FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Statistical Methods in Managing Web Performance

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Jürgen Cito
Matrikelnummer 0828112

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Schahram Dustdar
Mitwirkung: Mag. Dr. Philipp Leitner

Wien, 29.01.2014

(Unterschrift Verfasserin) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Statistical Methods in Managing Web Performance

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering & Internet Computing

by

Jürgen Cito
Registration Number 0828112

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Dr. Schahram Dustdar
Assistance: Mag. Dr. Philipp Leitner

Vienna, 29.01.2014

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Jürgen Cito
Linke Wienzeile 94/17, A-1060 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasserin)

Acknowledgements

First and foremost, I want to thank my parents, Lindi and Koco Cito. Words can't express the gratitude and respect I feel for the sacrifices you have made, and the support, guidance and love I have received throughout my whole life. **Faleminderit!**
I also want to thank my younger brother, Christian, for his support, and besides being family, also being one of my best friends. A special tribute goes out to my best friends, who have *always* been there. Thank you.

I would also like to express my sincere gratitude to my advisors, Philipp Leitner and Schahram Dustdar, for their great support and the opportunity to write my thesis at the Distributed Systems Group. Their expertise and valuable feedback were an important cornerstone in successfully finishing this thesis. I also want to thank Gernot Salzer for our whiteboard discussions on formal specification and notation.
A very special thanks to everyone at Catchpoint Systems in New York (where the research part of this thesis was conducted) for this exciting experience. Especially to my advisor, Drit Suljoti, for his patience and our endless discussions, and, Leo Vasiliou, for his expertise and guidance in the topic Web Performance Benchmarking.

Last, but not least, I want to thank my fellow students: Bernd, Gerald, Janos, Manuel, and Michael, that I have known since high school, for making the experience of working in university teams a very pleasant one.
In this regard, I want to also thank Thomas and Thomas, for the fruitful discussions and excursions we had - always a pleasure.

Abstract

Over the past few decades, the internet has grown to an enormous infrastructure connecting millions of people world-wide. The large scale of the internet has offered unique economic opportunities by enabling the ability to reach a tremendous, global user base for businesses and individuals alike. The great success and opportunities also open up overwhelming challenges due to the drastic growth and increasing complexity of the internet in the last decade. The main challenge for development and operations is to provide reliable and fast service, despite of fast growth in both traffic and frequency of requests. When it comes to speed, internet users have high demands on the performance of online services. Recent research has shown that 47% of online consumers expect load times of *two seconds or less*. This puts high pressure on web applications and websites in terms of performance and reliability, when even minor changes in performance can have significant effects on how users perceive an application and decide whether or not to continue with their interaction. If the revenue of a business is generated through their online presence, performance directly correlates with loss of sales, and damage of brand perception.

With the growth of the internet and its user base, the underlying infrastructure has drastically transformed from single server systems to heterogeneous, distributed systems. Thus, the end performance depends on diverse factors in different levels of server systems, networks and infrastructure, which makes providing a satisfying end-user experience and QoS a challenge for large scale internet applications.

In this thesis, statistical methods are applied to deal with the management of web performance issues. In order to properly manage performance, the timely observation and analysis of performance degradation is of key importance. Statistical changepoint analysis and exploratory visual analysis are applied to results obtained through synthetic monitoring of a simulation system to identify and discuss how performance issues in web applications can be detected.

Web Performance Benchmarking of competitors is discussed as a way to establish a baseline for an overall performance goal. A general methodology for web performance benchmarking is developed based on statistical analysis of finite collections of observed data from production systems. The presented approach is then applied in a case study for benchmarking search engine providers.

Kurzfassung

In den letzten Jahrezenten ist das Internet zu einer ernormen infrastruktur angewachsen, die Millionen von Menschen auf der Welt vernetzt. Die weite Verbreitung des Internets hat einzigartige wirtschaftliche Möglichkeiten eröffnet, welche jedoch auch mit üeberwältigenden Herausforderungen durch das drastische Wachstum und der immer steigenden Komplexität, vor allem in den letzten Jahren, verbunden sind. Die Herausforderung für Development und Operations ist trotz dieses Wachstums ein zuverlässiges und performantes Service bereitzustellen. Wenn es um Performance geht, haben Internetnutzer hoch Ansprüche an Online Services - aktuelle Forschung zeigt, dass Ladezeiten von *zwei Sekunden oder weniger* erwartet werden. Diese Erwartungen erhöhen den Druck auf Webapplikationen und Websites, da auch kleinste Änderungen in der Performance die Wahrnehmung einer Applikation drastisch beeinflussen. Wenn der Umsatz eines Unternehmens durch dessen Onlinepräsenz generiert wird, so korreliert mangelnde Performance direkt mit schlechten Verkaufszahlen und Markenwahrnemung.
Durch das zuvor erwähnte Wachstum hat sich die zugrunde liegende Infrastruktur des Internets von Single Server Systemen zu heterogenen, verteilten Systemen entwickelt. Die resultierende Performance des Gesamtsystems hängt somit von diversen Faktoren in verschiedenen Ebenen der Serversysteme und Netzwerke ab, welche die Bereitstellung einer zufriedenstellenden Servicequalität eine Herausforderung in Large Scale Internet Applikationen macht.
In dieser Arbeit werden statistische Methoden vorgestellt, die sich mit dem Management von Web Performance Problemen auseinandersetzen. Die ordentliche Verwaltung von Performance hängt mit der zeitnahen Überwachung und Analyse der zugrunde liegenden Systeme zusammen. Statistische Changepoint Analyse und explorative, visuelle Verfahren werden auf Performance Daten angewandt, die durch synthethisches Monitoring innerhalb einer Simulation bezogen werden. Es wird gezeigt und diskutiert wie Performance Probleme durch diese Methoden erkannt werden können.
Web Performance Benchmarking von Wettbewerbern wird im zweiten Teil der Arbeit als Methode diskutiert um ein Performance Ziel zu definieren. Es wird eine allgemeine Methode zu Web Performance Benchmarking vorgestellt, welche auf statistische Analyse von über Zeit hinweg gemessenen Daten von Produktionssystemen basiert. Das präsentierte Vorgehen wird schließlich in einer Case Study angewandt, bei der Performance von Suchmaschinenanbietern verglichen wird.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Contribution . . . 4
- 1.3 Organization . . . 4

2 Background **7**
- 2.1 Web Performance Monitoring . . . 7
  - 2.1.1 Availabilty, Performance and Reliability . . . 7
  - 2.1.2 Anatomy of a Web Transaction . . . 9
  - 2.1.3 Performance Monitoring . . . 12
  - 2.1.4 Web Performance Metrics . . . 15
- 2.2 Statistical Methods . . . 19
  - 2.2.1 Measures of Central Tendency . . . 19
  - 2.2.2 Measures of Dispersion . . . 20
  - 2.2.3 Exploratory Data Analysis . . . 21
  - 2.2.4 Time Series Analysis . . . 23
  - 2.2.5 Changepoint Analysis . . . 26

3 Related Work **29**
- 3.1 Statistical Approaches . . . 29
  - 3.1.1 Statistical Process Control . . . 30
  - 3.1.2 Anomaly Detection . . . 31
- 3.2 Web Performance Benchmarking . . . 33
  - 3.2.1 Load Testing Benchmarks . . . 33
  - 3.2.2 Operational System Benchmarks . . . 34

4 Observing Changes in Web Performance **35**
- 4.1 Taxonomy of Root-Causes in Server Performance Regression . . . 36
  - 4.1.1 General Cause of Performance Delays . . . 36
  - 4.1.2 Global Delay . . . 37
  - 4.1.3 Periodic Delay . . . 38
  - 4.1.4 Partial Delay . . . 39
- 4.2 Simulation Design . . . 39

4.2.1 Synthetic Agent Nodes . . . 40
4.2.2 Scenario Generation Component . . . 41
4.2.3 Dynamic Web Service Component . . . 43
4.3 Experimental Setup . . . 44
4.3.1 Synthetic Agent Node Setup . . . 44
4.3.2 Web Server Setup . . . 44
4.4 Methodology . . . 45
4.4.1 Simulation Scenarios . . . 45
4.4.2 Data Collection . . . 45
4.4.3 Exploratory Data Analysis . . . 46
4.4.4 Statistical Analysis . . . 46
4.4.5 Threats to Validity . . . 48
4.5 Simulation Scenarios . . . 48
4.5.1 Global Delay . . . 48
4.5.2 Partial Delay . . . 49
4.5.3 Periodic Delay . . . 51
4.6 Results and Interpretation . . . 53
4.6.1 Global Delay . . . 53
4.6.2 Partial Delay . . . 56
4.6.3 Periodic Delay . . . 60

5 Web Performance Benchmarking **63**
5.1 Methodology . . . 64
5.2 Case Study . . . 68
5.2.1 Goal and Area . . . 68
5.2.2 Test subjects . . . 68
5.2.3 Parameters . . . 68
5.2.4 Metrics . . . 68
5.2.5 Data Collection and Analysis . . . 69
5.2.6 Results . . . 69

6 Conclusion **73**
6.1 Summary . . . 73
6.2 Research Questions Revisited . . . 73
6.3 Future Work . . . 74

A Acronyms **75**

B List of Synthetic Agent Locations **77**

Bibliography **79**

List of Figures

1.1 Average transfer size and number of requests over time [39] 2
1.2 Infrastructure of a large scale web service [72] 4

2.1 Anatomy of a simple HTTP transaction 9
2.2 Anatomy of a persistent HTTP transaction 11
2.3 Anatomy of a parallel HTTP transaction 11
2.4 Different types of monitoring 12
2.5 Browser simulation through simple HTTP GET illustrated on the example of retrieving www.test.com [30] 14
2.6 Browser puppetry through emulating a real browser illustrated on the example of retrieving www.test.com and performing actions on the DOM [30] 15
2.7 Example plot of a time series together with its moving average with window size *10* in red 25

3.1 Control chart with 2 measurements beyond its limits - plotted with R package 'qcc' [67] 30
3.2 Outline of automated performance regression detection approach in [59] 31
3.3 Example of anomalies in 2-dimensional data [21] 31
3.4 Original application signature compared to the application signature of the new software release [24] 32

4.1 Taxonomy of Root-Causes in Server Performance Regression 37
4.2 Simulation model 40
4.3 Depiction of the periodic development scheme defined in Equation 4.5 43
4.4 Results of Global Delay – non-filtered time series plot 54
4.5 Results of Global Delay – red vertical line indicates changepoint in variance 55
4.6 Results of Global Delay – red horizontal lines indicate the mean values for each detected different distribution 55
4.7 Partial delay scenario results time series plot 57
4.8 Partial delay scenario results time series plot, together with 90th percentile 57
4.9 Partial delay scenario results depicted as a scatterplot 58
4.10 Results of the partial delay scenario – red vertical lines indicate changepoints in variance 59

4.11 Results of partial delay scenario – red horizontal lines indicate the mean values for each detected different distribution 59
4.12 Periodic Delay scenario results as a time series plot 60
4.13 Results of the periodic delay scenario - red lines indicate changepoints in the variance 62
4.14 Results of the periodic delay scenario - red lines indicate changepoints in the mean 62

5.1 A methodology for web performance benchmarking 64
5.2 Suggested graph for a follow up process - Goal is set at 600ms for 85% of users . . 67
5.3 Response times ranking 70
5.4 Render start time ranking 70
5.5 Webpage response ranking 70

List of Tables

1.1 Time correlating with user perception [39] 2

4.1 Example of informal description of simulation parameters in tabular form 44
4.2 Synthetic node setup 44
4.3 Web server setup 45
4.4 Informal description of the global delay simulation scenario $\mathcal{S}_G$ 50
4.5 Informal description of partial delay simulation scenario $\mathcal{S}_P$ 51
4.6 Informal description of periodic delay simulation scenario $\mathcal{S}_{PD}$ 52
4.7 Changepoint in the variance for global delay simulation scenario $\mathcal{S}_G$ 54
4.8 Changepoint in the mean for global delay simulation scenario $\mathcal{S}_G$ 56
4.9 Changepoint in the variance for partial delay simulation scenario $\mathcal{S}_P$ 58
4.10 Changepoint in the mean for partial delay simulation scenario $\mathcal{S}_P$ 59
4.11 Changepoint in the variance for periodic delay simulation scenario $\mathcal{S}_{PD}$ 61
4.12 Changepoints in the mean for the periodic delay simulation scenario $\mathcal{S}_{PD}$ 61

5.1 Results of median response time, render start time, and webpage response time . . 71

CHAPTER 1

Introduction

"Motivation is the art of getting people to do what you want them to do because they want to do it."

— Dwight D. Eisenhower,
34^{th} President of the United States

The internet started out as a small-scale research network for scientists to share centralized computing resources across different geographic locations. Over the past few decades, the internet has grown to an enormous infrastructure connecting millions of people world-wide. Its usage spans over a variety of different areas ranging from communication and collaboration to entertainment and commerce. The large scale of the internet has offered unique economic opportunities by enabling the ability to reach a tremendous, global user base for businesses and individuals alike. The great success and opportunities also open up overwhelming challenges, that will be addressed in the following. The internet is growing at a rapid pace; both average transfer size and number of requests have been growing drastically in the last decade, especially in recent years (as can be seen in Figure 1.1). The main challenge for development and operations is to provide reliable and fast service, despite of fast growth in both traffic and frequency of requests.

Psychology of Performance

When it comes to speed, internet users have high demands on the performance of online services. Early research in HCI shows that frustration increases with overall load times that exceed *8-10 seconds* [17, 48]. More recent research has shown that faster internet connections and overall improvements in web performance has made internet users even less patient over time: 47% of online consumers expect load times of *two seconds or less* [2].
While time can be accurately measured in units, performance is based on human perception that

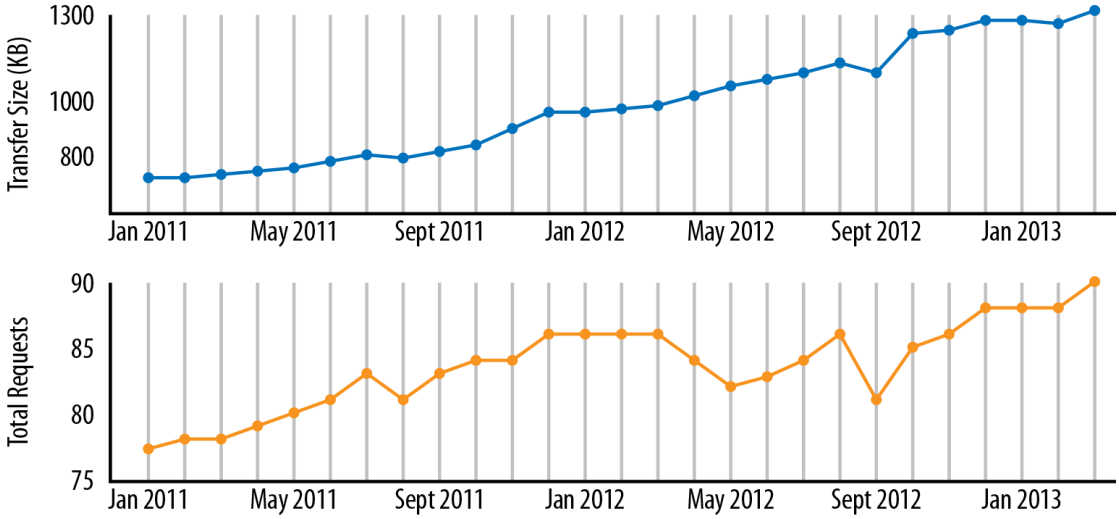

Figure 1.1: Average transfer size and number of requests over time [39]

depends on context and expectations. Table 1.1 shows how performance delays measured in units of time correlate with user perception [39].

Delay	User perception
0-100ms	Instant
100-300ms	Small perceptible delay
300-1000ms	Machine is working
1000+ ms	Likely mental context switch
10000+ ms	Task is abandoned

Table 1.1: Time correlating with user perception [39]

These user-centric perceptions on time put high pressure on web applications and websites in terms of performance and reliability. Even minor changes in performance can have significant effects on how users perceive an application and decide whether or not to continue with their interaction.

1.1 Motivation

From the perception internet users have on time spent on web applications and websites follow drastic implications for e-commerce and other businesses operating over the internet. The advantages that have enabled economic opportunities, such as a large scale, globalized user base, also facilitate the emergence of more competition. The simple and effortless possibility of switching to a competitive service through a mouse-click makes it all more important to avoid user frustration and abandonment of the service.

