which are not apparent at first glance.

3.4.1 Classification - Findings Patterns

Classification methods mean that an input data are classified into several classes, based on a set of rules, patterns, or other similar techniques. In the case of anomaly detection, it is usually a binary classification, a division into only two classes – normal data and anomalies.

Algorithms for Deriving Rules

These are methods that can search for relationships between variables in the form of association rules: *„when an X event is discovered, then the event Y is likely to occur“*. The idea is to search for such sets X and Y, that the X implies Y (at least with some significant probability). The Y value is constant and the task is to find a set of X, which predicts the Y value.

Strengths	**Weaknesses**
· Is ideally suited for working with very large amounts of transactional data.	· Not very helpful for small datasets.
· Results in rules that are easy to understand.	· Takes effort to separate the insight from common sense.
· Useful for data mining and discovering unexpected knowledge in databases.	· Easy to draw spurious conclusions from random patterns.

Table 3.2: The evaluation of the method using association rules (taken from [6])

As mentioned in Table 3.2, benefits of using the rules are that they are simple and intuitive, unstructured, and not quite severely. On the other hand, it is difficult to maintain them, and in some cases are not entirely suitable to represent all the information. Association rules are often used for market basket analyses, but they are also helpful for finding patterns in many different fields.

In the context of information technology, association rules data mining technology can be widely used in intrusion detection system to obtain a comparison between the abnormal pattern and the normal behavior pattern [1].

3.4.2 Clustering - Finding Natural Groupings of Data

The basic idea of clustering methods is to find similar objects to each other and gather them into so-called clusters. Objects in one group must be more similar (in some sense or another) to each other than to those in other groups. Clustering [6] automatically divides the data into the cluster without having been told what the groups should look like ahead of time. As we may not even know what we are looking for, clustering is used for knowledge discovery rather than prediction.

The K-Means algorithm

The k-means algorithm is perhaps the most often used clustering method. It involves assigning each of the n examples to one of the k clusters, where k is a number that has been defined ahead of time [6]. The goal is to minimize the differences within each cluster and maximize the differences between clusters.

The algorithm uses a heuristic process that finds the optimal clusters, because computation across all possible combinations of examples is not feasible. Otherwise, this method is simple and easy to implement with a simple interpretation of the clustering results. An overview of the benefits and outcomes can be found in the table 3.3.

Strengths	**Weaknesses**
· Uses simple principles for identifying clusters which can be explained in non statistical terms. · It is highly flexible and can be adapted to address nearly all of its shortcomings with simple adjustments. · It is fairly efficient and performs well at diving the data into useful clusters..	· It is less sophisticated than more recent clustering algorithms. · Because it uses an element of random chance, it is not guaranteed to find the optimal set of clusters. · Requires a reasonable guess as to how many clusters naturally exist in the data.

Table 3.3: The evaluation of the k-means method (taken from [6])

Anomaly detection based on the K-mean clustering algorithm is also frequently used approach in network data analysis to detect intrusions and attacks [10, 4].

Chapter 4

PerfCake Testing Tool

PerfCake is an open source tool for performance testing of server applications and a load generator developed by QE engineers from Red Hat Czech s.r.o. It can be used for monitor response time, throughput, memory consumption, and many others performance metrics. Benefits of using the PerfCake are that it is easy to use, platform independent, and has a minimum influence on the regular system. It provides users with commonly used functions such as load generation, measured values accumulation, and results reporting. Recently, additional features like CSV reports and warm-up detection were added to the tool. It has a component design, so it is easy to customize the test process as much as possible to specific requirements.

4.1 Architecture

As shown in Figure 4.1, PerfCake basic architecture is composed of nine blocks – `Generator`, `Sender`, `Message`, `Sequence`, `Receiver`, `Correlator`, `Reporter`, `Destination`, and `Validator`. All of them have their unique and irreplaceable role in the system. In this section, we will provide a quick description of each of these components.

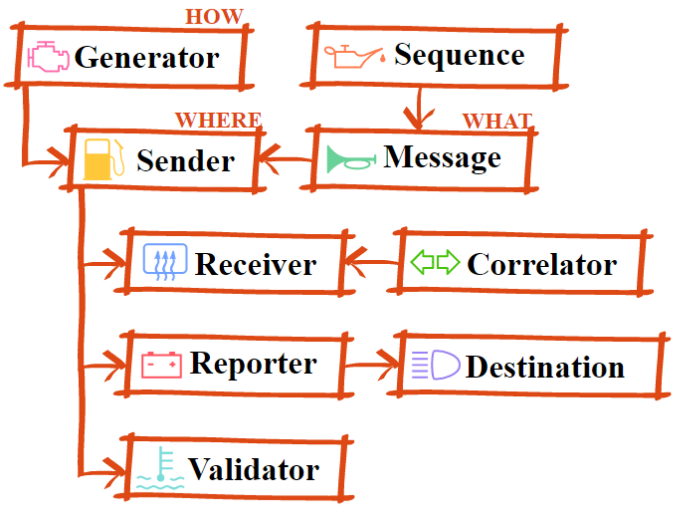


Figure 4.1: PerfCake architecture (taken from [13])

Generator

A Generator component is capable of generating a load for a specific duration, which can be an amount of time or a number of iterations specified by number of concurrent threads. This component has the responsibility of generating all the messages and load. It specifies the whole process on how the load or messages are generated – we can send all of them at once or we can add threads gradually. There are various types of the Generator component available and each of them represents a particular technique on how the load (sending the messages) is handled.

Sender

The PerfCake tool supports a broad range of interfaces and protocols (HTTP, REST, JMS, JDBC, SOAP, socket, file, etc.) that could be used to transmit messages to a target service or system under test. A sender has the responsibility for a transport of messages to a target system described by a particular address which must be provided for each protocol.

Message

A message is a basic communication unit that is sent to a target system during a test to measure a (response) time within a valid response was received. Message is also the smallest unit of a load that can be generated to a target system with the aim to investigate the maximum load that a target system can bear.

Sequence

To ensure a diversity of sending messages, we can use a Sequence component that is, a simple interface that returns another value each time it is called.

Receiver

A response to an original request can be cached through a separate message channel by a Receiver and forwarded to a Correlator component.

Correlator

A Correlator then matches the response to the original request based on some specific information, which is contained in both messages (i.e. correlation ID). This is happening because it is possible to send request via some protocol and receive it via a completely different one. At this point, it is feasible to measure a complete request-response cycle.

Reporter

A Reporter component manages reporting of measured values. The component is responsible for monitoring a process of measurement – it accumulates measured values and transforms them into meaningful values (e.g. throughput, memory usage, response time, service time, response time histogram, etc.). Based on intervals specified by a user, measured values are forwarded to a Destination part where they are interpreted in some kind of report.

Destination

A Destination determines an output format of a report, where the measured results will be stored. It could be a chart, CSV file, log file, database, console, etc.

Validator

Finally, there are Validator components that can validate the response to see if a system under test operates correctly. The main task is to check response content and decide whether it is a valid response or just an error message. Validators can also be used to write an end-to-end test of a system.

4.1.1 Reporting Component

This work focuses primarily on report generation. Thus a Reporter and a Destination component will be described more in detail. Figure 7.7 introduces main blocks of a reporting component. It includes a `SenderTask` that measures the time spent on sending a message and creates a `MeasurementUnit`, which is a basic unit representing a single measurement. It is a component that carries all measured values obtained during a test. All MeasurementUnits are collected in a `ReportManager` from where they are broadcast to all its registered Reporters. A `Reporter` then transforms measured values into a meaningful one according to a purpose for which it was created. It could represent report such as average throughput, memory usage, response time histogram, etc.