If the revenue of a business is generated through their online presence, performance directly correlates with loss of sales and revenue:

- If Amazon increased page load time by +100ms they lose 1% of sales [50]
- Google Maps reduced its initial page size by 25%, traffic went up 10% in the first week, and an additional 25% in the following three weeks [33]
- 500ms of extra load time at Google decreased traffic and ad revenues by 20% [76]
- 400ms of extra load time caused 5-9% drop in full-page traffic, i.e., visitors leaving before the site was fully loaded at Yahoo! [76]

While these statistics show direct correlation from performance to sales and revenue, [5] suggests that the quality of experience on the internet reflect the company's brand. Hence, poor performance or unavailability can damage the overall brand perception and image of a company and impact loyalty.

With the growth of the internet and its user base, the underlying infrastructure has drastically grown from single server systems to heterogeneous distributed systems. Modern large scale web services run on loosely coupled, complex systems, spanning multiple data centers and content distribution networks (as illustrated in Figure 1.2). The end performance depends on diverse factors in different levels of server systems, networks and infrastructure. This makes providing a satisfying end-user experience and Quality of Service (QoS) a challenge for large scale internet applications and websites. Analysis and diagnosis of underlying causes of web performance issues is a difficult and complicated endeavor: problems that cause performance bottlenecks can be manifested in any parts and layers in the ecosystem of the application.
Performance monitoring provides a means of quality assessment and improvement. Monitoring is the process of continually observing the state of a system and measuring its quality attributes (e.g., performance). Web performance monitoring primarily aims at early detection of unavailability and performance degradation, in order to avoid or minimize loss of revenue and brand damage. Furthermore, the collected measurements over time enable more complex, statistical analysis. This makes active performance monitoring an essential tool in maintaining a satisfying end-user experience and QoS, as well as learning from past observations through enabling analysis on collected data.

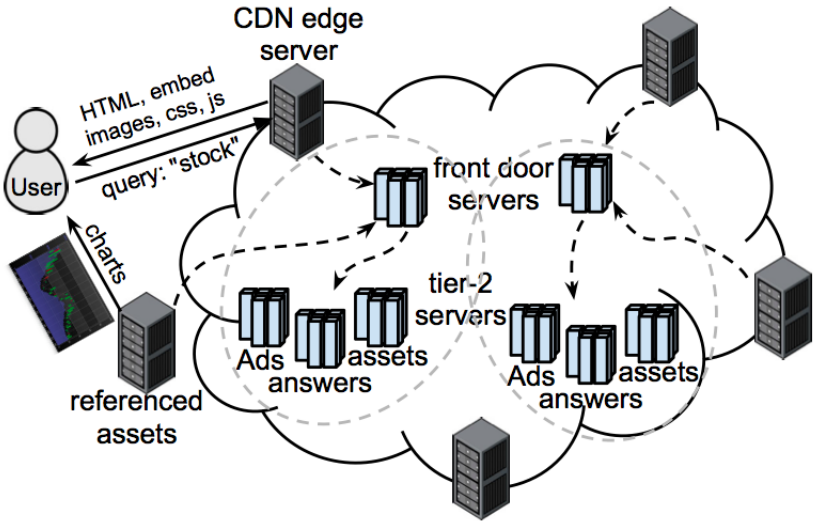

Figure 1.2: Infrastructure of a large scale web service [72]

1.2 Contribution

In this thesis, we apply statistical analysis and methods to deal with the management of web performance issues. It primarily aims to fill the gap between web performance monitoring and the field of statistics. The following research questions are addressed:

1. How do performance issues manifest in performance data gathered through active monitoring? How can these changes in performance be observed, both visually and through statistical methods and algorithms?

2. How do we determine a performance goal? When a certain performance goal is determined, how can the progress of achieving this goal be actively tracked and communicated?

The performance data for this thesis is collected by the means of active performance monitoring through synthetic agents, either within a data center or geographically distributed throughout the world.

1.3 Organization

The topics of the thesis will be organized as follows:

- Chapter 2 provides an overview of the background, covering the basics of web performance analysis. Further, it will give an introduction to basic statistical methods that are used, as well as an introduction to time series analysis and changepoint analysis.

- Chapter 3 will present related research work, which is closely related to statistical methods in web performance.

- The first research question will be examined in Chapter 4. It will present a simulation design covering scenarios that will simulate root-causes in web performance degradation. Furthermore, it will present and discuss the results of the defined simulations.
- Chapter 5 deals with the second research question, and will suggest a general methodology for web performance benchmarking based on statistical analysis of finite collections of observed data from production systems. It also suggests a process that visually assists the pursuit of a performance goal by a development team.
- Concluding remarks and an outlook to possible future work will be found in Chapter 6.

CHAPTER 2

Background

In order to gain thorough understanding of this thesis' topic, this chapter will work through the background that provides all basics of web performance analysis and statistics based on literature study of state-of-the-art publications, as well as current research on the topics.

2.1 Web Performance Monitoring

"If you can't measure it,
you can't improve it."

— Lord Kelvin,
18^{th} Century Physicist

Web performance monitoring refers to the process of observing the state of web services or web applications over time by employing either external or internal monitoring or measuring artifacts. The aim of a monitoring process is the identification and detection of existing and potential issues in the monitored system. Web performance monitoring activities are heavily coupled with measurement activities of a certain set of performance metrics. In order to fully understand the web performance monitoring process, the following sections aim on establishing an understanding on how certain elements within the monitoring process work.

2.1.1 Availabilty, Performance and Reliability

When it comes to the state of a web service or web application, we distinguish between the QoS attributes *availability*, *performance*, and *reliability*.

Availability

Availability is defined as the property that the system is ready to be used at any given point in time [77]. Colloquially, the availability of a web service is referred to as *up time*. Conversely,

the unavailability is refereed to as *down time*. Consequently, a web service is referred to as either 'up' or 'down', depending on whether it is available (i.e., ready to be used at the given time) or not. Availability is used as a QoS attribute in contractual agreements (especially Service Level Agreements (SLA)), and is usually given in percent. It is the relative relationship between *up time* and *down time* as defined in [62]:

$$Availability = \frac{\text{System up time}}{\text{System up time + System down time}} \tag{2.1}$$

When it comes to (external) monitoring and measurement of web services and web applications, availability is not a trivial matter. The state of availability (or, for that matter, unavailability) cannot always be queried[1]. The question is, when is a service seen as unavailable? Usually, in web performance monitors, a threshold is defined that a certain performance metric must not exceed in order to be still seen as available. This threshold is known as a *connection timeout*.

Performance

Performance refers to how fast a system delivers its service with regard to a certain performance metric. Performance is measured in the unit that follows from the collected metric through the measurement system. Section Refsubsec:metrics lists and describes prevalent metrics that are used in web performance monitoring. The presence of a performance measurement at a specific point in time, already infers availability.

Reliability

Reliability is about continuous availability of a system without failure. It is defined in terms of a time interval, as opposed to availability, that is defined in terms of a point in time [77].

[1] Sometimes the HTTP status does return a proper status code to indicate service unavailability, but this is not always the case

2.1.2 Anatomy of a Web Transaction

Communication in distributed systems happens through a series of protocols on different layers. In web services and web applications, communication between client and servers happens through the HTTP protocol, which has a significant impact on performance. When dealing with web performance monitoring, it is important to learn the fundamentals of the protocol. Understanding how HTTP works and what each component in an HTTP transaction means is key to interpreting the data collected by any (external) monitoring service. In the following, we will describe the basics of a web transaction over the HTTP protocol. This section is largely based on [20, 34, 38, 58].
HTTP is an application layer protocol built on top of the Transfer Control Protocol (TCP) protocol. It guarantees reliability of delivery, and breaks down larger data requests and responses into chunks that can be delivered over the network. TCP is a "connection" oriented protocol, which means, whenever a client starts a dialogue with a server, the TCP protocol will open a connection, over which the HTTP data will be reliably transferred. When the dialogue is complete that connection should be closed.

Simple HTTP transaction

A simple HTTP transaction is one where the client (either a web browser or another application in case of a web service) makes a single request for HTTP content to the *web server* hosting the web service or web application. Figure 2.1 depicts the workflow of such a simple HTTP transaction. In the following, we describe the steps taken in this workflow from starting the request to receiving the response.

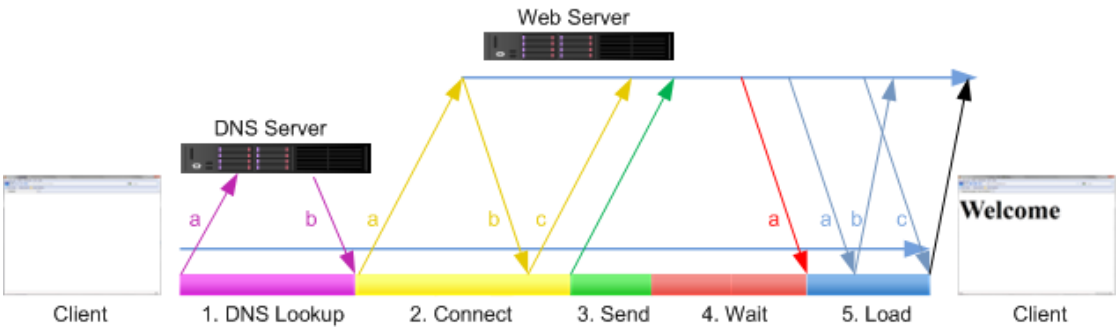

Figure 2.1: Anatomy of a simple HTTP transaction

1. DNS Lookup

The Domain Name System (DNS) lookup resolves a domain name for the request. The client sends a DNS query to the local Internet Service Provider (ISP) DNS server[2]. The DNS server

[2]The local ISP's DNS Server is usually configured on the client, but can be configured to different DNS server as well

responds with the IP address for the requested host name.

2. Connect

Establishes a TCP connection with the IP address of the webserver (which has been retrieved in the previous step). The client sends a SYN packet to server. The server sends a SYN-ACK packet back to the client. The client responds with an ACK packet, which establishes the TCP three-way handshake.

3. Send

The actual HTTP request is sent to the web server.

4. Wait

The client waits for the server to respond to the request. The web server processes the HTTP request, generates a response, which is sent back from the server to the client. The client receives the first byte of the first packet from the server (containing the HTTP Response headers and content)

5. Load

The client loads the content of the response. The web server sends the second TCP segment with a PSH flag. The PSH flag informs the server that the data should be pushed up to the receiving application immediately. The client sends an ACK message (for every two TCP segments it receives) and the web server sends third TCP segment with an HTTP_Continue status code (which denotes that the client may continue with its request).

6. Close

The client sends a FIN packet to close the TCP connection.

Persistent HTTP Transactions

Persistent connections allow the client to utilize the same connection for different object requests to the same host. They are supported natively in the HTTP 1.1 protocol. In HTTP 1.0, persistent connections are controlled via the Keep-Alive HTTP header.

Figure 2.2 depicts the workflow of a persistent HTTP transaction. As opposed to the single request, in the persistent transaction the subsequent workflow of $SEND \rightarrow WAIT \rightarrow LOAD$ is repeated for each different object resource that is loaded from the same host. Afterwards the connection is closed. The remainder of the protocol follows the simple HTTP transaction, as discussed before.

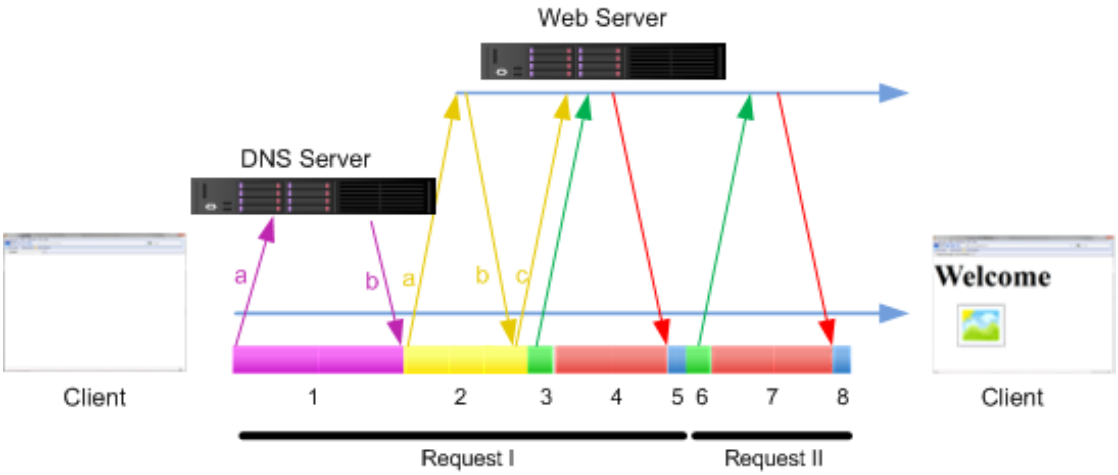

Figure 2.2: Anatomy of a persistent HTTP transaction

Parallel HTTP Transactions

The HTTP 1.1 protocol provides clients with the ability to open multiple connections and perform HTTP requests in parallel.

In Figure 2.3, the browser loads a resource from *hostname.com* which contains two image requests to *hostname.com* and two requests to *adserver.com*. Once the first request is completed, the subsequent requests are performed in parallel, which results in performance improvements.

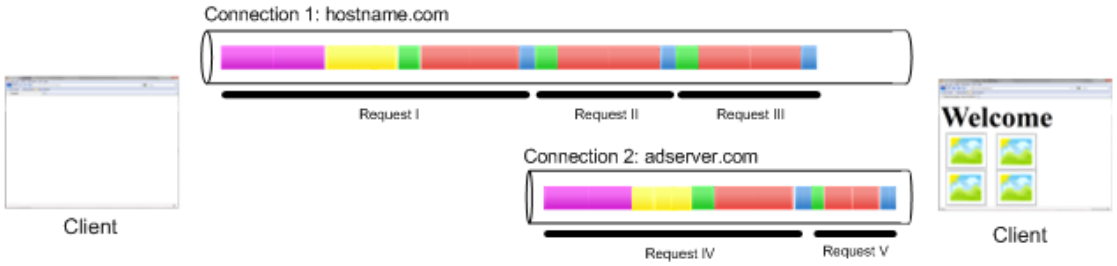

Figure 2.3: Anatomy of a parallel HTTP transaction

2.1.3 Performance Monitoring

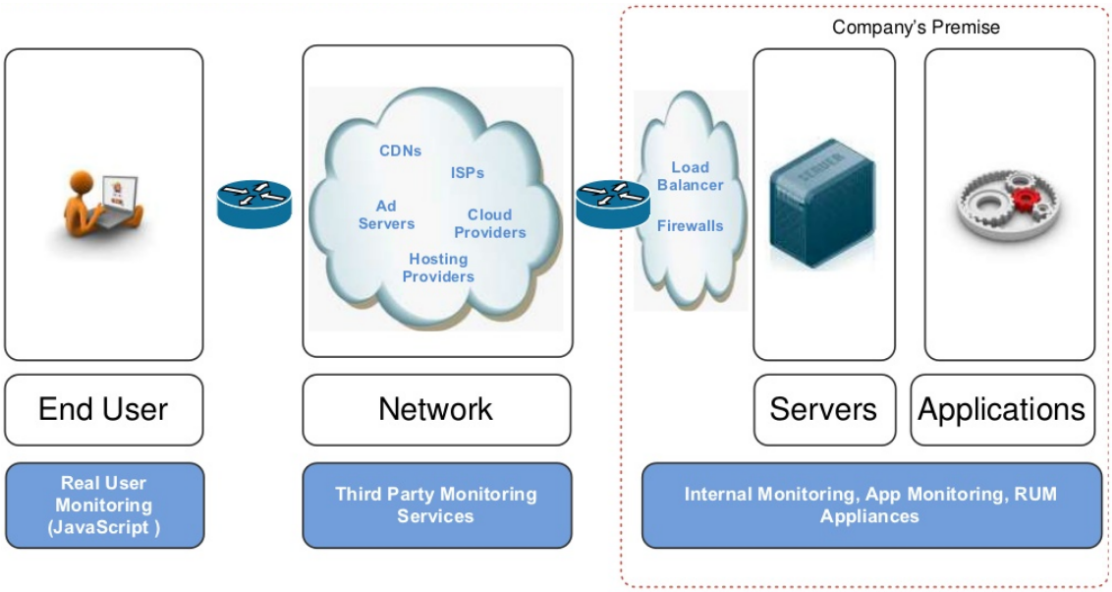

Figure 2.4: Different types of monitoring

Internal Monitoring

Internal monitoring, sometimes also referred to as traditional Application Performance and Resource Management (APM), allows the monitoring of health, availability and performance of applications and underlying infrastructure from within the organization's own datacenter. Internal monitoring for web services and web applications is comprised of tools and processes, which are known from "classical" monitoring in distributed systems, such as

- Systems monitoring (CPU, I/O, Memory, Load, etc.)
- Network monitoring (Bandwidth, Throughput, etc.)
- Log analysis
- Middleware monitoring

Internal monitoring is referred to as a passive monitoring system. This means that monitoring data is gathered only through actual interactions with the system itself, i.e., the monitoring system does not trigger any action to actively obtain performance measurements. Internal monitoring provides deep insight and visibility into the application's performance, as well as complete understanding of internal infrastructure performance and resource utilization.
The downside of internal monitoring is the lack of insight and visibility on the performance experienced by end users. The overall user experience in terms of performance does not only depend

on the internal applications and infrastructure, but also on external networks, third-parties, and frontend performance. These are aspects that internal monitoring fails to capture.

Synthetic Web Performance Monitoring

Due to the complexity of modern distributed systems, the infrastructure powering web services and web application - the *internet* - can encounter issues in many different ways, in the many layers and components of systems. Some of these issues may reside within applications and datacenters, and can thus be detected through internal monitoring. Other problems are of external nature: they reside beyond controlled environment, but also impact performance that is perceived by the end-user.

Synthetic Monitoring simulates end-user experience by actively measuring performance metrics through geographically distributed software-based agents. It is called synthetic because it does not involve web traffic generated by real clients and end-users. In literature synthetic monitoring is also referred to as *active monitoring* or *active probing* [25], because the agents are actively trying to connect to the target system in order to collect performance data, whether the target system is accessed by real end-users or not. Probing is usually performed in uniformly distributed time intervals. The software-based agents are usually deployed in data centers throughout the world, in order to obtain an objective view on latency experienced in different geographic regions. Due to the emergence of mobile devices [55], the deployment of synthetic agents is not limited to data centers that are connected through broadband internet-connectivity. Vendors have expanded deployment to nodes that are connected through 3G or 4G connectivity from carriers in the respective areas of probing. Thus, the measurements obtained through synthetic monitoring are representations of performance metrics by simulated clients that produce real world results. This allows us to know the state of the target system at any given time as perceived by end-users. This approach yields the advantage of knowing and acting on the information of either unavailability or performance degradation as soon as these events occur, and preferably before it is affecting any end-users.

Depending on the target system that needs to be monitored within the ecosystem of a web application, different tools are used in the monitoring process. The simplest way to test basic (network) connectivity is the method *ping* (also known as ICMP ECHO), which sends a single packet to a webserver and receives a response. However, the capabilities to measure proper performance metrics with *ping* are very limited. Synthetic monitoring agents use HTTP GET to simulate user interaction with the target systems in order to verify functionality and measure the timing of various milestones in request and response (see Section 2.1.2). There are several possibilities to leverage the use of HTTP GET, from only requesting a single resource of an application (e.g., only the *products* page of an e-commerce site) to measuring the performance metrics of a whole transaction or process (e.g., *checkout process* of an e-commerce site), which involves many steps that need to be scripted.

Client Simulation

When looking at different synthetic monitoring solutions there is an important distinction to be made in the way an end-user client is simulated. The software-based agent can either simulate a browser (*browser simulation*) or run a real browser (*browser puppetry*) [30]. The difference between these approaches is crucial and will be briefly described in the following.

In **browser simulation** a synthetic agents emulates the client through a series of HTTP commands combined into a testing script. The responses are parsed and tested for the desired outcome (e.g., HTTP status, containing certain contents, specific assertions, etc.). Figure 2.5 shows the approach used in browser simulation. While this method is very easy to implement, it has limitations that *browser puppetry*, that will be explained in the following, does not have.

Instead of simply setting up a script simulating HTTP requests, in **browser puppetry**, we run tests by *"manipulating an actual copy of a web browser"* [30]. The script is now more advanced and operates on the Document Object Model (DOM) of the returned HTML resource. This allows for more complex and real world testing, as many modern web sites and applications work with JavaScript, which otherwise would not be executed. Figure 2.6 displays the approach used in browser puppetry.

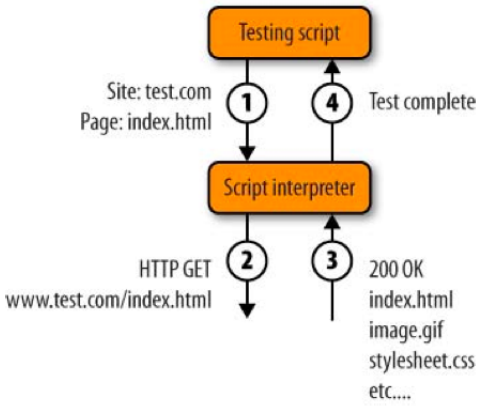

Figure 2.5: Browser simulation through simple HTTP GET illustrated on the example of retrieving www.test.com [30]

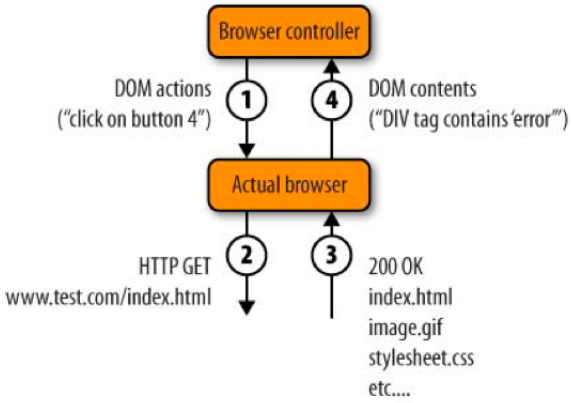

Figure 2.6: Browser puppetry through emulating a real browser illustrated on the example of retrieving www.test.com and performing actions on the DOM [30]

Real-User Monitoring (RUM)

Real-User Monitoring (RUM) tracks web page performance from the browser, measuring the performance as experienced by the end users. Thus, with RUM, we obtain data on real users when they access the application. It gathers the user experience across all active users, geographies, and devices, thus allowing not only for (near) real-time metrics, but also further statistics on usage to correlate performance degradation. It is a form of passive monitoring, meaning that it only measures if the system is actively used by its end users. Thus, if end users cannot access the application, or there is a low/no traffic period, or not enough data captured.

Implementation

Real-user monitoring is usually implemented through a small JavaScript tag that imports an external script to the page. This allows either an external service handling the monitoring or an internal monitoring appliance to measure metrics from the moment a request was initiated to the final loading of the whole page. The code in the snippet activates a ping-back of the measured data to the desired monitoring server where the performance data is finally stored. The specific performance metrics that are measured through RUM are described in Section 2.1.4. Additionally, RUM also captures device information, operating system, browser information and geographic origin.

2.1.4 Web Performance Metrics

The Oxford English Dictionary (OED) defines the term "metrics" as *"a set of figures and statistics that measure results"* [60]. In [40], time is defined as *"the basis of all computer performance management"* that all other metrics in performance measurement and analysis are build upon. In the previous sections, we established the basic means of communication on the internet, as well as the ways we are able to measure and monitor our internal and external systems by the means

of agents. Web performance metrics are the results we obtain by these measurement that yield a specific meaning to the QoS and Quality of Experience (QoE) of the end users. In the following, we introduce the main metrics employed in web performance analysis by synthetic agents and real user monitoring. We divide the metrics into *request component metrics* and *frontend metrics*.

In order for the synthetic agent to measure the metrics, it requires the location of the web application or web service to be examined. This will be referred to as the 'target URL' in the description of the metrics. If the target URL has a **redirect chain**, the specific metric should be calculated as the sum of the time it took to for each step to be completed, i.e., for each domain in the redirect chain. For DNS, it is the time it took to *resolve* the DNS for each URL in the redirect chain.

Request Component Metrics

Request component metrics examine the communication involved in retrieving the target URL. The metrics are derived from the transport components in HTTP (see Section 2.1.2). The following metrics can only be measured through synthetic node agents.

DNS

Time it took the agent to resolve the DNS for the target URL.

Connect

Time it took to establish a connection with the server for the target URL.

Send

Time it took to send the request to the server for the target URL.

Wait

Time from when the connection was established successfully and the request was sent to the server, to the time we receive the first byte of response for the target URL.

SSL

Time it took to establish the SSL handshake with the target URL.

Time to First Byte (TTFB)

Time it took from the request being issued to receiving the first byte of data from the target URL. It is calculated as follows

$$TTFB = DNS + Connect + Send + Wait \tag{2.2}$$

Load

Time it took between loading the first byte to the last byte of target URL. It is also referred to as *Receive Time*.

Response

Time it took from the request being issued to the primary host server responding with the last byte of the target URL.

Server Response

Time it took from when the DNS was resolved to the server responding with the last byte of the target URL. This shows the server's response exclusive of DNS times. For tests where the primary URL has a redirect chain, the 'Server Response' metric is calculated as the response time for each domain in the redirect chain minus the DNS time.

File Size

File size in bytes of the response received for the target URL.

Throughput

The numerical representation of how efficiently the system was able to retrieve the target URL in KB/s. The throughput is calculated as follows

$$Throughput = \frac{FileSize}{Wait + Load} \tag{2.3}$$

Frontend Metrics

The metrics on the frontend additionally examine events that occur after the contents of the target URL have been loaded, including additional elements that are referenced by the markup. These type of metrics are only possible to measure if the synthetic agent employs browser puppetry (see Section 2.1.3) and only target websites and applications that deliver markup (HTML).

DOM Load Time

Time it took to load the DOM on the page of the target URL.

Content Load

Time it took to load the entire content of the website after the connection was established with the server for the target URL. This is the time elapsed between the end of Send until the final element, or object, on the page is loaded. Content Load does not include the DNS, Connect, Send, and SSL time on the target URL (or any redirects of the target URL).

Render Start

Time it took the browser to start rendering the page.

Document Complete

Time it took for the JavaScript 'onLoad' event to fire.

Time to Title

Time it took the browser to display the title of the page.

Webpage Response

Time it took to load every element of the webpage. This starts from the initial request and ends at the last received byte of the final request on the page (including images, scripts, stylesheets, third party services, etc.).

2.2 Statistical Methods

"There are three kinds of lies:
lies, damned lies, and statistics."

— Benjamin Disraeli,
British prime minister and author

In this chapter, we discuss basic statistical methods for exploratory data analysis of numeric attributes that are crucial in the analysis and management of web performance. We will focus on the univariate analysis of measures of center or location, as well as of measures of dispersion. Furthermore, we will briefly introduce time series analysis and methods of change detection in time series. The following descriptions of basic statistical notions is largely based on [42, 68].

2.2.1 Measures of Central Tendency

Measures of central tendency or location of a distribution of data are single values that attempt to describe what we deem to be typical or central within the set of data. The most common measures are the statistics *arithmetic mean* (usually referred to as "average"), the *median*, and the *mode*. On occasion, other statistics for mean values such as the *geomtric mean* or *harmonic mean* are used for measures of center when appropriate. When referring to the "average" mostly the arithmetic mean is meant.

The aforementioned statistics are all legitimate measures of central tendencies, but under certain circumstances, some measures are more appropriate than others. The following sections will briefly outline the center statistics that are important and widely used in the world of web performance data: *arithmetic mean* and *median*.

Arithmetic Mean

The arithmetic mean is probably the most popular and well known statistic for the measure of central tendency, and it is often colloquially referred to as the "average". The mean can be applied to both discrete and continuous data, although it is most often applied to the latter.
In mathematical terms, we assume that we are looking at a data set having n data points labeled x_1, x_2, ... through x_n. The data set can either be a statistical population (i.e., the entire collection of data points that could be possibly observed) or a statistical sample (a subset of the statistical population in question). Depending from what set the mean has to be calculated, it is either specified as the population mean or the sample mean. It is mostly not feasible to observe the entire statistical population, so most of the times the sample mean is calculated.
The sample mean $\bar{x}$ is defined in Equation 2.4.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{2.4}$$

The arithmetic mean is simply defined as the sum of all data values in the data set divided by the number of values. It can be essentially thought of as the value that is most common or representative within the data. In the context of measurements of web performance metrics (of any kind), the mean captures an approximate view of what all users included within the measurement process have experienced.

Median

The median is another measure of central tendency that is less affected by skewness of data and outliers within the data set. Again we differentiate between population median and sample median. Given the order statistics [8] defined in Equation 2.5, we define the sample median $\tilde{x}$ in Equation 2.6.

$$Y_1 = min_j X_j, Y_2, ..., Y_{n-1}, Y_N = max_j X_j \tag{2.5}$$

$$\tilde{x} = \begin{cases} Y_{(n+1)/2} & \text{if } N \text{ is odd} \\ \frac{1}{2}(Y_{\frac{N}{2}} + Y_{1+\frac{N}{2}}) & \text{if } N \text{ is even} \end{cases} \tag{2.6}$$

$\tilde{x}$ is the middle value after all of the values are put in an ordered list. This means that at least half of the values are less than or equal to the median and at least half of the values are greater than or equal to the median. If the number of values in the list is even, the median is the arithmetic mean of the two middle values.

For symmetric distributions, the median is equal to the mean. For skewed distributions (also called asymmetric distributions), the median is preferred as a measure of central tendency. The preference in these cases is given due to the median's robustness. A sample statistic has the property of robustness if the value of the property is not affected by outliers in the data or other rather small deviations from model assumptions. The median is a highly robust statistical property, because you can move almost all of the upper or lower half of the data values any distance away from the median without changing its value. From a more practical point of view, a few observation points that are significantly above or below the median usually have no effect on its value.

2.2.2 Measures of Dispersion

Measures of dispersion express the properties of variation and spread or dispersion in the values of a random variable. There are also referred to as the measure of spread of a distribution. Higher diversification in the examined data is expressed through in higher measures of statistical dispersion. Literature in the area performance measurements and statistics has been limited with a focus on the (arithmetic) mean of performance metrics, such as the response time [9].

However, as the performance experienced by the end user is highly affected by the variability in response times and other metrics, measures of dispersion of performance metrics are crucial in understanding end user experience. The most common measures of dispersion found in basic literature on descriptive statistics are the *variance* and *standard deviation*. The following sections will briefly introduce these statistics.

Variance

The variance is a standard measure of dispersion (and similarly to measures of central tendency) is calculated from a data set having n data points labeled x_1, x_2, ... through x_n, which can either be a statistical population or a statistical sample. It is the average squared distance between the arithmetic mean and each item in the data set (Equation 2.7).

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{(n-1)} \tag{2.7}$$

The most commonly used symbol for the sample variance is s^2; the population variance is commonly σ^2. When calculating the sample variance $n-1$ is used as the denominator instead of simply n in order to reduce the bias that was introduced through the already used statistic $\bar{x}$ in the calculation.

Standard deviation

The standard deviation is simply the positive square root of the variance. Its advantage compared to the variance is that it expresses spread in the same units as the originally observed values in the data set, making the statistic more understandable.

2.2.3 Exploratory Data Analysis

Exploring and analyzing data that are produced through monitoring of large scale internet applications is becoming increasingly difficult, due to nature and the high volumes of data that is observed. In exploratory data analysis, we analyze data sets in order to obtain a summary, quick view and grasp of certain main characteristics of the data. This can either be done through summary statistics or through visual data exploration, without using a specific statistical model. Grasping interrelationships between multiple performance metrics and time within complex layers of various systems is a hard undertaking. It is difficult to analyze and derive conclusions of this abstract collection of (multidimensional) performance data using traditional techniques that do not employ visual representations of information. A potential solution is the use of visualization techniques to convert high volume of data into an image that domain experts can use for exploration, identification of patterns, and further diagnostics.

Data Visualization

Data Visualization presents information in form of illustrations that aim to improve the understanding of data, or sometimes even establish understanding for underlying systems and processes. Presentation of data in visual form allows humans to apply their perceptual abilities to get insight into the data, derive a mental model in presence of domain knowledge and draw conclusions.