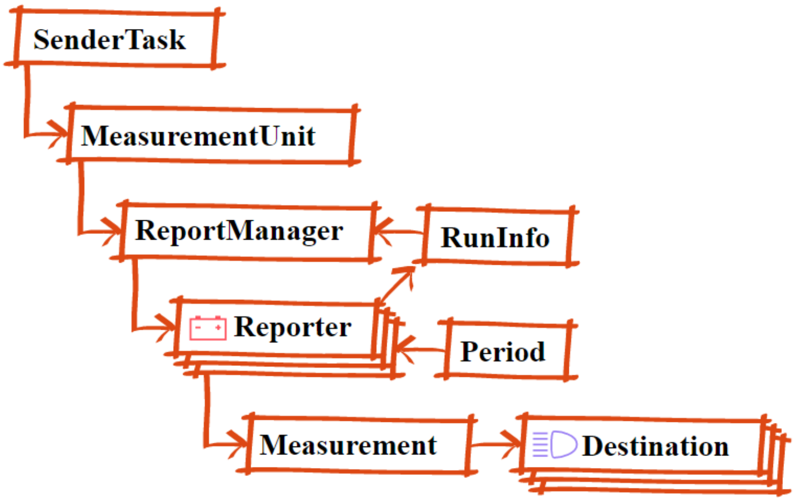


Figure 4.2: Reporting architecture (taken from [13])

A time when a `Measurement` should be reported to an output is specified by a `Period` property, which can be set in a test configuration called `scenario`. The scenario configuration will be more discussed in the following section 4.4. A format of an output is defined by a `Destination` component that represents a place where the results will be reported (CSV file, console, database).

4.2 Monitored Performance Issues

This section provides an overview of performance issues that can be currently monitored via the PerfCake tool.

Memory Consumption of the Target JVM

PerfCake can to measure a current memory usage of a tested system. It requires a PerfCake agent to be installed in the tested system's Java Virtual Machine (JVM). There is a `MemoryUsageReporter` that is capable of possible memory leak detection in such a way that periodically gathers memory usage from the tested system using the `PerfCakeAgent` and remembers a window of N last measured values. Once the window is filled, the reporter uses a linear regression analysis of the data from the time window to compute the used memory trend. The possible memory leak is considered detected when the slope of the memory trend exceeds the specified slope threshold [14]. The reporter is also able to dump memory when a possible memory leak is detected. The memory dump is then saved in a file. The reporter can ask the agent to perform a garbage collection each time a memory usage of the tested system is measured and published. All the features (leak detection, dump or garbage collection) and other appropriate attributes are configurable via particular reporter's properties.

Response time

A reporter called `ResponseTimeStatsReporter` can report statistics - current, minimal, maximal, and average value of a response time (in milliseconds) from the beginning of a measuring to the moment when the results are published.

Throughput

It is possible to measure a plain high-level throughput (in the means of the number of iterations per second) from a beginning of a measuring process to the moment when the results are published. The result is computed from the current number of processed iterations at the moment of publishing result and the time duration from the beginning (or warm-up) [14]. The measured values could be then reported via an `IterationsPerSecondReporter`. However, there is also a more sophisticated way of monitoring the system throughput. When a `ThroughputStatsReporter` component is set in the scenario, it is possible to report a statistic of current, minimal, maximal, and average value of a throughput. There is also an option to measure the throughput over a specified time window.

A System warm-up

A definition of `WarmUpReporter` component in a test scenario configuration provides a possibility to determine when a tested system is warmed up. A reporter internally keeps track of current throughput – each second, checks the number of processed iterations, and computes the current throughput as the difference in number of iterations per checking period (second). It also remembers the current throughput from the previous checking period to calculate a difference in throughput. The throughput is considered NOT changing in time „much“ when the relative difference in current throughput between the current checking period and the previous one is less than the specified relative threshold value or the absolute difference in current throughput between the current checking period and the previous one is less than specified absolute threshold value [14].

Normally, the maximal length of the warm-up period is determined by the length of the performance test itself. This can be further customized by setting adequate properties when the system under test cannot get warmed-up within a reasonable time frame and we do not want to waste time during the test. The system is considered warmed up when all of the following conditions are satisfied: The current throughput is not changing much over the time, the minimal iteration count has been executed and the minimal duration from the very start has exceeded.

4.3 Reporting Capabilities

There are several ways to interpret the results of measured values throughout a test. As mentioned before, an output format of a report could be specified by a `Destination` component. A place where results will be reported could be a console, log file, database, or a comma-separated values file. Having the raw data in a tabular format like this is very beneficial, but it's also helpful to view the results in a graphical format. PerfCake provides a `ChartDestination`, which solves this problem. The option to plot a chart of the results makes it much easier to identify trends.

4.4 Performance Test Configuration

To run PerfCake, it is necessary to supply a test configuration – called `scenario` which consists of several building blocks defining the test process. As seen from the example below, the scenario is an XML file containing various sections. As you may have noticed, XML marks are mostly equivalent to components of the PerfCake architecture. Sections `run`, `generator`, and `sender` are mandatory, the rest of them are optional. It is a significant advantage that the PerfCake has this component design. This provides a high reusability and flexibility of the scenario configuration. It is possible to customize the way in which the test will be executed and use just these building blocks that fit our requirements. It is possible to specify how PerfCake would generate load by configuring a generator, where and what to send by defining a sender and messages. To get any measured results, it is appropriate to use the conveniences of reporting and set this component also in the scenario. Additionally, a validation part is available to check that the responses are valid and have a correct content.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scenario xmlns="urn:perfcake:scenario:7.0">
    <run value="${perfcake.run.duration:500000}"
        type="${perfcake.run.type:iteration}"/>
    <generator threads="${perfcake.thread.count:500}"
        class="ImmediateMessageGenerator"/>
    <sender class="JdbcSender">
        <target>jdbc:postgres:/localhost:5432/postgres</target>
        <property value="org.postgresql.Driver" name="driverClass"/>
        <property value="postgres" name="username"/>
        <property value="password" name="password"/>
    </sender>
    <reporting>
```

```
        <reporter class="WarmUpReporter"/>
        <reporter class="ThroughputStatsReporter">
            <destination class="CsvDestination">
                <period type="time" value="30000"/>
                <property value="${perfcake.scenario}-throughput-stats.csv"
                    name="path"/>
            </destination>
            <destination class="ConsoleDestination">
                <period type="time" value="30000"/>
            </destination>
        </reporter>
    </reporting>
    <messages>
        <message uri="jdbc-simple-select.sql"/>
    </messages>
</scenario>
```

Listing 4.1: XML scenario definition example

As you can see, the scenario runs in 500000 iterations. It generates requests using 500 threads and sends them via jdbc to the (database) server specified in a property. Performance test results are reported to the console every 3 seconds as well as to the CSV file. The content of the messages (sql commands) that are being send id specified in the `jdbc-simple-select.sql` file.

Test Execution

When the scenario is prepared, we have everything to run PerfCake. The Main Requirement is having JDK Installed in our environment. PerfCake is a command line tool, so it provides a command line interface for executing a PerfCake scenario. There is a script `perfcake.sh` to which we have to supply a scenario and execute it by invoking the following command in the PerfCake home directory:

```
$ ./bin/perfcake.sh -s scenarioExample
```

It is the only mandatory parameter and it is not even necessary to specify the file extension, perfcake can recognize a couple of them automatically.

There are also other ways to run PerfCake. The scenario can be included into other application lifecycle using Maven, or it can be directly invoked from a test via PerfCake fluent API. PerfCake is written in Java and Maven manages the project lifecycle, so it is also possible to run it directly by invoking the Java command after building our own distribution from source code.