In [65] the visualization of information has been formalized:

- A data set D contains n data elements, e_i, such that $D = \{e_1, ..., e_n\}$
- A data set A representing a set of data attributes, a_j, such that $A = \{A_1, ...A_m\}$ where $m \geq 1$[3]
- Data elements encode values for each attribute: $e_i = \{a_{i,1}, ...a_{i,m}\}, a_{i,j} \in A_j$.
- A function $M(V, \Phi)$ maps raw data to visual information, where
  - $V = \{V_1, ..., V_m\}$ is a set of m visual features with V_j selected to represent each attribute A_j, and
  - $\Phi_j : A_j \mapsto V_j$ maps the domain of A_j to the range of displayable values in V_j

This means that visualization is the process of selecting an appropriate transformation function M, creating images that enable proper exploration and analysis of the underlying processes or data.

[3] $m = 1$ denotes univariate data, $m > 1$ denotes multivariate data

2.2.4 Time Series Analysis

Time series analysis comprises methods and techniques for analyzing data and extracting meaningful characteristics, which is largely employed throughout the sciences and engineering in a variety of fields. This section on this prevalent topic is based mostly on [18, 57, 71].

A time series represents an ordered sequence of observations collected over time. The data points are usually recorded at uniform temporal intervals. Data points recorded in an irregular fashion are typically interpolated to form values at regular intervals before the time series is analyzed. A data point may consist of a single observation (univariate time series) or multiple observations (multivariate time series). More formally, we can represent a time series X as a discrete function with value x_i for every equally spaced point in time t_i ,where $t_i \in T$ is a time variable in a set of n points (Equation 2.8).

$$X = \{x_1, x_2, ..., x_n\} \tag{2.8}$$

Stochastic Processes

A process is a mathematical description of an underlying system and can be defined as a sequence of random variables. In this context, a system can be either deterministic or stochastic. A process is said to be deterministic if its future states are completely determined by the present state of the system. If there are different possibilities for transitions to other states, which are not completely determined by the present state, the system is stochastic.

Stochastic processes are statistical models for time series. A stochastic process $\{Y(t), t \in T\}$ maps events of the process to real numbers, where T is an index set for time points. T in this context is a set of times where the process is observed (i.e., sampled). This means that, at any time $t \in T$, $Y(t)$ is a random variable. The connection between time series and stochastic processes, can be explained as follows: A time series X, $t = 1, ..., T$ can be seen as a realization (or the observations) from a stochastic process $\{Y(t), t \in T\}$.

In web performance, the underlying system driving the stochastic process is our web application, which is clearly not deterministic.

Stationarity

Stationarity is a common assumption in many time series techniques. This means that many time series models only yield meaningful results if the underlying process delivers stationary data. However, many time series observed in practice (including time series in web performance measurements) are non-stationary. Therefore, non-stationary time series should be transformed to stationary time series before applying any further analysis. We distinguish between *strict stationarity* and *weak stationarity*. In the literature, the terms stationarity and weak stationarity are used synonymously, unless specified otherwise. In practice, strict stationarity is too limiting of an assumption, and rarely holds true. In the following, we will provide a brief definition on

weak stationarity.

A stationary process is a process where the statistical properties of mean, variance and autocovariance do not change over time. In mathematical terms, we denote Equation 2.9 to be the mean of Y (where E is the function calculating the expected value) and Equation 2.10 to be the autocovariance of Y.

$$\mu(t) \;=\; E(Y(t)) \tag{2.9}$$

$$\gamma(s,t) \;=\; Cov(Y(s), Y(t)) \tag{2.10}$$

If $\mu(t)$ is the same for each $t \in T$ and $\gamma(s,t)$ only depends on the distance $|s-t|$ and not on the point in time itself, then we say that Y has (weak) stationarity. In conclusion, a time series

is said to be stationary if there is

- No systematic change in the mean (i.e., no trend)
- If there is no systematic change in variance
- If strictly periodic variations have been removed (i.e., no seasonality)

There are different ways of obtaining a stationary time series from non-stationary processes. For stochastic processes, this is done through differencing [71]. A more detailed explanation on trend and seasonality can be found in [18].

The following sections briefly discuss additional ways and particular methods of dealing with time series derived from time correlated stochastic processes.

Smoothing Techniques

Smoothing refers to a technique that attempts to reduce random fluctuations in time series data in order to capture important behaviour of the underlying process (signal) and leave out data that might impact the overall judgment of the process (noise). In the context of web performance data in this thesis, smoothing is primarily used to filter out short-term spikes (outliers) that might be caused by network phenomena and that are not associated with a fundamental shift in the underlying process. Smoothing techniques are also used to reveal the underlying trend, seasonal components and cyclic components of time series data. A very common way of smoothing is by building the 'moving average'. There are several other methods, such as exponential smoothing, which are further explained in [36].

Moving Average

A moving average is a method for smoothing time series by creating a new series containing the averages of fixed-sized subsets of the original series. More formally, given a time series $X = \{x_1, ...x_n\}$, a moving average with a window of w elements is a new time series $S = \{s_1, ...s_{n-w+1}\}$ defined by taking the arithmetic mean of subsequences of w data points (Equation 2.11).

$$s_i = \frac{\sum_{j=1}^{i+w-1} x_j}{w} \tag{2.11}$$

Figure 2.7 illustrates an example of a possible time series plot, displaying both the values of the time series containing the observed values (black line), as well as the newly constructed sequence with a moving average window size $w = 10$ (red line).

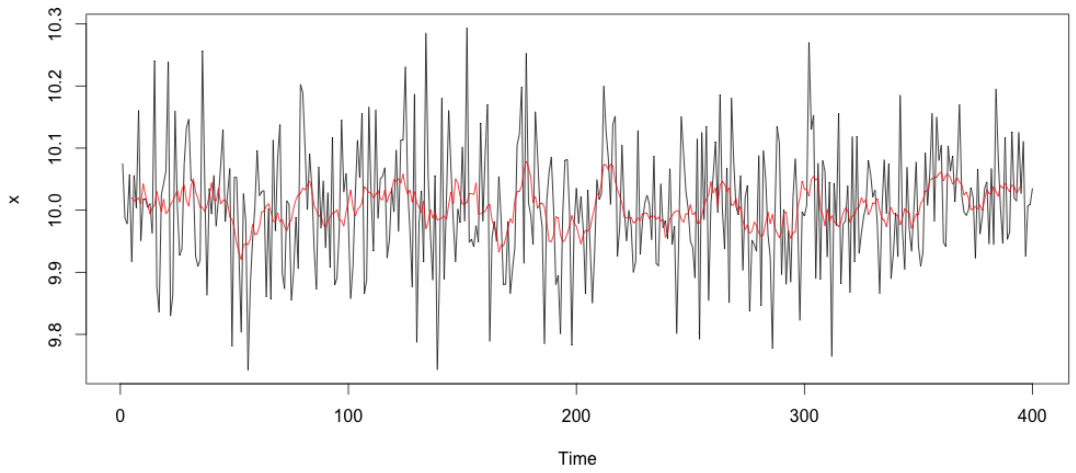

Figure 2.7: Example plot of a time series together with its moving average with window size *10* in red

In conclusion, a moving average is commonly used to smooth out short-term fluctuations and emphasize long-term cycles or trends. However, when working with moving average, or any smoothed data sets for that matter, we must be aware of the fact that this data has been newly created from our observations and thus contains its own error.

2.2.5 Changepoint Analysis

Changepoint analysis deals with the identification of points within a time series where the statistical properties change. For the context of observing changes in web performance data in particular, we are looking for a fundamental shift in the underlying Probability density function (pdf). In a time series, we assume that the observations come from one specific distribution initially, but at some point in time, this distribution may change. The aim of changepoint analysis is to determine if such changes in distribution have occurred, as well as the estimation of the points in time when these changes have taken place. In [46], the basic problem of changepoint detection has been formalized as follows:

- Let us assume we have a sequence of observations $y_{1:n} = (y_1, ..., y_n)$.
- A changepoint is said to occur within this set when there exists a time, $\tau \in \{1, ..., n-1\}$, such that the statistical properties of $\{y_1, ..., y_\tau\}$ and $\{y_{\tau+1}, ..., y_n\}$ exhibit differences.
- If we extend the notion from single changepoints to multiple changepoints, we say we have a number of points in time m in the set $\tau_{1:m} = \{\tau_1, ..., \tau_m\}$ where $\tau_i \in \tau$.
- We assume that the changepoints are an ordered set, such that $\tau_i < \tau_j$ iff $i < j$.
- Thus, our m changepoints will split our initial data set $y_{1:n}$ into $m+1$ segments, with the i^{th} segment containing the range $y_{(\tau_{i-1}+1):\tau_i}$.
- Now, each data segment will be summarized by a set of parameters, where the i^{th} segment will be denoted $\{\theta_i, \phi_i\}$ where
  - ϕ_i is a set of nuisance parameters that are not of immediate interest but must be accounted for in the analysis
  - θ_i is a set of parameters that may contain the (distribution) we are looking for
- Next step is to estimate the values of the parameters of each segment in order to know how many changepoints are present in the data

Now we've introduced the notion of changepoints within an ordered sequence of data. The detection of these points in time generally takes the form of hypothesis testing. The null hypothesis, H_0, represents no changepoint (i.e., $m = 0$) and the alternative hypothesis, H_1, represents changepoints exist (i.e., $m > 0$).
If we go from a general point of view (i.e., testing for a statistical property) to the more specific property we are looking for, namely changes in the distribution, we can formulate H_0 as in Equation 2.12.

$$H_0 : F_1 = F_2 = ... = F_n \tag{2.12}$$

where $F_1, F_2, ..., F_n$ are pdf associated with the random variables of our segments. The alternative hypothesis would then be defined as in Equation 2.13.

$$H_1 : F_1 = ... = F_{k_1} \neq F_{k_1+1} = ... = F_{k_2} \neq F_{k_2+1} = ... = F_{k_m} \neq F_{k_m+1} = ... = F_n \quad (2.13)$$

where $1 < k_1 < k_2 < ... < k_m < n$. m is, as before, the number of changepoints. $k_1, k_2, ...k_m$ are the unknown positions of changepoints $k_i \in \tau$. As we constructed our parameter θ for each segment before, we only need to test the changes in the parameters. Then the hypothesis testing results to

$$H_0 : \theta_1 = \theta_2 = ... = \theta_n \quad (2.14)$$

versus the alternative hypothesis

$$H_1 : \theta_1 = ... = \theta_{k_1} \neq \theta_{k_1+1} = ... = \theta_{k_2} \neq \theta_{k_2+1} = ... = \theta_{k_m} \neq \theta_{k_m+1} = ... = \theta_n \quad (2.15)$$

with the same meaning for k_i as before and with θ_i being the parameter to be tested.

In order to test this hypothesis, a test statistic needs to be constructed which can decide whether a change has occurred. A general likelihood-ratio based approach can be used. For further information we refer to the original papers [22, 46], as well as to work providing a more comprehensive review [74].

CHAPTER 3

Related Work

This chapter gives an overview over existing research work employing other approaches in statistical methods in a performance context, and performance benchmarking.

3.1 Statistical Approaches

This section discusses research work that employs statistical methods for performance analysis. Analyzing performance data observed through continuous monitoring in distributed systems and web services has produced a whole body of research dealing with statistical modeling of underlying systems, anomaly detection and other traditional methods in statistics to address problems in performance monitoring and analysis.

Pinpoint [23] collects end-to-end traces of requests in large, dynamic systems and performs statistical analysis over a large number of requests to identify components that are likely to fail within the system. It uses a hierarchical statistical clustering method, using the arithmetic mean to calculate the difference between components by employing the Jaccard similarity coefficient [43], which is used to compare the diversity and similarity of sample sets.
[27] makes use of statistical modeling to derive signatures of systems state in order to enable identification and quantification of recurrent performance problems through automated clustering and similarity-based comparisons of the signatures.
[3] proposes an approach which identifies root causes of latency in communication paths for distributed systems using statistical techniques.
[26] proposes that the statistics geometric mean, geometric standard deviation and geometric confidence intervals are superior to their arithmetic counterparts in web performance analysis. Using empirical data they show that response times often follow a lognormal distribution [4].
An analysis framework to observe and differentiate systemic and anomalous variations in response times through time series analysis was developed in [72].

A whole body of research at WUT[1] highlights the use of statistical methods in web performance monitoring and analysis [10–16]. The research work spans from statistical approaches to predict web performance, design of architectures for multiagent internet measurement systems for monitoring, to empirical studies to assess web quality by employing statistical performance models.

3.1.1 Statistical Process Control

Statistical Process Control (SPC) is a technique using statistical methods to monitor and control the quality of a certain process [70]. SPC sets control limits that are within three standard deviations from the mean. A measurement beyond these limits indicates that the process has shifted or has become unstable. Figure 3.1 shows a control chart that has two measurements that are beyond the limits. SPC was originally created for quality control in manufacturing processes, but has been used for quality assurance in various fields of business, science and engineering [56].

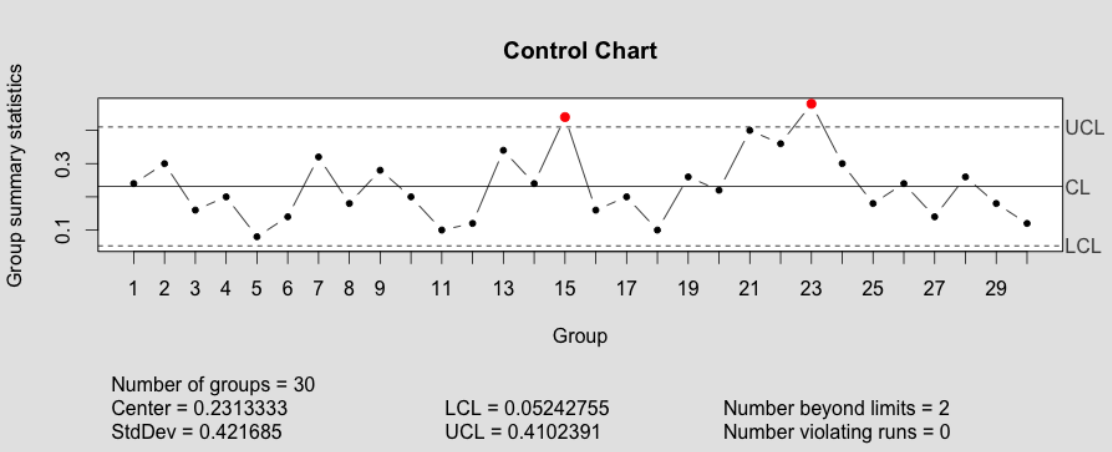

Figure 3.1: Control chart with 2 measurements beyond its limits - plotted with R package 'qcc' [67]

[59] suggests an approach to automated detection of performance regressions using control charts that is outlined in Figure 3.2. The lower and upper control limits for the control charts are determined through load testing that establish a baseline for performance testing. The preprocessing steps 'scale' and 'filter' in the approach are applied to satisfy the assumptions of a non-varying process and normality of the measured data for SPC. In these steps, the baselines are scaled employing a linear regression model to minimize the effect of load differences. Finally, a violation ratio is determined as an indicator of performance regression in the new software release.

In [44] the basic idea of SPC is used to develop an improvement model to analyze end-user experience in terms of performance, and make data-driven decisions in order to improve overall

[1] Wrocalaw University of Technology, Poland

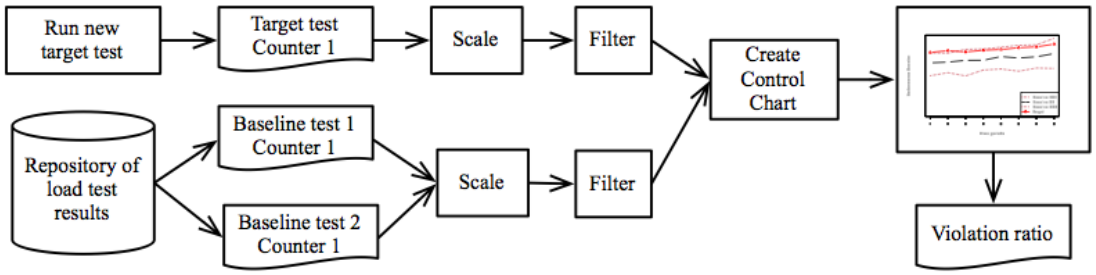

Figure 3.2: Outline of automated performance regression detection approach in [59]

QoS.
The open source project Skyline [73] is a near real time anomaly detection system for various metrics, which uses the idea of control charts as its basic algorithm. It is described in more detail in Section 3.1.2.

3.1.2 Anomaly Detection

Anomalies are defined as patterns in data that do not comply to expected normal behavior. Anomaly detection can, subsequently, be seen as the definition of regions in data representing normal behavior and any observation that does not adhere to this notion of normality is seen as an anomaly [21]. Figure 3.3 illustrates the idea of anomalies. The data sets N_1 and N_2 are considered the regions representing normal behavior, as most observations are grouped in these regions. The points o_1 and o_2, as well as the regions O_3 are considered to be anomalous.

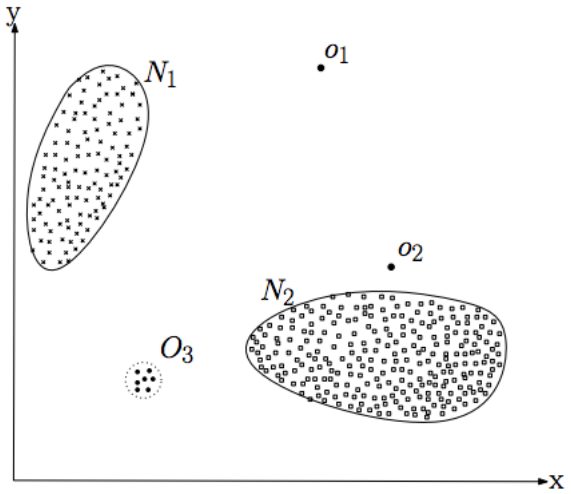

Figure 3.3: Example of anomalies in 2-dimensional data [21]

Anomaly detection in a performance context is a widely used approach in current research work. [19] applies statistical methods to detect anomalies in network communication. Specifically, statistical characterization of TCP traffic was used and several stochastic models have been compared.

[24] proposes an automated anomaly detection and performance modeling approach in large scale enterprise applications. A framework is proposed that integrates a regression based transaction model reflecting resource consumption and an application performance signature that provides a model of runtime behavior. The constructed application signature serves as a metric to uniquely reflect application transactions and their CPU requirements. After software releases, the application signatures are compared in order to detect changes in performance. Figure 3.4 illustrates a change in performance through comparison of signatures before and after new software release.

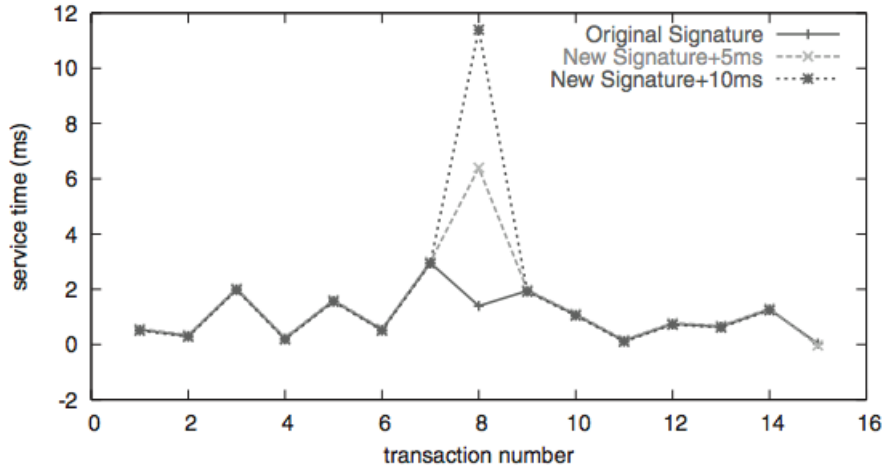

Figure 3.4: Original application signature compared to the application signature of the new software release [24]

As briefly discussed in Section 3.1.1, Skyline [73] is an open source tool developed by Etsy[2] that aims on providing anomaly detection for various performance metrics in web applications. As it is designed to deal with a variety of different metrics, non-parametric algorithms are employed for anomaly detection that do not need prior statistical modeling of the data. In order to minimize the amount of false positives, Skyline applies many different algorithms, and only classifies a metric to be anomalous if the majority of algorithms agree with the classification. The amount of algorithms which are deemed to be the majority can be configured in the project.

[2]http://www.etsy.com

3.2 Web Performance Benchmarking

Performance benchmarking in computing is a widely used concept to rate and compare the efficiency and performance of the targeted systems. Benchmarking in the context of web applications and web services is a process used to compare the performance of hardware and software systems under specific metrics and measurement procedures. We differentiate between two different kinds of benchmarks:

- Benchmarks that are used to evaluate the performance of a system under a well-defined synthetic workload specification.
- Benchmarks that evaluate performance of systems already in production.

3.2.1 Load Testing Benchmarks

The aim of performance benchmarks for load testing is usually capacity planning, testing of scalability, and proper sizing of systems that are not yet in production. The System Under Test (SUT) is evaluated under a well-defined workload specification. This means the type of requests and frequency of submission of requests is predefined and simulated. There are a few organizations offering benchmark specifications for web systems, which will be briefly coverd in the following.

Standard Performance Evaluation Corporation (SPEC) is an organization that develops standardized performance tests, such as SPECweb [28], which is designed to measure a system's ability to handle a maximum number of simultaneous connections, while still meeting specific throughput and error rate requirements. Workload characteristics are drawn from statistics analyzed from web server logs.
Transaction Processing Performance Council (TPC) is a nonprofit organization that defines transaction processing and database benchmarks. The TPC-W benchmark aims at evaluating the performance of webservers for e-business and e-commerce applications [54]. The activities that characterize the workload are driven by emulated browsers, which generate specific web interactions.
[78] gives a comprehensive overview on benchmarking techniques used by both academic researchers and practitioners for hardware and software systems.
[6] reviews benchmarking models, tools and techniques that are used to evaluate performance and scalability of distributed web server systems. Furthermore, it describes common possibilities and pitfalls for data collection and analysis of the benchmark results.

3.2.2 Operational System Benchmarks

When it comes to benchmarks that evaluate live systems, in order to establish a proper performance baseline for comparison, related work is limited to company whitepapers, presentations or marketing ebooks.

Yottaa[3] offers best practices and other recommendations on variables and metrics in web performance [79].
In [64] Radware[4] analyzed response times of the homepages of e-commerce sites. It ranks the top 10 performing sites and compares the ranks with results from previous years. Furthermore, the whitepaper provides interpretation on the results in the form of key findings.
Compuware[5] published a whitepaper [29] with best practices for benchmarking web and mobile site performance, which covers basic methods of conducting web performance benchmarks. Additionally, the best performing response times for key business transactions across various industries are presented. The differences in average response times are also compared across different browsers and mobile devices.

[3] http://www.yottaa.com
[4] http://www.radware.com
[5] http:/www.compuware.com

CHAPTER 4

Observing Changes in Web Performance

"To acquire knowledge, one must study; but to acquire wisdom, one must observe."

— Marilyn vos Savant,
American columnist and author

The purpose of continually collecting data on web performance is to observe and track changes in the desired performance metrics over time. Reasons to observe these changes are different in their nature, and range from:

- Detecting anomalies
- Identifying patterns
- Ensuring reliability
- Measuring performance improvements after new software releases
- Discovering performance regressions

Whatever the reason may be, the measurements are only useful when we know how to properly analyze them and turn our data into informed decisions.

This chapter provides ways of understanding performance data. It does so by analyzing how common underlying root causes of web performance issues manifest themselves in data gathered through synthetic monitoring. We introduce a taxonomy of root-causes in server performance regressions, which serves as the basis for our experiments.

Furthermore, we describe the methods and steps we take to obtain our results. It also explains how the results will be examined and discussed. Following this, we will outline the design of the simulations that will be conducted, as well as the physical experimental setup enabling the simulations.
We conclude by providing interpretation of the simulation based results, explaining how we can derive certain conclusions. Following this, the display of performance changes in data are described and explored using both visual data analysis as well as statistical changepoint analysis.

Not that, in this thesis, we focus on **server performance**, i.e., the time it takes the server to process a certain request and generate a response.

4.1 Taxonomy of Root-Causes in Server Performance Regression

The underlying causes of performance regressions in web application and web service backends is diverse. They differ significantly in the way they manifest themselves in performance data gathered through active monitoring. We propose a simple taxonomy of root-causes in web performance regression. First, we divide a possible root cause in three main categories, which can be determined through external monitoring:

- Global Delay
- Partial Delay
- Periodic Delay

Further classifications and proper assignment to the main categories in the taxonomy have been derived through semi-structured interviews and discussions with domain experts [49, 75].

In the following, we provide a brief explanation of the general causes of performance delays in computer systems. We then classify the main categories of our taxonomy by grouping the root-causes by the distinguishable effect they have on our web service backend that is also manifested in data that can be recorded by the means of external monitoring. The taxonomy is depicted in Figure 4.1, and further explained in more detail the following.

4.1.1 General Cause of Performance Delays

In general, if we consider performance and computation power of a system, we must consider resources that enable computation. These are usually hardware resources, such as processors, memory, disk I/O, and network bandwidth. We also need to consider the the ways and methods these resources are allocated and utilized.
The demand on resources of a computer system increases as the workload for the application of interest increases. When the demand of the application is greater than the resources that can be supplied by the underlying system, the system has hit its resource constraints. This means the maximum workload of the application has been reached and, typically, the time taken for each request to the application will increase, or with extreme oversaturation, result in system failure.

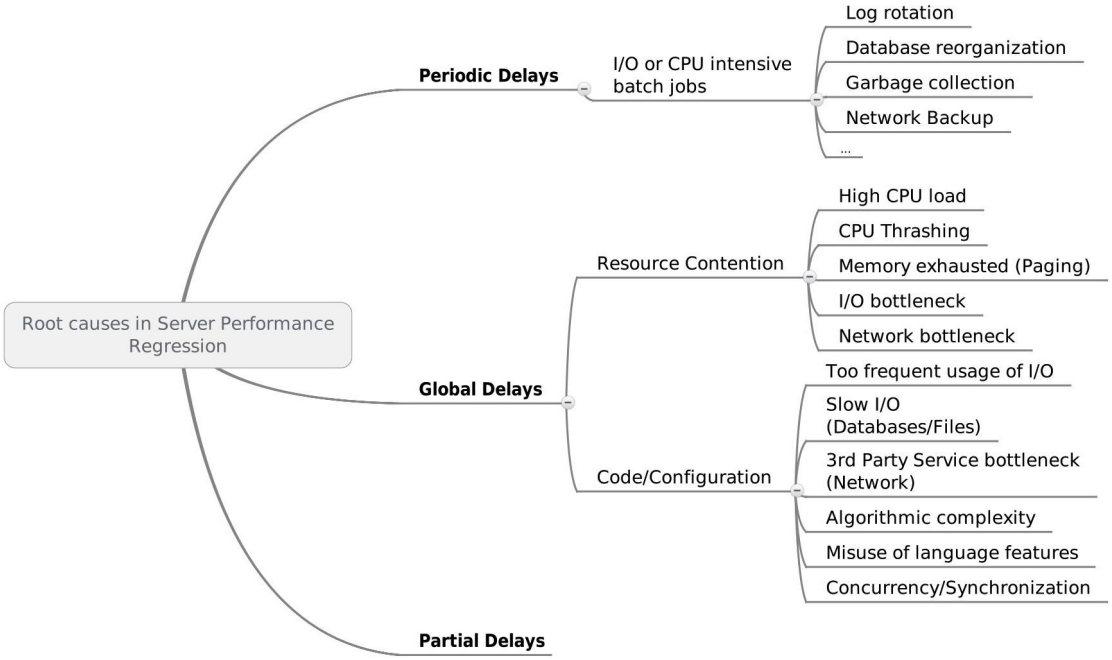

Figure 4.1: Taxonomy of Root-Causes in Server Performance Regression

For web applications and web services, this translates into poor response times or (temporary) unavailability.

A delay in performance as observed through active monitoring can be defined as a negative change in response time at a certain point in time t. This means we look at two observations[1] of response times x_t and x_{t+1} where

$$\lambda = |x_t - x_{t+1}| > c \tag{4.1}$$

where c denotes a certain threshold accounting for possible volatility.

In the following, this simplified notion of performance delays λ over a threshold c will be used in the description of the elements of the taxonomy.

4.1.2 Global Delay

A global delay means that a change in a new deployment or release of the backend system introduced a significant difference in response time λ, which is higher than a defined threshold c on *all* incoming requests. The following sections will briefly outline the causes that might induce

[1] An observation can be a single data point of our recording, the moving average or the moving median as already described in Section 2.2.4

such a global delay. We distinguish between global delays that are caused through resource contention and code or configuration issues.

Resource Contention

I/O Bottleneck

In a database bottleneck the flow of data from the database to the application (usually referred to as a database query) is contended. This means the query takes longer to perform and return data which causes an overall performance delay.

Application Logic delay/Algorithmic Complexity

Delays in application logic refer to problems in performance due to increased code execution time during a request. The performance problem in this case is induced in the development of logical units and algorithms in the backend service that require higher time complexity than previous releases.

Concurrency/Synchronization

The introduction of concurrency by the means of threads or processes may cause global delays due to the need of synchronization. It may also cause temporary service outage when a deadlock situation occurs that cannot be immediately resolved by the underlying operating system or virtual machine.

(Temporary) Service Outage

Service outages are system failures that result into loss of availability. The failure can be either temporary, i.e., through the lack of resources due to too high demand and the inability to scale.

4.1.3 Periodic Delay

In periodic delays, we differentiate between *short-term periodicity* and *long-term periodicity*. They mostly differ in their time parameters, which are explained in the following.

Short-Term Periodicity

Short term periodicity usually happens in micro cycles of the processor and thus manifests itself only in subseconds of time. This kind of periodicity usually happens on all virtual machines with garbage collection.

Garbage Collection

The process of garbage collection tries to recover memory occupied by objects of a computer program that are no longer used by the program. There are many strategies implemented for garbage collection. When requests increase and local memory cannot meet this demand in resources, it causes the garbage collector to work more often, using up other hardware resources,

such as CPU for its processing and disk I/O for swapping, which in turn results in delays. If the garbage collection process freezes the full Virtual Machine, and this freezing lasts too long, the whole application stops its computation for that moment. Of course, the amount of time the application is frozen due to this kind of issue always depends on your internal memory size.
When looking at web performance data that is collected through active external monitoring, delays in subsecond area are often regarded as spikes and can rarely be detected by only looking at the collected data. That means, they are not significant enough to be detected or not regarded as a false positive, because it is too small of a performance change. Nevertheless, delays in subsecond area increase when load increases, then it might even cause unavailability of the service as the lack of memory might result into a memory leak, that in the long run causes the application to fail.

Long-Term periodicity

Delays with long term periodicity last longer and usually happen a few times a day or even less frequent. A periodic delay is mostly not induced through a new deployment or release, but rather through a background process causing certain aspects of the system to use an increased amount of resources, which introduce delays in periodic points in time.

Log rotation

Log rotation is an automated process in server systems administration, where log files that exhibit certain characteristics are archived. The most common characteristics being the date of certain log files, as well as the reach of a certain limit in size of the data or lines stored within the log file. The process is usually configured to run as a periodic cronjob to be fully automated. This process might or might not involve the transfer of the archived logs to a different storage system, which might cause even further issues in performance.

4.1.4 Partial Delay

Partial delays are basically global delays that occur only for certain requests. This situation occurs in an environment that employs any form of load balancing and only a subset of the servers have a release deployed that introduced the performance regression.

This taxonomy does by no means raise the claim of completeness. It is rather an attempt to give an overview of common pitfalls that cause slowness in performance. On its higher level, it captures a certain granularity of issues in server performance that will be the basis of the upcoming simulation design and experiments.

4.2 Simulation Design

We want to identify characteristics in performance data, when certain performance change events occur. Furthermore, we want to focus on which statistical measures are well suited for identifying such changes. To reflect the nature of performance regressions in server backends, we

consider a simulation model of a simple web service environment for our experiments. The model consists of the following components:

- Synthetic Agent Nodes
- Scenario Generation Component
- Dynamic Web Service Component

The simulation model and its communication channels are depicted in Figure 4.2.We consider a system with this architecture where the requests that are incoming from our synthetic nodes are governed by a stochastic process $\{Y(t), t \in T\}$, with T being an index set representing time.

The components are described in more detail in the following sections.