```
$ mvn clean install -DskipTests
$ mvn exec:exec -Dscenario=src/main/resources/scenarios/scenarioExample
```

Chapter 5

Design of Anomaly Detection Mechanism

When detecting anomalous profiles, we focus on statistical methods, which have the ability to learn directly from the observed data, and therefore do not have to go through the initial training phase, as in more sophisticated methods. This is mainly about the different approaches to trend analysis in the graph and correlation of important values. This chapter follows the theory outlined in Section 3.2. It also introduces additional modifications of the basic algorithms which improved the performance of the methods.

5.1 Suspicious Results Detection

As mentioned in Section 3.2, this work deals with a simple regression when the explanatory variable is dependent on only one regressor. The dependence is linear and therefore the relation of both quantities linear. Analyzing the performance of the application in the context of regression analysis involves examining the dependence of a given performance metric (response time, throughput, the amount of resource consumed) on time – that is, how the measured values change over time. This affects the relation of dependence between these two values, which can be defined by a number of variables by using of correlation analysis. Therefore, in the regression analysis formula 3.6, we would set the tested performance metric as a dependent variable and the time as a independent variable.

The Slope of The Regression Line

From a graph on the figure 2.5 one can see a kind of growth tendency of a line of estimated regression function showing the amount of memory used relative to the test execution time. From a mathematical point of view, we would quantify this tendency, the degree of leakage of the two variables by the parameter b_1, which can be estimated according to the formula 3.9 by the smallest square method. Parameter b_1 here represents the directive – the slope of the estimated regression line, and therefore it can be estimated that there is a growth trend in the graph. On the basis of the knowledge of this parameter, it is, therefore possible to detect the degradation of the measured performance metrics, either from the short-term period of time due to the overloaded operation or gradual degradation over a long period of time when the application is running.

Significance Level of The Slope

The aspects of the linear regression model must be verified in the regression analysis. We will use the basic statistical tests in the regression model according to the chapter 3.2.3. We find the existence of a linear relationship between two variables by asking formally whether the b_1 directive is equal to zero. If the answer to this question is positive, it means that the regression line directive differs from zero simply by chance, i.e. the relationship between the monitored variables is not linear.

When we move into the context of the detection of performance problems again, we can assume that if the relation between the measured variables is not linear, there are such values in the result set of the measured values that are significantly different from the average and may represent, for example, substantial fluctuations as shown in the figure 2.1. In this way, we identify a potential anomalous profile, which can be - with respect to severity - identified as a performance problem.

Quality of The Regression Model

The quality of the regression model can be evaluated using the determination coefficient R^2. As explained in the 3.2.2 chapter, the determination coefficient indicates how many percents of the variance of the explanation variable is explained by the model. In practice, this means how much the individual values are different from the mean value. We will use this knowledge to confirm previous hypotheses about the occurrence of an anomalous profile.

5.2 Performance Issues Identification

The PerfCake testing tool provides a measurement of various metrics, such as response time, throughput, memory usage, etc. In this work, we are concentrating mainly on the response time as it is the most common and useful performance metric, which it is also easy to log. Based on the above-mentioned approaches, the heuristic rules for recognizing suspicious values were created and subsequently tested on a larger sample of data, and thus improved. The rules are described in the following Algorithm 1. We use the algorithm to recognize the patterns that determine the performance issues, that are frequently occurring in case of response time.

Algorithm 1 Performance issues identification

```
procedure AnomalousProfileIdentification(regression model)
    if The value of the regression line directive is significantly different from zero then
        Degradation can be detected;
    else
        if The hypothesis on the significance of regression line coefficients is rejected then
            Regular spikes can be detected;
        else Normal profile;
        end if
    end if
    return detected anomalous or normal profile
end procedure
```

Chapter 6

Implementing the Destination Component

The approaches mentioned in the Chapter 5 were integrated into the PerfCake testing tool as a new reporting component used to detect anomalous profiles in the results of the performance test. After a performance test is finished, using the statistical methods, the developed component performs an analysis of measured values in order to detect suspicious records and provides a report with possibly occurred performance issues and their probability. The developed component provides such a report in which the detected problems are properly highlighted, so they are immediately visible. The main contribution of the developed component is to provide a mechanism that will speed up and simplify analysis of the performance test report that developers would otherwise have to do manually.

6.1 Crystal Destination Component

A new `Destination` component called `CrystalDestination` was created. Table 6.1 provides an overview of the component's properties, which can be used to influence its behavior and the final report form.

Name	**Description**	**Req.**	**Default value**
path	A path to the directory where the report will be stored.	No	`PerfCake-report`
title	Title of a performed test. The title must be unique within the unified report.	No	`PerfCake Test Result`
order	Defines the order of the test result within the final report.	No	`null`
resultsAnalysis	The option to analyze measured values and display results.	No	`true`
simpleStats	The option to display simple statistics overview.	No	`false`
threshold	The threshold used for evaluation of results.	No	`-`
window	The time sliding window size for results analysis (ms).	No	`${perfcake.run.duration}` `(time duration of the scenario in ms)`

Table 6.1: CrystalDestination properties

What we designate as the **test** is every `Reporter` component defined in the test scenario, which manages reporting of the measured values to the specified `Destination`. According to the **path** property, we determine that the results of specific tests should be generated in a single report (HTML document). If we want to create multiple reports, we need to define more than one path. We do not have to set the **title** property if we report only one measurement. Otherwise, all names of all tests must be defined and the names must be unique within the report. If the **threshold** property is not set, the default value, recommended for the monitored metric is selected or comparison with the threshold is not included in the report evaluation part. If the **window** property is not defined, the statistical analysis with the sliding window is skipped and an analysis over the entire data set is performed instead. The window size must be large enough to cover at least two reported records; otherwise, regression analysis cannot be done.

Since all the results in the final report are displayed as charts, they can be influenced using the same properties as in the ChartDestination component, listed in *Table 4.45 - ChartDestination properties* in the PerfCake Users' Guide [14].

6.2 Definition in a Test Scenario

The following XML definition 6.1 demonstrates the example of using the `CrystalDestination` component inside a test scenario. The `ResponseTimeStatsReporter` is used to report measured values to the `CrystalDestination`, and therefore an application's response time is the examined metric in this test. According to the scenario, the response time should not exceed **1 second**, otherwise the problem will be reported in the final report. Based on Algorithm 1, the measured metric will be scanned for performance peaks and degradation over time. A sliding window of size **60 seconds** will be used for this analysis. The final report will be generated into the `ResponseTimeReport` folder. The chart properties will match the properties defined in the scenario, and the analysis of the results will be displayed under the graph. It will contain the identified problems and their probability. An overview of simple statistics will be displayed as well.

```xml
<reporter class="ResponseTimeStatsReporter">
    <destination class="CrystalDestination">
        <period type="time" value="500"/>
        <property name="path" value="./ResponseTimeReport/"/>
        <property name="title" value="Response time"/>
        <property name="order" value="1"/>
        <property name="threshold" value="1000"/>
        <property name="resultsAnalysis" value="true"/>
        <property name="simpleStats" value="true"/>
        <property name="window" value="60000"/>
        <property name="chartYAxis" value="Response Time [ms]"/>
        <property name="chartGroup" value="response"/>
        <property name="chartAttributes" value="Result,Average"/>
    </destination>
</reporter>
```

Listing 6.1: XML scenario definition example with the developed component

In order to perform a results analysis, at least one destination of type `CrystalDestination` must be defined in the test scenario configuration. Results analysis starts directly after the last value from a Reporter is reported to the CrystalDestination and followed by a final report generation. It is possible to affect the number of reports and their content by defining properties in the test scenario. According to the path property, we determine if the results of individual tests should be generated into one or more reports (HTML documents). It can be done by setting the same target path of a final report for components that want their results in a single document. According to this parameter, it is possible to sort the results of individual tests and group them into various documents.

6.3 The Main Procedure

An input of performance test, performed by the PerfCake testing tool, is always some test scenario configuration. The test process starts with reading a test scenario properties and ends with a report generation. After the integration of our component into the tool, the process consisting of the following steps:

- Based on the definition of the scenario, the corresponding numbers of `CrystalDestination` instances are created. The instances share some static fields both for the synchronization of the final report generation and for the storage of significant intermediary outputs.
- When a test is performed within a `Reporter`, the results are generated into our component, where they are stored in a shared data structure under the unique title string of the test.
- If the **resultsAnalysis** property is set, the analysis of measured records is performed using regression analysis and other basic statistics (e.g. mean, 95th percentile value, variance, etc.), which are usually compared with a specified threshold value and evaluations are then reported into the final report.
- Once all analyses are terminated, the final report is generated in an HTML format. The elected "master" destination is responsible for creating it. The result of individual tests are retrieved from the shared data structure and combined into final report according to their specified path property. The key value in the data structure for each test results is the unique title of a test.
- Results analysis evaluations are generated in the final report as well. A pre-created template in HTML is filled with data and then assigned to the corresponding performance test result (chart).
- If the **simpleStats** property is set, an overview of simple statistics is also added to the report. This overview helps to understand the relationship between the measurement values and the results of the result analysis.

Destination Instances Synchronization

Data structure `ConcurrentHashMap` is used for sharing significant data across instances and the `ReentrantLock` mechanism is used to ensure exclusive access to structures. Both are available in *Java concurrency library*[1].

[1]https://docs.oracle.com/javase/7/docs/api/index.html?java/util/concurrent/package-summary.html

6.4 Results Analysis

The analysis of the results is performed for the purpose of detecting suspicious values and identifying the performance issues. The outputs of the analysis are included in the final report. As shown in Figure 6.2, they will appear under the relevant graphs and with information about the performance issues that may have occurred and with what probabilities. Further, other basic statistical parameters are evaluated in order to confirm the performance problem.

Simple Regression Analysis

For the purpose of detecting suspicious values, a simple linear regression was implemented using the `SimpleRegression` class provided by the *Apache Commons Math* library. The output of the regression analysis is a regression model described by several parameters:

· `slope` – the slope of the estimated regression line,

· `intercept` – the intercept of the estimated regression line,

· `r` – Pearson's product moment correlation coefficient,

· `rSquare` – the coefficient of determination,

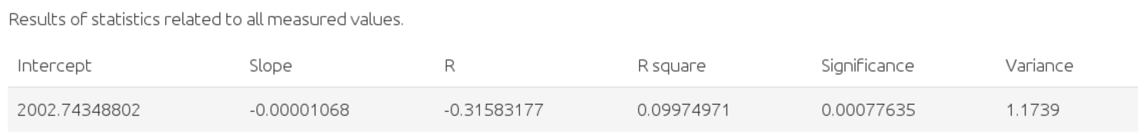
· `significance` – the significance level of the slope.

We used Algorithm 1 proposed in Section 5.2 for performance issues classification, as we have added more specific numbers. These numbers are only estimated values, but several tests have verified their accuracy as stated later in Section 7.1 and confirmed by the results in Section 7.2.

Algorithm 2 Performance issues classification