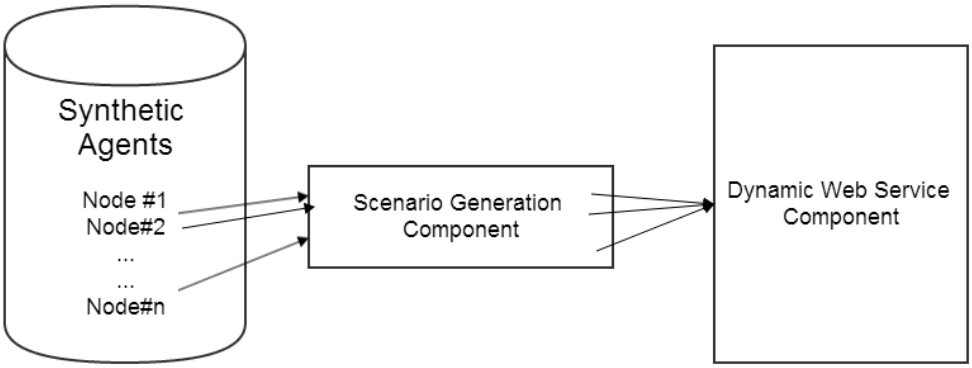

Figure 4.2: Simulation model

4.2.1 Synthetic Agent Nodes

We gather data from our 'Dynamic Web Service Component' by utilizing active, periodic monitoring through synthetic agents. A synthetic monitoring agent acts as a client in order to measure availability and performance metrics such as response time. Every synthetic agent is able to perform active measurements, or synthetic tests. An active measurement is a request to a target URL, where subsequently all performance metrics that are available through the response are obtained. When configuring a set of synthetic tests, we can configure the following parameters:

- **Target URL:** URL of the web service or web application that should be tested.
- **Test interval:** Frequency of the test, e.g., every n minutes, ever hour, etc.
  This parameter is also called sampling interval in the literature, as it retrieves sample measurements of the target system.

- **Test duration:** Total length of the test in terms of time. This parameter could alternatively be given as a *test count* or *sample count*.

4.2.2 Scenario Generation Component

Since only the main classifications of root-causes can be identified through synthetic monitoring, it makes sense to induce changes following the notions of *global delays*, *partial delays*, and *periodic delays* for the simulation.

We achieve this by introducing a *scenario generation component* into our model. It functions as an intermediary between the request sent by the synthetic agent nodes and the web service. Instead of manually injecting faults into our web server, we define a set of scenarios that, subsequently, reflect the desired scenario, i.e., the faults over time in our system. The scenarios need to reflect performance degradation and performance volatility within a certain system, i.e., a single web server. It also needs to take into account possible geographic distribution of the agents, as well as load balancing mechanisms. The following section will introduce a formal model for defining scenarios that reflects these notions that can be used to formally describe certain use cases.

Scenario Definition

We consider certain parameters to compose a complete scenario within our simulation model. Within a scenario, we need to able to specify how our performance metrics (i.e., response times) develop over time, as well as synthetic agents that are actively probing our target system. In the following, we will introduce a formal model and notation for the specification of these scenarios, which is used throughout the thesis:

- On the top-level, we define a scenario $\mathcal{S}$ that describes how our target system that is observed by each of our synthetic agents $\mathcal{A} = \{a_1, a_2, ..., a_n\}$. Each agent observes a development in performance $D_i \in \mathcal{D}$ with $\mathcal{D}$, being the set of all possible developments (Equation 4.2).

$$\mathcal{S} : \mathcal{A} \mapsto \mathcal{D} \tag{4.2}$$

- A development $D \in \mathcal{D}$ maps from a certain point in $t \in T$ of the stochastic process $\{Y(t), t \in T\}$ (driving the requests of the synthetic agent nodes) to an independent random variable $X_i \in \mathcal{X}$, where $\mathcal{X}$ being the set of possible random variables (Equation 4.3).

$$D : T \mapsto \mathcal{X} \tag{4.3}$$

  where $X_i \in \mathcal{X}$ and $\forall\, X_i \sim \mathrm{U}(a, b)$

This formalization allows us to express any performance changes (either positive or negative) as a classification of performance developments over time attributed to specific synthetic agents. More accurately, it models a performance metric as a uniformly distributed random variable of a system at any given time point $t \in T$. Specifying an assignment for every point in time is a tedious and unnecessary exercise. In order to define scenarios in a more efficient and convenient way, we introduce the following notation for developments $D \in \mathcal{D}$:

Simple Developments

$[X_0, t_1, X_1, ..., t_n, X_n]$ defines a *simple development* as a sequence of independent random variables X_i and points in time t_i, further defined in Equation 4.4.

$$[X_0, t_1, X_1, ..., t_n, X_n](t) = \begin{cases} X_0 & 0 \leq t < t_1 \\ X_1 & t_1 \leq t < t_2 \\ \quad \vdots & \\ X_n & t_n \leq t \end{cases} \tag{4.4}$$

This allows us to easily define developments X_i in terms of time spans $t_{i+1} - t_i, i \geq 0$ within the total time of observation. The last development defined through X_n remains until the observation of the system terminates.

Periodic Developments

A *periodic development* is essentially a simple development, which occurs in a periodic interval p. It is preceded by a "normal phase" up until time point n. The "periodic phase" lasts for $p - n$ time units until the development returns to the "normal phase". A periodic development $[X_0, n, X_1, p]^*$ is defined in Equation 4.5.

$$[X_0, n, X_1, p]^*(t) = \begin{cases} X_1 & \text{for } kp + n \leq t < (k+1)p \\ X_0 & \text{otherwise} \end{cases} \tag{4.5}$$

where $k \geq 0$.

Figure 4.3 depicts how a periodic development can be seen over time with given parameters X_0 as the "normal phase" random variable, X_1 as the "periodic phase" random variable, n to define the time span for a "normal phase", p as the periodic interval and $(p - n)$ as the time span for the "periodic phase".

These two defined functions allow us to conveniently define scenarios which adhere to our notions of global, partial and periodic delays. We can now define changes in a declarative way in response times at certain points in time, as well as define changes in periodic time intervals that result in changes in response times for a specific amount of time.

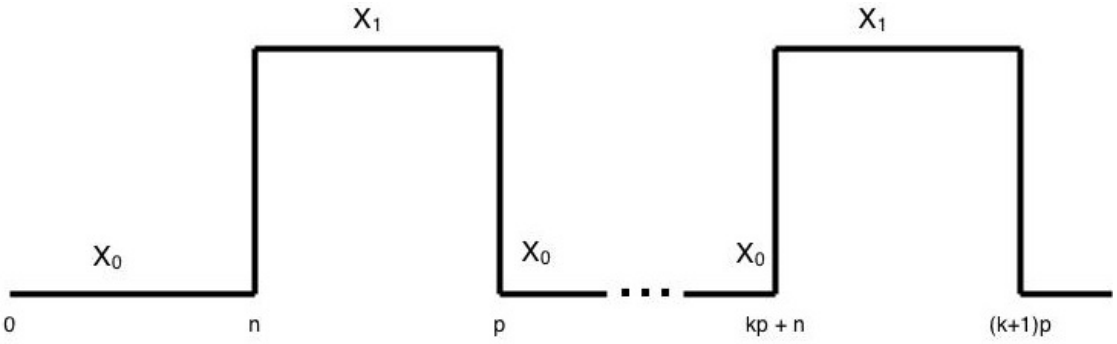

Figure 4.3: Depiction of the periodic development scheme defined in Equation 4.5

Example Scenario

Let us consider the following example to show in both formal and informal notation: **Three synthetic agent nodes** measure response times on the target system. Node#1 observes 100-120ms, up until 20 minutes, then observes 150-175ms in response time. Node#2 observes 110-150ms, up until 45 minutes, then observes 190-210ms. Node#3 always observes 120-125ms.

This particular scenario can be defined as in Equation 4.6.

$$\begin{aligned} \mathcal{S} = \{a_1 \mapsto [X_{a_1,0}, 20, X_{a_1,1}], \\ a_2 \mapsto [X_{a_2,0}, 20, X_{a_2,1}], \\ a_3 \mapsto [X_{a_3,0,\infty}]\} \end{aligned} \tag{4.6}$$

where Equation 4.7 holds.

$$\begin{aligned} X_{a_1,0} \sim \mathrm{U}(100, 120), X_{a_1,1} \sim \mathrm{U}(150, 175) \\ X_{a_2,0} \sim \mathrm{U}(110, 150), X_{a_2,1} \sim \mathrm{U}(190, 210) \\ X_{a_3,0} \sim \mathrm{U}(120, 125) \end{aligned} \tag{4.7}$$

A more intuitive, *informal description* of the simulation scenario parameters can be given using descriptive parameter names displaying them in tabular form (Table 4.2.2).

In the scenario definitions for the conducted experiments, we will categorize our scenarios in global, partial, and periodic delay scenarios. We will use both the formal and informal approach to describe the scenario, and also provide an example practical use case that adheres to the theoretical parameters.

4.2.3 Dynamic Web Service Component

The *dynamic web service component* works together with the *scenario generation component* to achieve the reflection of performance issues for the *synthetic agent nodes*. In order to do so, it

Synthetic Node Agents	a_1, a_2, a_3
Initial response times	
a_1 response time range	100-120ms
a_2 response time range	110-150ms
a_3 response time range	120-125ms
Deferred response times	
a_1 after 20 minutes	150-175ms
a_2 after 45 minutes	190-210ms

Table 4.1: Example of informal description of simulation parameters in tabular form

offers an endpoint that simulates delays over parameters passed from a specific scenario script. This means that the declared scenarios are executed within the web service component and thus simulate workload through the parameters given in the scenarios.

4.3 Experimental Setup

The experiments based on the described simulation model were executed in a local test-bed consisting of 5 synthetic agent nodes and 1 dynamic web service component. The specific setups are described in the paragraphs below.

4.3.1 Synthetic Agent Node Setup

The synthetic nodes operate in the local network as well, every of the 5 nodes sends out a request every 5 minutes that is handled by a scheduler that uniformly distributes the requests over time. This results into 1 request/minute that is being send out by a synthetic node agent that records the data. The physical node setup is described in Table 4.2.

CPU	Intel Pentium(R) 4, 3.4 GHz x 2 (Dual Core)
RAM	1.5 GB
OS	Ubuntu 12.04 64-bit

Table 4.2: Synthetic node setup

4.3.2 Web Server Setup

The webservice running on the server has been implemented in Ruby and runs over the HTTP server and reverse proxy nginx and Unicorn as the webserver for Ruby that runs on its own proxy. The physical web server setup is described in Table 4.3.

CPU	Intel Pentium(R) 4, 3.4 GHz x 2 (Dual Core)
RAM	1.5 GB
OS	Ubuntu 12.04 64-bit

Table 4.3: Web server setup

Random Number Generation

During simulation, a random number generator is applied to generate artificial user behavior as specified in the distributions of our simulation scenario (see Section 4.2.2). Such user behavior data can be represented efficiently with common random numbers [41]. However, as with deterministic load tests, replaying user behavior data may not always result into the same server response. Even with the server state being the same, server actions may behave nondeterministically.
To adhere the production of independent random variables that are uniformly distributed (as described in the formal specification of a scenario in Section4.2.2) the scenario generation component uses MT19937, Mersenne twister [53] as a random number generator.

4.4 Methodology

In general, change in backend performance translates to the concept where the time it took the server to process a certain request and generate the right response has become significantly slower or faster. From a statistical point of view, we consider that the performance of a system has changed if the underlying probability density function has experienced a fundamental shift. In this research, we simulate web performance regressions induced by changes in the the backend of a web application. We periodically monitor the system within an experimental setup, measuring response metrics of the application through synthetic agents and store the observed data points as a time series. We then use time series analysis to examine the development of the observations. The following sections briefly explain the employed analysis approaches.

4.4.1 Simulation Scenarios

We define simulation scenarios (according to our formal model) that induce delays that capture the main categories of root causes that we have defined in our taxonomy. For each theoretical scenario, we describe a possible real life use case scenario in order to establish a point of reference.

4.4.2 Data Collection

We collect data through active monitoring as described in the simulation design. The synthetic agents send out HTTP GET requests every n minutes from m agents and collect all request component metrics that have been described in the background section. As already described in our physical experimental setup, the simulation will sample ever 1 minute resulting in 1 new observation of our system every minute. Each observation is stored in a database with the

corresponding timestamp. This results into a time series for each of our simulation scenarios, which in the following can be explored and analyzed.

4.4.3 Exploratory Data Analysis

Our first analysis approach is to examine the time series visually over graphs in order to explore and gain further understanding on the effect specific underlying causes have on the resulting data. We also determine what kind of statistical attributes are well suited to identify key characteristics of the data sample and for humans to properly observe the performance change. For this initial analysis we plot the raw time series data[2] as line charts. This allows for visual exploratory examination, which we further complement with proper interpretations of the data displayed in the visualization that correlates with induced changes in the server backend.

4.4.4 Statistical Analysis

After manually inspecting the time series over visual charts, we evaluate the observation of performance changes by the means of statistical changepoint analysis. Specifically, we evaluate algorithms that are employed in the R [63] package **changepoint**.

We are not interested in the detection of spikes or other phenomena that can be considered as outliers, but rather in the detection of fundamental shifts in our data that reflect a longer standing performance degradation. We also want to keep false positives low, and determine whether a change is actually a change that implies a root-cause that requires some kind of action. Thus, we also need to determine the magnitude of the change.

Using statistical analysis, we evaluate the detection of the moment of the change as well as the magnitude of the change (comparison of the changes in the mean). For this, we recall the simplistic view on delays we introduced in our taxonomy: $\lambda = |x_t - x_{t+1}| > c$ where x_t and x_{t+1} represent two consecutive measurements in time, λ is the absolute difference quantifying the magnitude of the change and c denotes a certain threshold. We adapt this simplistic model of change as follows. Instead of comparing two consecutive data points, we compare the changes in mean at the time point where changes of variance occur. In other words, we compute the difference between the mean of the distribution before the detected changepoint occurred, $\mu_{<\tau}$, and the mean of the distribution after the detected changepoint occured, $\mu_{>\tau}$, where τ denotes the changepoint. This difference is then our new λ, and can be defined as in Equation 4.8 by replacing two variables.

$$\lambda = |\mu_{<\tau} - \mu_{>\tau}| > c \tag{4.8}$$

The threshold c is difficult to determine. When the threshold is set up too high, legitimate changes in performance that were caused by problems may not be detected and the system is in risk of performance degradation and, in the long run, may be in risk of unavailability. On

[2]Raw in this context means that we will not smooth the data by any means and will not apply statistical models in any way.

the other hand, the threshold can be set unnecessarily sensitive, leading to a higher likelihood of detection of change in normal system operation, i.e., false positives. The value of the threshold depends on the application and must be either set by a domain expert or be determined by statistical learning methods through analysis of past data and patterns. If a SLA or other contractual agreement is in place, then the threshold may be determined by agreed values (e.g., response time mean always below 3 seconds). Sometimes it is useful to compare new metrics in relation to old metrics, this is a very simple way of statistical learning through past data. In the conducted experiments, we set the threshold as in Equation 4.9.

$$c = \mu_{<\tau} \cdot 0.4 \tag{4.9}$$

This means that if the new metric after the changepoint $\mu_{>\tau}$ is 40% above or below the compared to the old metric $\mu_{<\tau}$, the change is considered a *real change* as opposed to a *false positive* in the opposite case. If we want to consider positive changes as well, the calculation of the threshold must be extended in a minor way to not yield into false negatives due to a baseline that is too high (Equation 4.10).

$$c = \min(\mu_{<\tau}, \mu_{<\tau}) \cdot 0.4 \tag{4.10}$$

Note that the threshold 40% of the mean was chosen after discussions with a domain expert, as it is seen as an empirically estimated baseline. In practice, a proper threshold depends on the type of web application, service level agreements, and other factors and is most probably determined by own empirical studies.