```
procedure AnalyzeResults(RegressionModel)
    alpha = 0.05;
    if ($slope > 0.4$) then
        if ($rSquare > 0.7$) then
            It is very likely that degradation occurs;
        end if
    else
        if ($slope < 0.001$) then
            It is almost certain that this is a normal profile;
        else
            if ($slope < 0.1$)) then
                if ($rSquare > 0.7$) then
                    It is almost certain that this is a normal profile;
                else if ($significance < alpha$) then
                    It is almost certain that this is a normal profile;
                else if ($significance > alpha$) then
                    It is very likely that peaks occur;
                end if
            end if
            Can not be decided;
        end if
    end if
    return $PerformanceIssueType$
end procedure
```

If the **simpleStats** property is set in the test scenario, an overview of the parameters describing the estimated regression model is also added to the report, as shown in Figure 6.1. From the information in this table we can compile a regression model, since we know both the intercept and the slope value.


Results of statistics related to all measured values.

Intercept	Slope	R	R square	Significance	Variance
2002.74348802	-0.00001068	-0.31583177	0.09974971	0.00077635	1.1739

Figure 6.1: The simple statistics overview

Basic Statistics

To be certain that some value does not exceed some **threshold**, it is possible to defined the maximum accepted value. Every measured value is compared to this threshold and the overall result is evaluated in the end. Other basic statistics (e.g. mean, 95th percentile value, variance, etc.) were computed using the `DescriptiveStatistics` class provided by the *Apache Commons Math*[2] library. These statistics are compared with a specified threshold value and evaluations are then reported into the final report. If the threshold is exceeded, the performance problem is reported in the final report and it is clearly indicated in the chart.

Time Sliding Window

A size of the window is the property which influences the final result the most. That's why there is a possibility to defined it by a user in a test scenario configuration. The type of value should be related to the type of x-axis. For example, if we talk about a `ResponseTimeReporter`, the window size value should be defined in milliseconds, as the x-axis represents time. In the case of this `Reporter`, the size of the sliding window is calculated from the window size, specified in the test scenario:

$$\texttt{slidingWindowSize = window/period} \tag{6.1}$$

Where:

- `period` is the value determining how often the measured values are recorded,
- `window` is the time window size in milliseconds specified in a test scenario,
- `slidingWindowSize` is the number of records which have to be analyzed by statistical methods within a window block.

The window size is now related to the number of performance test records. In any such window, a regression analysis is calculated over the block of records. If the performance issue is detected, it is reported in the final report, including the interval in which it occurred. This will ensure that the issues are clearly highlighted in the chart.

[2]http://commons.apache.org/math

6.5 Report Generation

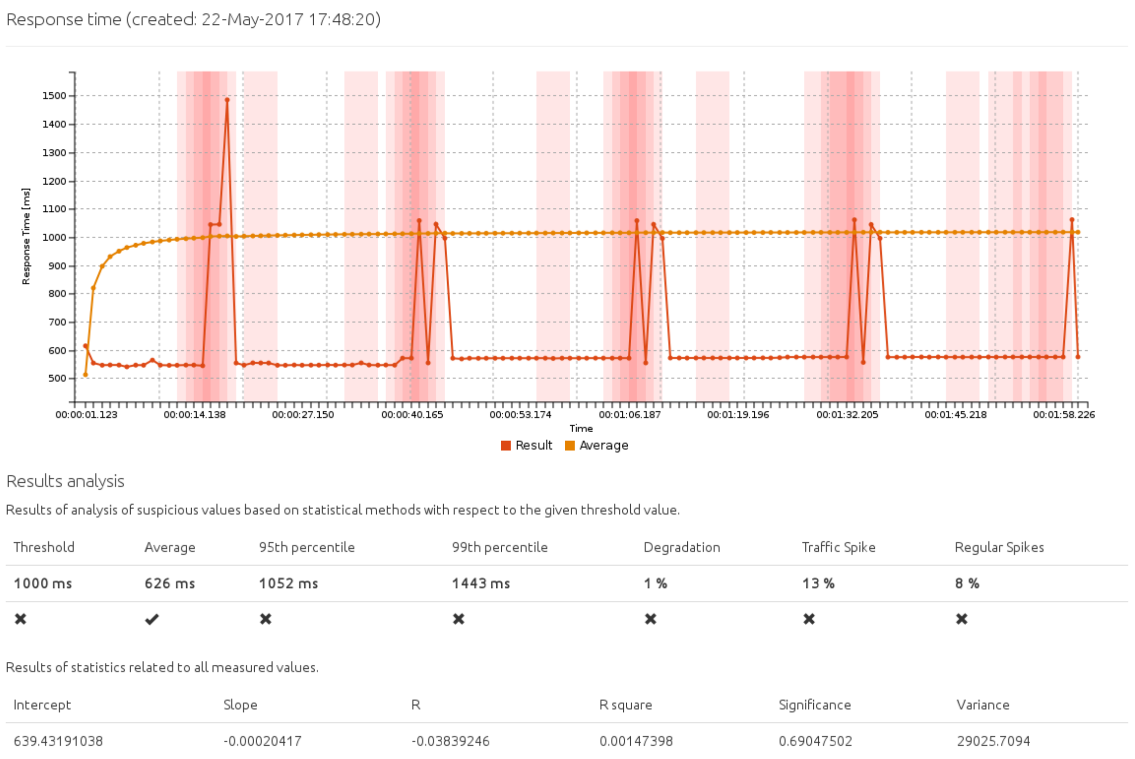
When all the tests results and results analyses are ready, the elected master destination generate the final report. It is simply the one in which performance test finishes as the last one. The destination takes all intermediary outputs and combines them into a single or more HTML documents and adds an evaluation part with a list of possibly occurred performance issues and their probabilities. In relation to the type of test defined in the test scenario, corresponding statistics are reported under a chart.

HTML Final Report

The *C3.js*[3] JavaScript library is used for generating a report in an HTML. It is already used in PerfCake in the `ChartDestinatnion` implementation for publishing the measured result into a chart, so we followed this approach.

6.5.1 Results Evaluation

The evaluation part in the final report provides an overview of detected performance issues in relation to the type of test performed. The type of performed test and monitored metrics are determined by the Reporter component defined in the test scenario.



Results analysis

Results of analysis of suspicious values based on statistical methods with respect to the given threshold value.

Threshold	Average	95th percentile	99th percentile	Degradation	Traffic Spike	Regular Spikes
1000 ms	626 ms	1052 ms	1443 ms	1 %	13 %	8 %
✖	✔	✖	✖	✖	✖	✖

Results of statistics related to all measured values.

Intercept	Slope	R	R square	Significance	Variance
639.43191038	-0.00020417	-0.03839246	0.00147398	0.69047502	29025.7094

Figure 6.2: The final report with results analysis and basic statistics overview

[3]http://c3js.org/

Threshold Value Exceeded

A threshold value can be specified in a test scenario configuration in order to detect, if the maximum acceptable value has been exceeded. As mentioned in 6.4, every measured value is compared to this threshold, so we can be sure, that and the overall result is evaluated in the end.

Average and Percentile Values

As explained in the Chapter 3.1.1, an average value is not very significant in terms of statistics, but it is mentioned just for the possibility to compare it to the most significant metrics and see the difference. The 95th or 99th percentile values are more significant values when it comes to meeting the real response time goals.

Performance Issue Occurrence Probability

As noted in Figure 6.2, the intervals on the x-axis, in which some suspicious value were detected is highlighted in red color. Depending on how often a problem has to be highlighted in a region, its probability is counted.

$$\texttt{probability = occurrence/total} \tag{6.2}$$

$$\texttt{percentage = 100 * probability} \tag{6.3}$$

Where:

- `total` indicates the total number of blocks in which the performance problem was investigated (the total number of sliding windows in which the regression analysis was performed); the window shift is always one value forward, so `total` corresponds to the total number of records in the data set,
- `occurrence` indicates the number of successful detection cases of a given performance problem within all blocks,
- `probability` is the probability of occurred performance problem,
- `percentage` is the percentage value for a more readable form.

Chapter 7

Testing

In this chapter, we present the results of experiments performed with developed component introduced in Chapter 6. First, we describe the tools we used to validate the implemented solution. The configurations of these tools will be provided and then, we show the effect of the new developed component.

7.1 Experimenting and Verification

Based on the knowledge of performance issues and the ability to recognize their characteristic profiles, experiments with the developed method have been performed to increase its accuracy. On larger data sets, the thresholds for comparing with regression analysis results, in particular the slope of a regression line (slope), the coefficient of determination (RSquare) and the level of significance of the slope of the line (significance) have been approximated to more appropriate numbers. These thresholds are crucial for deciding whether or not the performance problem has occurred. However, the solution is still heuristic and its results may vary based on the characteristics and the requirement for the tested system.

7.1.1 Verification by Simulated Broken Service

Several services was used to simulate anomalous profiles for the purposes of testing and optimizing the proposed algorithm. The most appropriate was an open-source service **Weaver**[1] that communicates with the outside world via HTTP protocol. It is possible to run multiple threads – called `Workers`, which perform requests based on their configuration, and thus simulate negative test scenarios. For example, the delayed response time, memory leakage simulation, switching between two different configurations, and simulation of normal behavior. The other services used were web services **mocky.io** [2] and **httpbin.org** [3] for HTTP "request & response" simulation running on a similar principle as Weaver.

7.1.2 Weaver Configurations

Several Weaver configurations have been created to simulate a particular response time performance issue.

[1]http://github.com/PerfCake/Weaver
[2]http://www.mocky.io/
[3]http://httpbin.org/

Regular spikes in response time

To create an anomalous profile containing several regular spikes, the `SwitchingWorker` class was used, that can periodically switch between other workers.

```
50x SwitchingWorker=switchPeriod:3010,worker1_class:NormalWorker,worker1_status \
    Code:200,worker2_class:DelayWorker,worker2_delay:2000
```

Delayed response time

For response time degradation the `DelayWorker` was used to delay the processing of requests, that can be specified by a `period`. We set this parameter to 200 milliseconds. To obtain the anomalous profile, it is necessary to set up more threads in the test scenario than in this configuration.

```
50x DelayWorker=delay:200
```

Normal profile

Normal profile was obtained by the definition of `NormalWorker` with appropriate response status code.

```
50x NormalWorker=statusCode:200
```

7.1.3 PerfCake Scenario Definition

The following scenario was used as the basis for most test cases. And then experiments with the individual properties were done to obtain extreme cases. As we measure response time, the `HttpSender` component was used to handle requests and responses.