R Package 'changepoint'

The R package 'changepoint' implemented by Killick and Eckley [46] offers multiple changepoint algorithms. In our experiments, we apply the algorithm *Binary Segmentation* on our time series data for the evaluation. Binary Segmentation is the most widely used method for changepoint detection noted in the literature on changepoint analysis. The method has its roots from early works including [32, 66, 69], but is also discussed in more recent publications in [31, 35, 47, 61]. The algorithm starts out by applying a single changepoint test statistic on the entire data set. If a changepoint has been identified, the set is split into two sets at the estimated changepoint location. This procedure is repeated and applied on each of the two new sets recursively. The method continues until no changepoints are found in any parts of the remaining data sets. Binary segmentation is an approximate algorithm, as it only considers a subset of the 2^{n-1} possible solutions for changepoints, and is, therefore, computationally fast with a computational complexity of $\mathcal{O}(n \log n)$. We are particularly interested to compare the results of the algorithmically detected changepoints (taking into consideration the defined threshold c) with the ones we determined by visual inspection of raw performance charts.

4.4.5 Threats to Validity

In this research, we employ a simplified simulation model in order to show how significant perfromance changes can be detected in continuously observed data of a constructed target system. The creation and design of a simulation model comes with inherent risk concerning the validity of the results when applied to real-life systems. The following sections will briefly outline the concerns that have been considered to be a potential threat to validity, but nonetheless has been addressed through the design of our model.

Workloads and Seasonality

In our simulation design, we do not actively inject the notion of web traffic workloads (as has been, for instance, taken in consideration in [52]), but rather simulate the variability of workloads (and therefore response times) in web services through specific scenarios parameters.

Network and Geography

The simulation bed is constructed in a local area network (LAN). Thus, the notions of network traffic and network volatility in wide area networks (WAN) are completely omitted to limit the interference of network noise in the recorded data. Of course, noise is still present even in LANs, but is limited to very few factors within the data center, as opposed to the possible factors in WANs. This also means that there is, for instance, no increased response time due to DNS resolution. While network and DNS do play a major role in web performance, the focus of this thesis lies within detecting changes in server performance.

Due to the mentioned locality, geographic location of the simulated users in synthetic monitoring is not implicitly incorporated in our results. However, due to the simulation design - which contains many independent synthetic node agents - we explicitly simulate issues due to higher response times in only certain geographic locations (i.e., certain agents) in our scenarios. This makes the scenario more controllable as opposed to running it through synthetic node agents in distant data centers due to the volatility of network latency.

4.5 Simulation Scenarios

In this section, we describe common scenarios that will be simulated within our presented experimental setup. First, we formally define the parameters that actually form a certain scenario. The aim of each scenario is to properly represent one of the top levels of the root causes previously defined in our taxonomy. We evaluate the results and discuss possible interpretations of the recorded data.

4.5.1 Global Delay

As already described in our taxonomy, a global delay is the introduction of a significant difference in response time on all incoming requests, i.e., it affects all users accessing the resource in

question.
In the following, we provide a brief description of an example use case where this global delay would take place to illustrate a connection in a real life software development process.

Scenario Use Case Example

- New feature needs to be implemented for a new release.
- Junior Developer in charge of the new feature introduces a new (slow) database query, causing significantly higher overall response times.
- The slow query is not caught in QA and the new release is deployed to all users.

Scenario Parameters

The parameter for this global delay is defined in Equation 4.11.

$$\mathcal{S}_G = \{a_i \mapsto [X_{a_i,0}, 420, X_{a,1}] \mid a_i \in \{a_1, a_2, a_3, a_4, a_5\}\} \tag{4.11}$$

For every index i, we define the initial response time range as in Equation 4.12.

$$\begin{aligned} X_{a_1,0} &\sim \mathrm{U}(90, 115) \\ X_{a_2,0} &\sim \mathrm{U}(100, 130) \\ X_{a_3,0} &\sim \mathrm{U}(110, 140) \\ X_{a_4,0} &\sim \mathrm{U}(95, 110) \\ X_{a_5,0} &\sim \mathrm{U}(100, 110) \end{aligned} \tag{4.12}$$

As well as the range for the global change over all agents in Equation 4.13.

$$X_{a,1} \sim \mathrm{U}(150, 175) \tag{4.13}$$

Informal Description

For the given formal parameters we also give an informal tabular description in Table 4.4.

4.5.2 Partial Delay

A partial delay scenario consists of requests that, at some point in time, cause a delay on a subset of the incoming requests. In the following, we provide a brief description of two example use cases where a partial delay would take place to illustrate a connection in a real life software development process.

Synthetic Node Agents	a_1, a_2, a_3, a_4, a_5
Initial response times	
a_1 response time range	90-115ms
a_2 response time range	100-130ms
a_3 response time range	110-140ms
a_4 response time range	95-110ms
a_5 response time range	100-110ms
Deferred response times	
a_1 after 420 minutes	150-175ms
a_2 after 420 minutes	150-175ms
a_3 after 420 minutes	150-175ms
a_4 after 420 minutes	150-175ms
a_5 after 420 minutes	150-175ms

Table 4.4: Informal description of the global delay simulation scenario $\mathcal{S}_G$

Scenario Use Case Example

Example#1

- New feature needs to be implemented for new release.
- Junior Developer in charge of the new feature introduces a new (slow) database query.
- The slow query is not caught in QA.
- Due to the deployment strategy, the new release is only rolled out to **20%** of all users.

Example#2

- A web application sits behind a load balancer handling 5 servers.
- One of the servers encounters unexpected hardware issues, which result in higher response times.
- The balancer uses a 'Round Robin' approach as its load balancing method. **20%** of all users perceive the application with higher response times.

Scenario Parameters

The parameter for this partial delay is defined in Equation 4.14.

$$\begin{aligned} \mathcal{S}_P = \{a_i \mapsto [X_{a_i,0}, \infty] | a_i \in \{a_1, a_2, a_3, a_4\}, \\ a_5 \mapsto [X_{a_5,0}, 360, X_{a_5,1} \\ \} \end{aligned} \tag{4.14}$$

For every index i we define the initial response time range as in Equation 4.15

$$\begin{aligned} X_{a_1,0} &\sim \mathrm{U}(115, 125) \\ X_{a_2,0} &\sim \mathrm{U}(115, 120) \\ X_{a_3,0} &\sim \mathrm{U}(120, 145) \\ X_{a_4,0} &\sim \mathrm{U}(105, 115) \\ X_{a_5,0} &\sim \mathrm{U}(110, 120) \end{aligned} \tag{4.15}$$

As well as the range for the partial change for agent a_5 in Equation 4.16.

$$X_{a_5,1} \sim \mathrm{U}(140, 165) \tag{4.16}$$

Informal Description

For the given formal parameters we also give an informal tabular description in Table 4.5.

Synthetic Node Agents	a_1, a_2, a_3, a_4, a_5
Initial response times	
a_1 response time range	115-125ms
a_2 response time range	115-120ms
a_3 response time range	120-145ms
a_4 response time range	105-115ms
a_5 response time range	110-120ms
Deferred response times	
a_5 after 360 minutes	140-165ms

Table 4.5: Informal description of partial delay simulation scenario $\mathcal{S}_P$

4.5.3 Periodic Delay

A periodic delay takes place when, due to a (background) process, resource contention occurs and, subsequently, higher usage of hardware resources leads to higher response times for a certain amount of time. This scenario addresses those processes that are (usually) planned ahead and are executed within a specific interval.

Scenario Use Case Example

Example#1

- Log files of an application make up a large amount of the server's disk space
- The system administrator creates a cron job to process older log files and move them over the network

- The process induces heavy load on CPU (processing) and I/O (moving over network), which result into temporarily higher response times
- The cron job is configured to take place in periodic intervals to ensure the server has enough disk space

Scenario Parameters

The parameter for this periodic delay is defined in Equation 4.17

$$\mathcal{S}_{PD} = \{a_1 \mapsto [X_0, 45, X_1, 65]^*\} \tag{4.17}$$

The response time ranges are defined as in Equation 4.18.

$$\begin{aligned} X_0 &\sim \mathrm{U}(95, 115) \\ X_1 &\sim \mathrm{U}(160, 175) \end{aligned} \tag{4.18}$$

Informal Description

For the given formal parameters we also give an informal tabular description in Table 4.6.

Synthetic Node Agents	a_1
Initial response times	
a_1 response time range	95-115ms
Deferred response times	
a_1 every 45 minutes for 20 minutes	160-175ms

Table 4.6: Informal description of periodic delay simulation scenario $\mathcal{S}_{PD}$

4.6 Results and Interpretation

The scenario simulations as defined in the previous section have been executed within the experimental setup following the described methodology. The resulting data has been analyzed and will be presented and discussed thoroughly in this chapter. At first, we display a raw plot of the resulting time series data without any filters and interpret its meaning. Further, we apply the moving average smoothing filter (with a window size $w = 5$) to each resulting time series and conduct a changepoint analysis with support of the methods and algorithms in the R package 'changepoint' [46]. We discuss methods that analyze both changepoints in the mean and the variance.

4.6.1 Global Delay

The global delay forms the basis of our assumptions on how performance changes can be perceived on server backends. Both, the partial and periodic delay, are essentially variations in the variables time, interval and location of a global delay. Hence, the findings and interpretations of this section on global delays is the foundation for every further analysis and discussion.

Exploratory Visual Analysis

In Figure 4.4, we see a non-filtered time series plot for our global delay scenario. We can clearly see the fundamental change in performance right after around 400 minutes of testing. The data before this significant change does seem volatile. There are a large amount of spikes occurring, though most of them seem to be outliers that might be caused in the network layer. None of the spikes sustain for a longer period of time, in fact between the interval around 150 and 250 there seem to be no heavy spikes at all. The mean seems stable around 115ms in response time and there is no steady increase over time that might suggest higher load variations. Thus, we can conclude that the significant performance change has occurred due to a new global release deployment of the application.

Statistical Changepoint Analysis

We apply changepoint analysis to the smoothed time series with a moving average window size of 5. In Figure 4.5, we can immediately see how the smoothing affected the chart, compared to the chart with the raw data in Figure 4.4: The spikes, i.e., the random noise, have been canceled out to a certain extent, making the main signal stronger and easier to identify. This makes it easier for our statistical analysis to focus on our signal and detect the proper underlying distributions. While this is definitely a pleasant effect of every smoothing technique, we also need to keep in mind that every model that we apply contains its own assumptions and own errors that need to be considered. What can further be seen in Figure 4.5 is the changepoint in the variance, denoted by a vertical red line at the changepoint location. Table 4.7 contains the numerical results of the changepoint in variance analysis and indicates the estimation for the changepoint τ_1 at 422. This number coincides approximately with our own estimation we concluded in the previous section for Figure 4.4.
Next, we look at Figure 4.6, where the change in the mean is indicated by horizontal lines

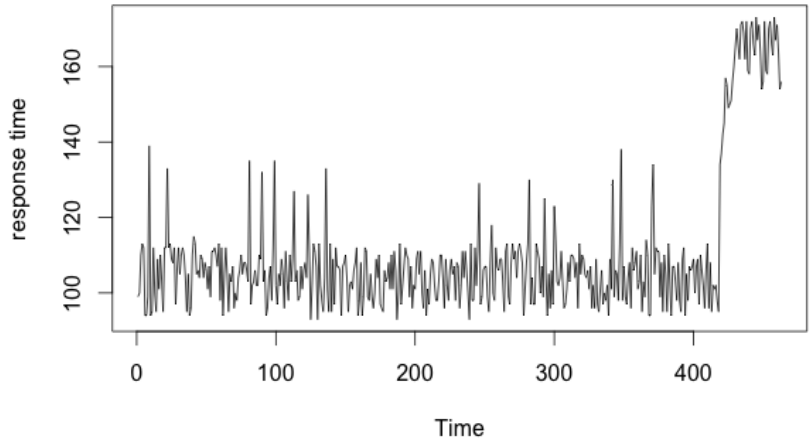

Figure 4.4: Results of Global Delay – non-filtered time series plot

depicting the mean value for each segment that has been detected. This method has detected more changepoints than the previous analysis, which can be not clearly seen in Figure 4.6 due to the very small change in the mean. The estimated numerical values for the changepoints are listed in Table 4.8. The table also lists the mean values, as well as the calculated threshold $c = \mu_{<\tau} \cdot 0.4$[3] (see Section 4.4,). The last column also states whether or not a detected changepoint in the mean is an *actual changepoint* as defined by our notion of *significant change* where $\lambda = |\mu_{<\tau} - \mu_{>\tau}| > c$, or a *false positive*. As we can see only one of the estimated changepoints has been identified as an actual changepoint, when considering the threshold c.
This shows that detecting a fundamental change is a difficult undertaking, especially considering non-parametric statistical analysis, as in our case. Post-processing and analysis of the estimated changepoints and its according mean values is important to avoid false positives.

τ	$\sigma^2_{<\tau}$	$\sigma^2_{>\tau}$
422	10.695	67.731

Table 4.7: Changepoint in the variance for global delay simulation scenario $\mathcal{S}_G$

[3]This is the first equation listed in the section of the methodology that does not take into account improvements in performance

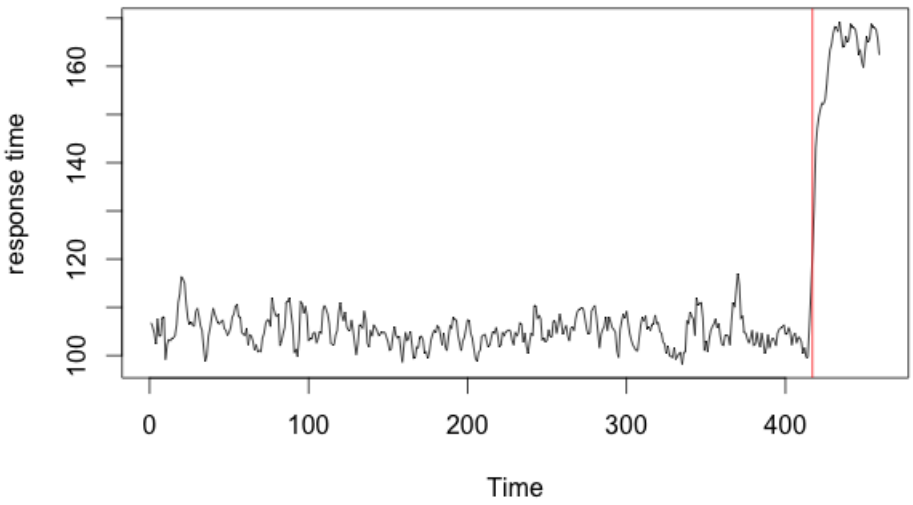

Figure 4.5: Results of Global Delay – red vertical line indicates changepoint in variance

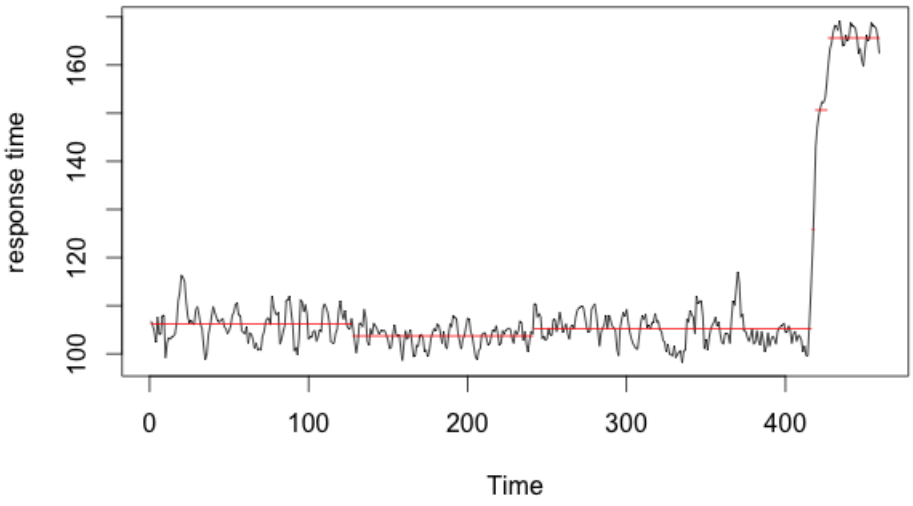

Figure 4.6: Results of Global Delay – red horizontal lines indicate the mean values for each detected different distribution

τ	$\mu_{<\tau}$	$\mu_{>\tau}$	λ	c	Actual CP/False Positive
127	106.18	103.7	2.48	42.47	False Positive
241	103.7	105.32	1.62	41.48	False Positive
366	105.32	110.85	5.53	42.12	False Positive
374	110.8	103.62	7.23	44.32	False Positive
421	103.62	150.65	47.03	41.44	**Actual Changepoint**
427	150.65	165.62	14.97	60.62	False Positive

Table 4.8: Changepoint in the mean for global delay simulation scenario $\mathcal{S}_G$

4.6.2 Partial Delay

Partial delays are global delays that only occur on a certain subset of requests and, therefore, need different techniques to properly examine the time series data and diagnose a performance degradation. Experiments on simulating partial delays have found that detection of a changepoint in partial delays, or even the visual detection of performance change is not at all trivial. In this section we introduce proper ways to handle this problem.