```xml
<?xml version="1.0" encoding="utf-8"?>
<scenario xmlns="urn:perfcake:scenario:8.0">
   <run type="${perfcake.run.type:time}" value="${perfcake.run.duration:120000}"/>
   <generator class="DefaultMessageGenerator"
       threads="${perfcake.thread.count:50}"/>
   <sender class="HttpSender">
      <target>http://${server.host}:${server.port}/post</target>
      <property name="method" value="POST"/>
   </sender>
   <reporting>
      <reporter class="ResponseTimeStatsReporter">
         <destination class="CrystalDestination">
            <period type="time" value="600"/>
            <property name="path" value="./Testing1/"/>
            <property name="title" value="Response time"/>
            <property name="order" value="1"/>
            <property name="threshold" value="1000"/>
            <property name="resultsAnalysis" value="true"/>
            <property name="basicStats" value="false"/>
            <property name="window" value="3000"/>
            <property name="chartYAxis" value="Response Time [ms]"/>
            <property name="chartGroup" value="response"/>
            <property name="chartAttributes" value="Result,Average"/>
         </destination>
      </reporter>
   </reporting>
   <messages>
      <message content="simple request message"/>
   </messages>
</scenario>
```

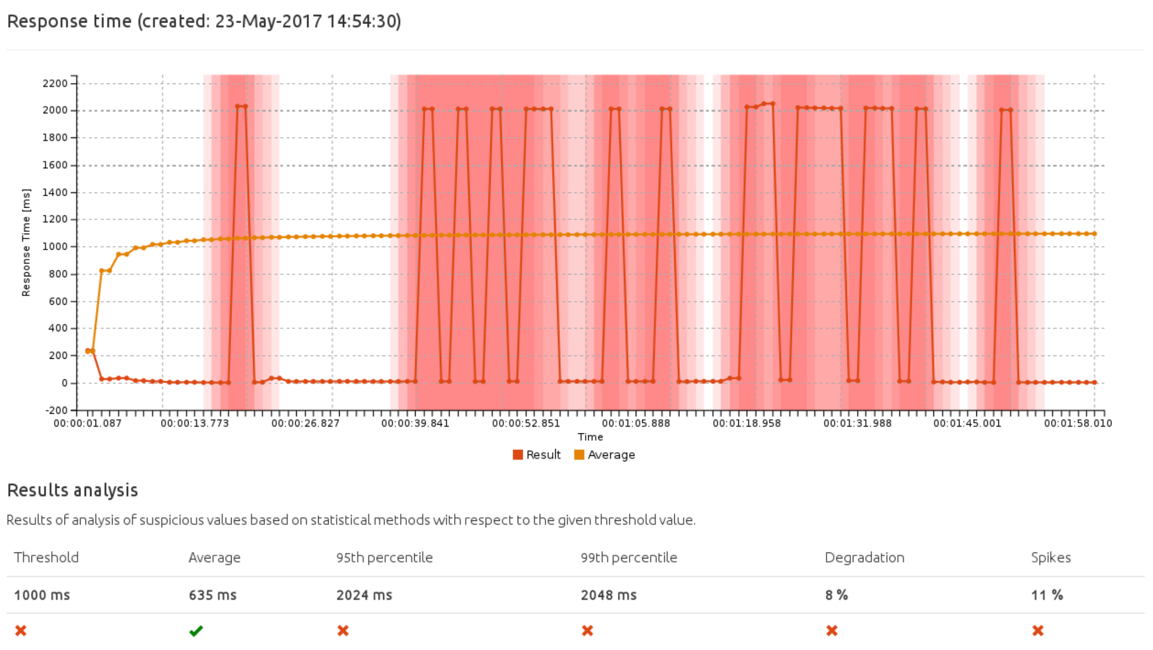
Listing 7.1: PerfCake scenario for response time testing

7.2 Achieved Results

The implemented solution for detecting performance issues is heuristic, so it is difficult to evaluate the results. There is no unified approach to what response time is acceptable as it always depends on the requirements of the developed system. Because we think that a Figure is worth a thousand words, several examples from experimenting with the developed component are provided with a brief description in the following section.

True Positive

In most cases, performance issues have been successfully detected.



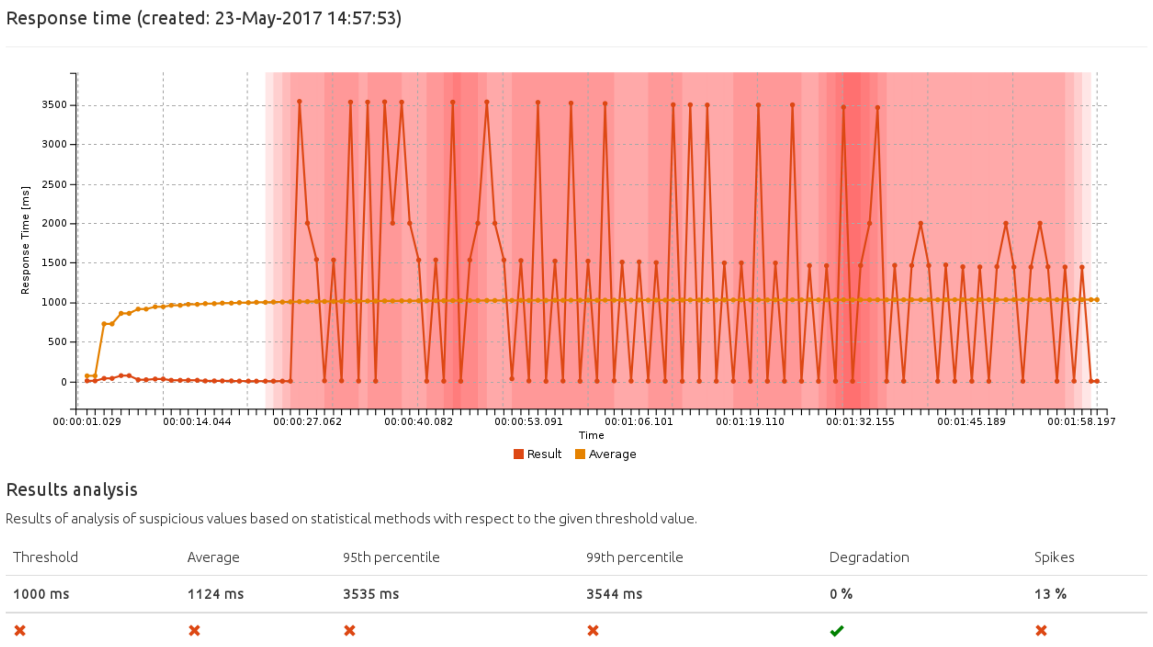
Response time (created: 23-May-2017 14:54:30)

Results analysis

Results of analysis of suspicious values based on statistical methods with respect to the given threshold value.

Threshold	Average	95th percentile	99th percentile	Degradation	Spikes
1000 ms	635 ms	2024 ms	2048 ms	8 %	11 %
✗	✓	✗	✗	✗	✗

Figure 7.1: Regular spikes in response time

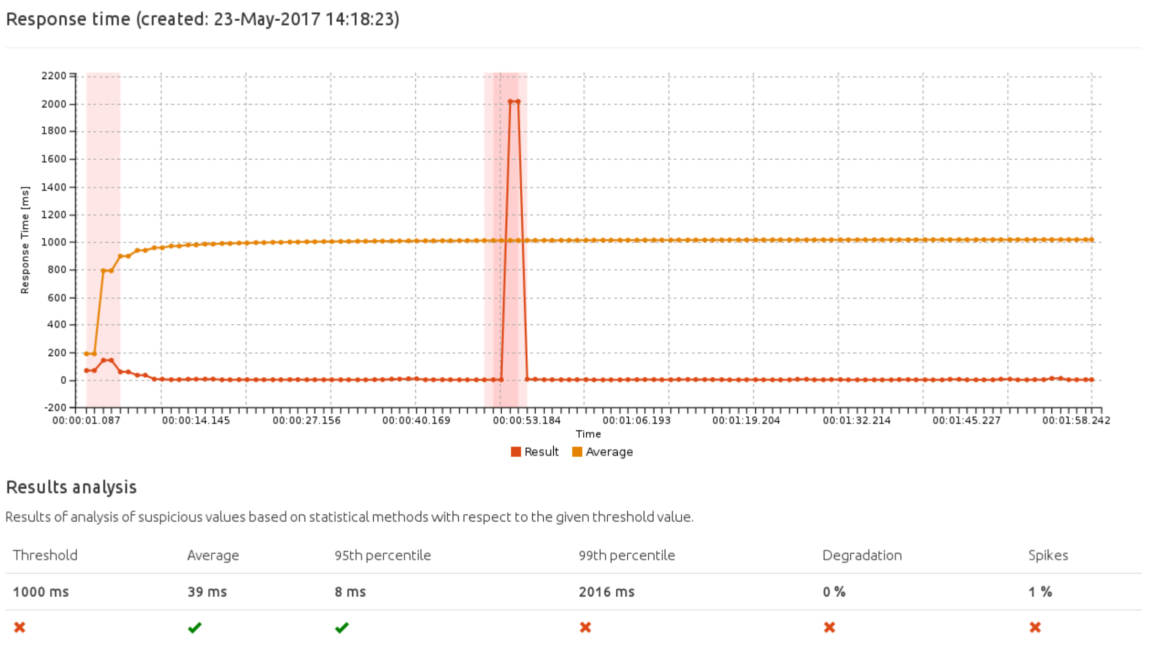


Results analysis

Results of analysis of suspicious values based on statistical methods with respect to the given threshold value.

Threshold	Average	95th percentile	99th percentile	Degradation	Spikes
1000 ms	1124 ms	3535 ms	3544 ms	0 %	13 %
✗	✗	✗	✗	✓	✗

Figure 7.2: Regular peaks and troughs in response time



Results analysis

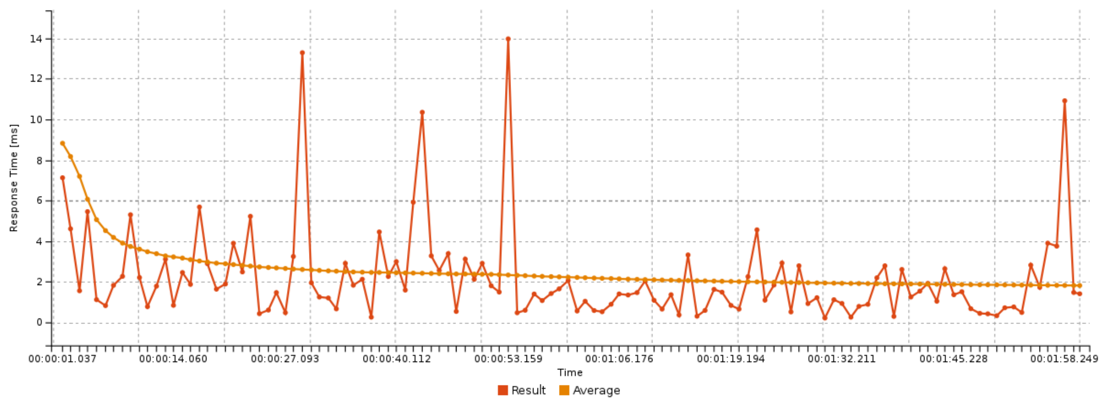
Results of analysis of suspicious values based on statistical methods with respect to the given threshold value.

Threshold	Average	95th percentile	99th percentile	Degradation	Spikes
1000 ms	39 ms	8 ms	2016 ms	0 %	1 %
✗	✓	✓	✗	✗	✗

Figure 7.3: Traffic spike in response time

True Negative

Response time (created: 23-May-2017 15:08:57)



Results analysis

Results of analysis of suspicious values based on statistical methods with respect to the given threshold value.

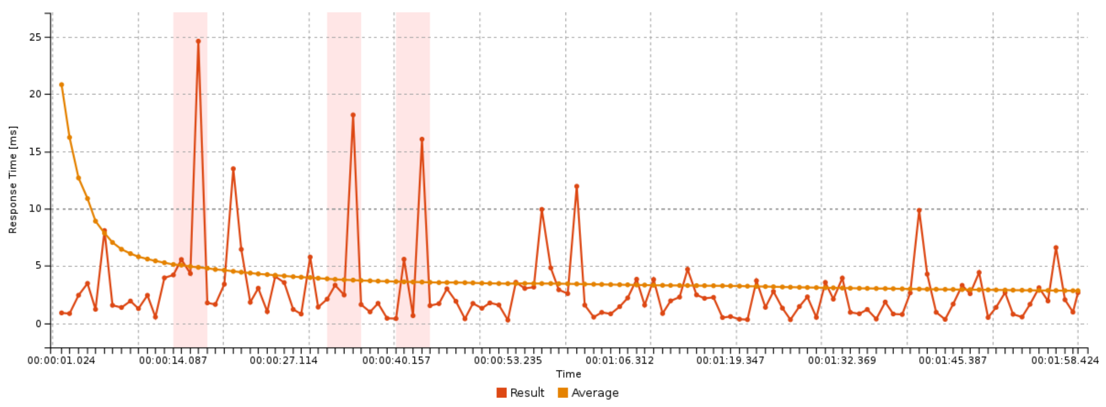
Threshold	Average	95th percentile	99th percentile	Degradation	Spikes
1000 ms	**2 ms**	**5 ms**	**13 ms**	**0 %**	**0 %**
✓	✓	✓	✓	✓	✓

Figure 7.4: Consistent response time

Special Cases

In some cases, it is up to the developer to decide, if the detected suspected profiles are signs of a rising performance problem or not.

Response time (created: 23-May-2017 15:11:34)



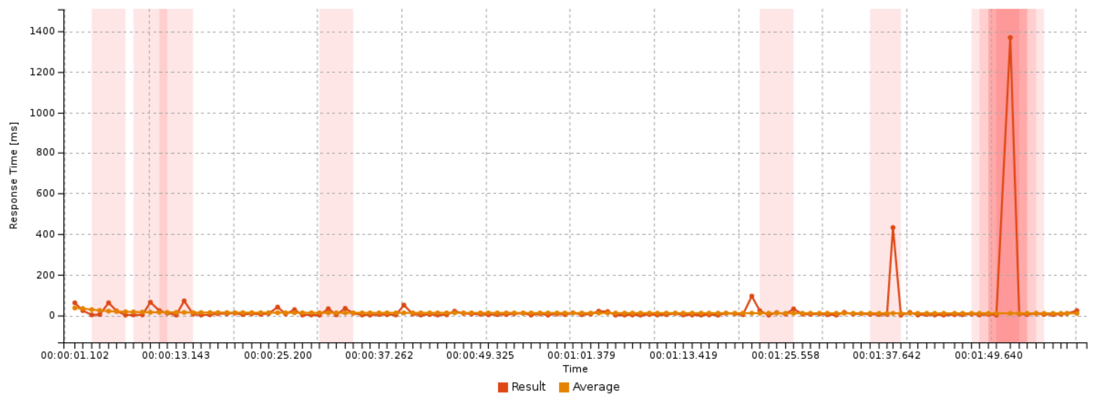
Results analysis

Results of analysis of suspicious values based on statistical methods with respect to the given threshold value.

Threshold	Average	95th percentile	99th percentile	Degradation	Spikes
1000 ms	**2 ms**	**10 ms**	**23 ms**	**0 %**	**2 %**
✓	✓	✓	✓	✓	✗

Figure 7.5: Special case required for the developer review

Response time (created: 23-May-2017 15:38:16)



Results analysis

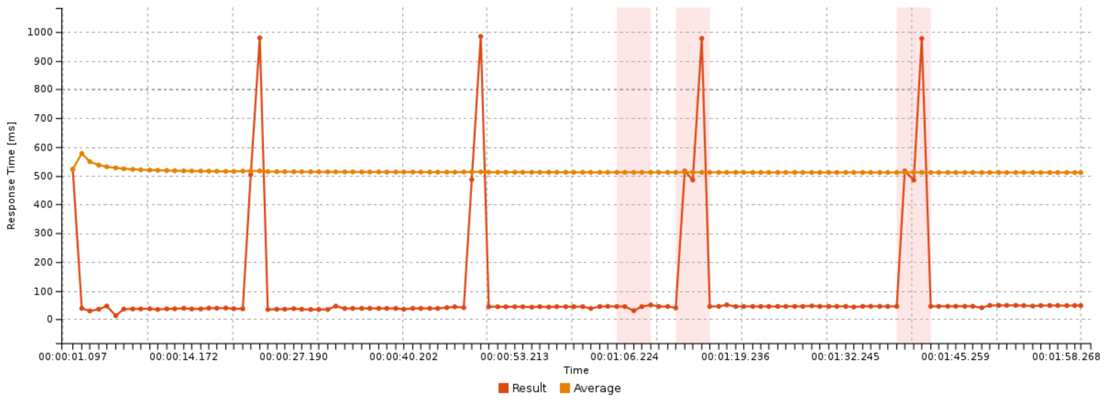
Results of analysis of suspicious values based on statistical methods with respect to the given threshold value.

Threshold	Average	95th percentile	99th percentile	Degradation	Spikes
1000 ms	**24 ms**	**45 ms**	**1283 ms**	**0 %**	**5 %**
✗	✓	✓	✗	✓	✗

Figure 7.6: Incorrectly identified spikes in response time

False Negative

Response time (created: 23-May-2017 16:40:46)



Results analysis

Results of analysis of suspicious values based on statistical methods with respect to the given threshold value.

Threshold	Average	95th percentile	99th percentile	Degradation	Spikes
1000 ms	**102 ms**	**516 ms**	**983 ms**	**0 %**	**2 %**
✓	✓	✓	✓	✓	✗

Figure 7.7: Incorrectly rejected spikes in response time

Chapter 8

Conclusion

Thesis conclusion

In this work, we proposed an approach to detect suspicious values in performance test results and thus identify a typical software performance issues. The approach is based on regression analysis that is one of the most commonly used statistical methods as they do not need to go through the training phase and they can analyze the results immediately. The proposed approach was implemented into the PerfCake testing tool and provides a feature to detect and report possible issues and their probability in a unified report. The report allows a fast interpretation of a performance test result and helps the developer evaluate the performance status of the system. This will speed up and simplify the analysis of performance test results. Our anomaly detection approach implementation has a heuristic solution, its success depends on the suitable definition of the parameter (window) – determining the size of the time window, in which the measured values are examined. The implemented algorithm is optimized for a window size of five records, since it has been found to be an ideal size from which we can distinguish a variety of profiles, both anomalous and normal. The implemented mechanism was successful in most of the tested cases, which have been performed in performance testing of the response time. There were a lot of true positive and true negative results. Fundamentals for testing and experimentation are included in the attached DVD.

Future work

The new component was developed in such a way that it can analyze the measured data directly and does not need a previous training phase to distinguish anomalies from normal behavior. However, it would be possible to implement the prediction of values according to the model of normal behavior (regression model trained in the training phase of normal behavior) and to compare the newly measured values with it. The model of normal behavior obtained during the training phase (before testing itself) could be entered as a parameter in the test scenario. Then, the regression model would be used to predict the y value for a new x value, which in our case represents time. This model can be obtained also with the help of our developed component, as the output report provides all the necessary parameters for its creation.

Regarding the use of more sophisticated methods, it may be interesting to apply methods based on data mining or machine learning. In the case of machine learning, there is no need to understand the nature of data-generating processes, because these methods are based on gradual learning and improving performance for a given task based on previous results.

This method would, therefore, be useful to use for long-term testing of software in terms of performance and thus to reveal other types of performance problems, especially the gradual degradation of various monitored metrics over time. In the case of data mining, methods based on classification are appropriate because they divide the input data into several classes based on a set of rules, patterns, or similar techniques. In the area of anomaly detection, it is usually a binary classification, i.e., division into only two classes - normal data and anomaly. Another option is cluster analysis. Cluster analysis is a generic name for cluster search methods in unknown (unexplained) multi-dimensional data. The main advantage of this method is its ability to learn from data and look for anomalies in it without having to provide a description of different profiles of these anomalies. The amount of training data that needs to be delivered to the system is also smaller than for other methods. The result of cluster analysis are definitions of the so-called *Outliers*, which are objects that do not belong to any cluster, and in the context of anomalies detection, they are likely to represent performance problems.

Bibliography

[1] Aung, K. M. M.; Oo, N. N.: *Association Rule Pattern Mining Approaches Network Anomaly Detection*. Singapore. 2015. ISBN 978-93-84468-20-0. 164 - 170 pp.

[2] bravenewgeek.com: *Benchmarking Message Queue Latency.* *http://bravenewgeek.com/tag/coordinated-omission/. February 2016.*

[3] Buch, D.: 4 types of load testing and when each should be used. http://www.radview.com/blog/4-types-of-load-testing-and-when-each-should-be-used. March 2015. [Online; visited 18.11.2016].

[4] Gaddam, S. R.; Phoha, V. V.; Balagani, K. S.: K-Means+ID3: A Novel Method for Supervised Anomaly Detection by Cascading K-Means Clustering and ID3 Decision Tree Learning Methods. IEEE Transactions on Knowledge and Data Engineering. *vol. 19. 2007: pp. 345 – 354. ISSN 1041-4347.*

[5] Gogoi, P.; Bhattacharyya, D. K.; Borah, B.; et al.: A Survey of Outlier Detection Methods in Network Anomaly Identification. The Computer Journal. February 2011.

[6] Lantz, B.: Machine Learning with R. Packt Publishing Ltd.. 2013. iSBN: 9781782162418.

[7] Lavrova, D.; Pechenkin, A.: Applying Correlation and Regression Analysis to Detect Security Incidents in the Internet of Things. International Journal of Communication Networks and Information Security (IJCNIS). December 2015.

[8] Meier, J. D.: Performance testing guidance for web applications: patterns & practices. United States: Microsoft. 2007. ISBN 0735625700.

[9] Montgomery, D. C.: Applied statistics and probability for engineers. New York: Wiley. 2003. ISBN 0-471-38181-0.

[10] Münz, G.; Li, S.; Carle, G.: Traffic Anomaly Detection Using KMeans Clustering. 2007.

[11] Oaks, S.: Java performance: the definitive guide. Sebastopol, CA: O'Reilly. 2014. ISBN 1449358454.

[12] Patcha, A.; Park, J.-M.: An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends. Comput. Netw.. *vol. 51, no. 12. August 2007: pp. 3448–3470. ISSN 1389-1286. doi:10.1016/j.comnet.2007.02.001.*

[13] *Večeřa, M.; Macík, P.: PerfCake 7.x Developers' Guide.* `https://www.perfcake.org/docs/developers-guide/perfcake-developers-guide.html`. *2016. [Online; visited 25.11.2016].*

[14] *Večeřa, M.; Macík, P.: PerfCake 7.x User Guide.* `https://www.perfcake.org/docs/user-guide/perfcake-user-guide.html`. *2016. [Online; visited 25.11.2016].*

[15] *Wang, Y.: Applying Correlation and Regression Analysis to Detect Security Incidents in the Internet of Things. Computers & Security. May 2005.*

Appendices

Appendix A

Content on DVD

The attached DVD contains:

- ***perfcake*** – source files, as can be obtained from GitHub repository[1],
- ***weaver*** – a service for simulating performance issues,
- ***examples*** – examples for testing the implemented solution,
- ***reports*** – reports presenting the results achieved,
- ***readme.txt*** – instructions for running the application,
- ***thesis.pdf*** – text of the thesis in PDF format,
- ***latex*** – LaTeX source files.

[1]https://github.com/KurovaMartina/PerfCake