Exploratory Visual Analysis

As before, we plot the time series data and look for changes in our performance. In Figure 4.7, we see a rather stable time series chart, relatively volatile, i.e., spiky, due to the higher amount of conducted tests and probably random noise due to the network. Around the time points 460-500 and 1600-1900, we see a slight shift, but nothing alarming that would be considered a significant change. From this first visual analysis, we would probably conclude that the system is running stable enough to not be alerted.

However, that aggregation hides a significant performance change. The convenience, or sometimes necessity, of computing summary statistics and grouping data to infer information this time concealed important facts about the underlying system. In order to detect this performance change, we have to look at additional charts and metrics.
In Figure 4.8, we plot the same aggregation as in Figure 4.7, but also plot the 90th percentile of the data to come to a more profound conclusion: There actually has been a performance change that is now very clear due to our new plot of the 90th percentile. While this can mean that percentiles also show temporary spikes or noise, it also means that if we see a significant and persisting shift in these percentiles, but not in our average or median, that a subset of our data, i.e., a subset of our users, indeed has experienced issues in performance and we need to act upon this information. Another way of detecting issues of this kind is to plot all data points in a scatterplot. This allows us to have an overview of what is going on and to identify anomalies and patterns more quickly. As we can see in Figure 4.9, a majority of data points is still gathered around the lower response time mean. But we can also see clearly that there has been a movement around the 400 time point mark that sustains over the whole course of the observation, building its own anomaly pattern. The scatterplot is a useful tool to further examine a data set, but might contain maybe too much information for a first glance, especially for users new to

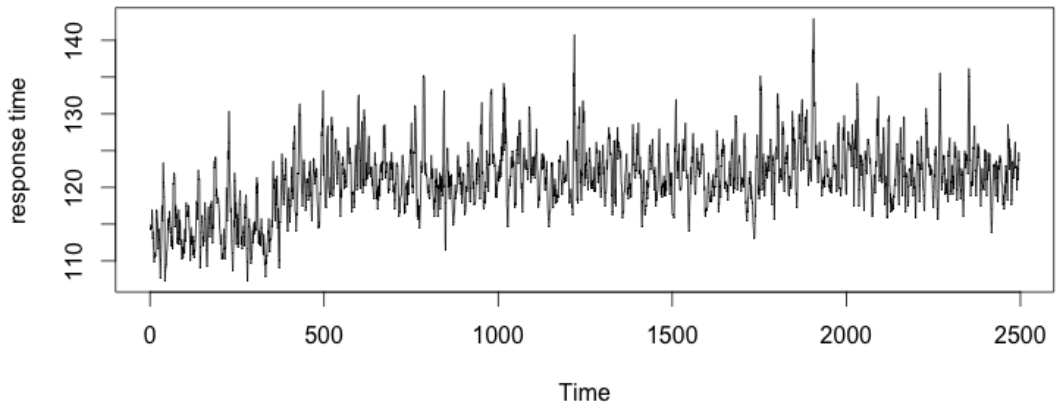

Figure 4.7: Partial delay scenario results time series plot

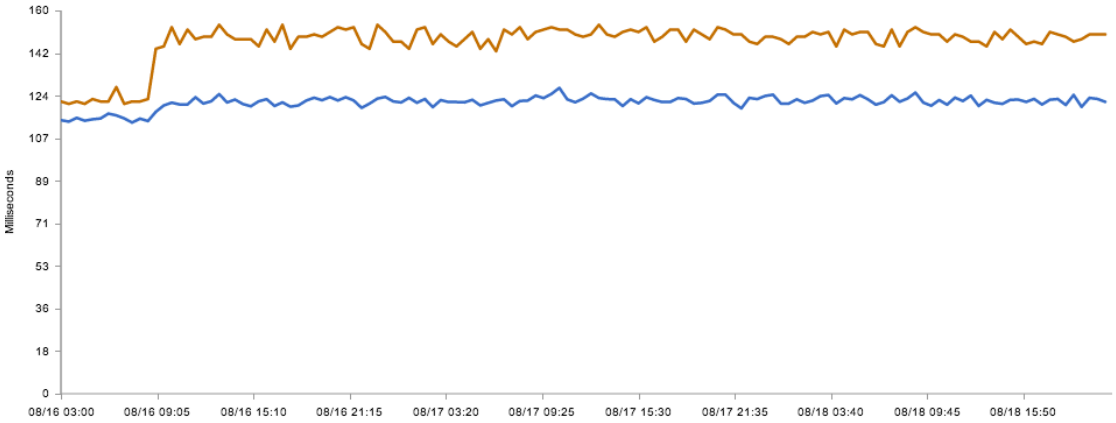

Figure 4.8: Partial delay scenario results time series plot, together with 90th percentile

performance analysis.

Statistical Changepoint Analysis

Analyzing both the changepoints in the variance in Table 4.9 and Figure 4.10, as well as the changepoints in the mean in Table 4.10 and Figure 4.11 yields no surprise following our initial exploratory visual analysis. The changes that have been identified are not significant enough to be detected through the mean and the variance respectively. Although we need to point out that both analyses actually detected the actual significant changepoint around the time point 374-376, but are disregarded as false positives by our post-processing step of checking the threshold.

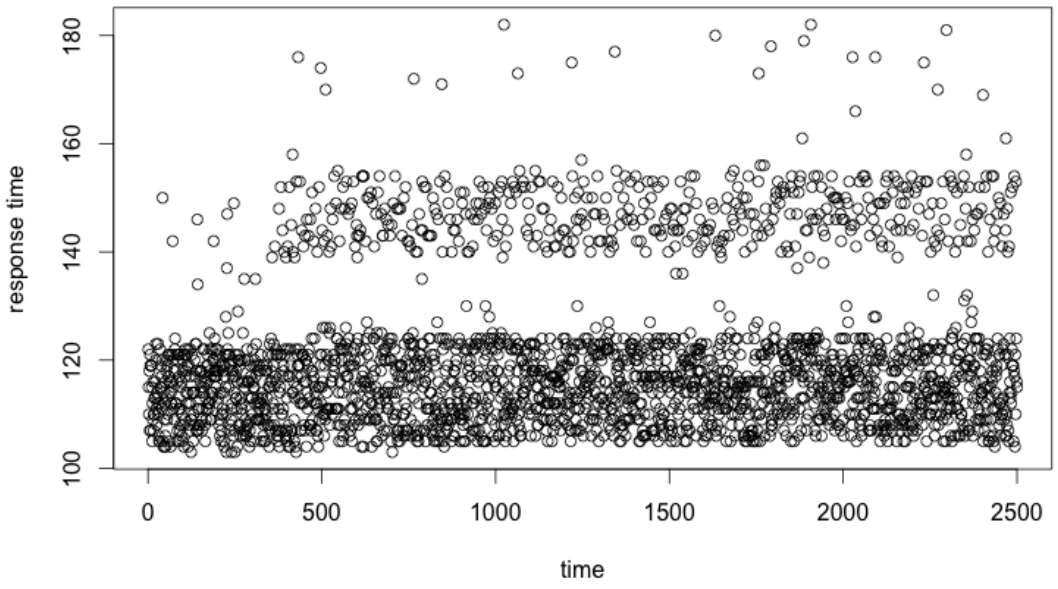

Figure 4.9: Partial delay scenario results depicted as a scatterplot

τ	$\sigma^2_{<\tau}$	$\sigma^2_{>\tau}$
374	12.88	24.25
1731	24.25	12.43
1911	12.43	6.63

Table 4.9: Changepoint in the variance for partial delay simulation scenario $\mathcal{S}_P$

Thus, our post-process actually resulted into a *false negative*. This is usually a sign that an indicator (in our case the threshold c) needs to be adjusted or rethought completely. However, before this kind of decision can be made, more information has to gathered on how this indicator has performed in general (i.e., more empirical evidence on false negatives, ratio between false positives and false negatives, etc.).
In order for our regular statistical analysis process, as applied previously, to properly work we need a further pre-processing step. Neither mean nor variance can detect the performance change, therefore, we need to consider a different metric. In our exploratory analysis, we concluded that the 90th percentile was able to detect the change. Thus, we need to apply our changepoint analysis to percentiles in order to detect a significant shift for partial delays.

τ	$\mu_{<\tau}$	$\mu_{>\tau}$	λ	c	Actual CP/False Positive
376	114.74	121.92	7.18	45.88	False Positive
1710	121.92	118.91	3.01	48.76	False Positive
1756	118.91	124.1	5.19	47.56	False Positive
2035	124.1	122.27	1.83	49.64	False Positive

Table 4.10: Changepoint in the mean for partial delay simulation scenario $\mathcal{S}_P$

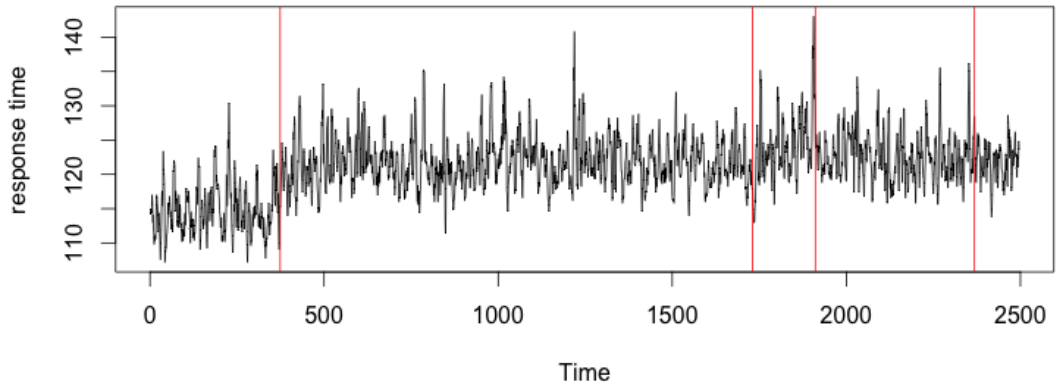

Figure 4.10: Results of the partial delay scenario – red vertical lines indicate changepoints in variance

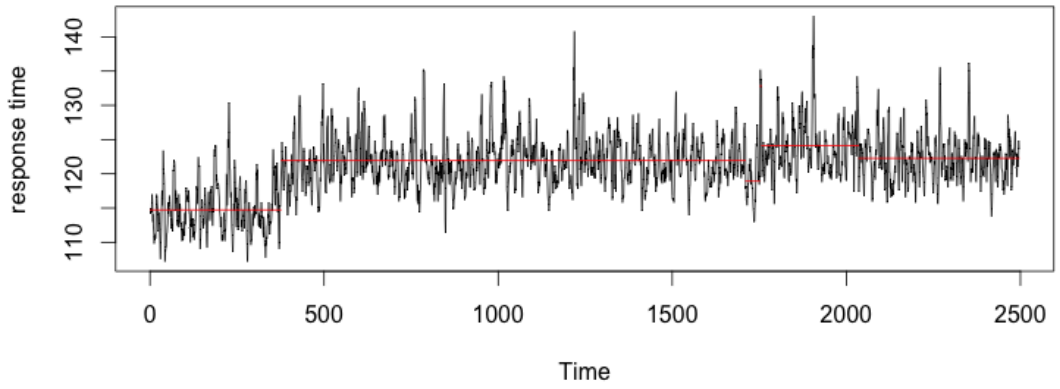

Figure 4.11: Results of partial delay scenario – red horizontal lines indicate the mean values for each detected different distribution

4.6.3 Periodic Delay

Periodic delays are either global or partial delays that occur in specific intervals, persist for a certain amount of time, and then performance goes back to its initial, 'normal' state.

Exploratory Visual Analysis

For this experiment we recorded enough values to result into three periods that indicate a performance degradation for a certain amount of time before returning back the system returns to its normal operation. The intuition we have on periodic delays can be clearly observed in Figure 4.12. Between the phases, we have usual performance operation with usual volatility, and in between we see fundamental shifts that persist for approximately 20 minutes before another shift occurs. Seeing periodic delays is fairly simple, as long as we are looking at a large enough scale. If we would have observed the time series within the periodic phase, before the second shift to the normal state occurred, we might have concluded it to be a simple global delay. Thus, looking at time series at a larger scale might help to identify periodic delays that would have otherwise been disregarded as short-term trends or spikes.

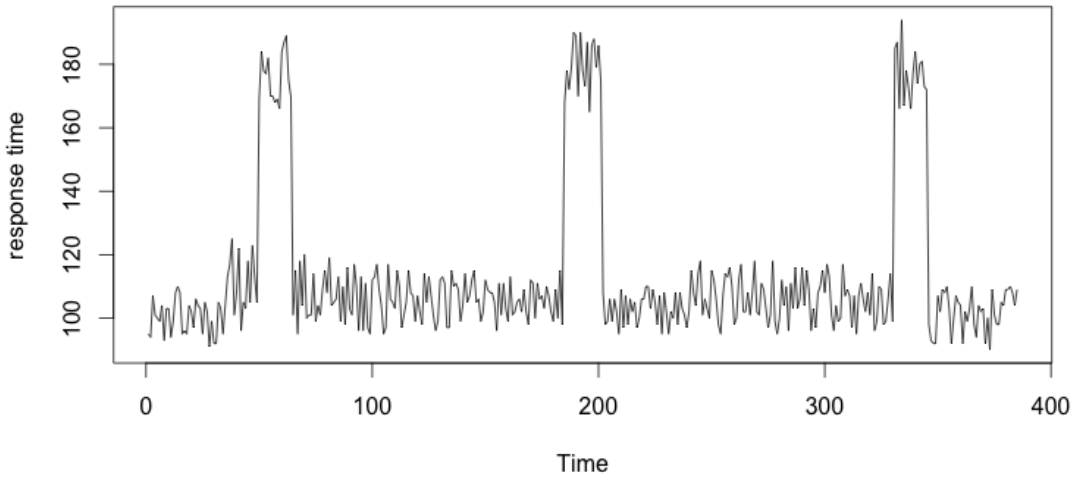

Figure 4.12: Periodic Delay scenario results as a time series plot

Statistical Changepoint Analysis

While the exploratory visual analysis in periodic delays was straight forward, the changepoint analysis brings some interesting insights. Figure 4.13 and Table 4.11 show the analysis for the changepoints in the variance, which mostly coincide with our visual assessment. Merely the first detected changepoint in the variance is a false negative.

τ	$\sigma^2_{<\tau}$	$\sigma^2_{>\tau}$
32	5.21	20.22
46	20.22	156.25
62	156.25	11.73
181	11.73	233.96
199	233.96	16.3
327	16.3	180.3
343	180.3	25.16

Table 4.11: Changepoint in the variance for periodic delay simulation scenario $\mathcal{S}_{PD}$

τ	$\mu_{<\tau}$	$\mu_{>\tau}$	λ	c	Actual CP/False Positive
33	100.47	113.27	12.8	40.18	False Positive
47	113.27	168.11	54.84	45.31	**Actual Changepoint**
63	168.11	106.18	61.93	42.47	**Actual Changepoint - improvement**
180	106.18	169.35	63.17	42.46	**Actual Changepoint**
199	169.35	110.52	58.83	44.20	**Actual Changepoint - improvement**

Table 4.12: Changepoints in the mean for the periodic delay simulation scenario $\mathcal{S}_{PD}$

The changepoint in the mean yields more interesting results. First of all, we needed to adjust the calculation of the threshold c (4.9), in order to also detect performance improvements as proper changepoints. We recall Equation 4.10 defined in the methodology: $c = \min(\mu_{<\tau}, \mu_{<\tau}) \cdot 0.4$, which now allows us to examine both degradations and improvements in performance. In Table 4.12 on changepoints in the mean we indicate an improvement at a proper changepoint with an additional adjective 'improvement' in the last column. Contrary to the results in the other experiments the number of false positives is very low at only one. When looking at the result in Figure 4.14, we can observe another interesting phenomena we have not seen before, a false negative. The third period was not detected as a changepoint in the mean by the algorithm, although it was detected by the changepoint in the variance method. This means, in order to avoid false negatives, we should apply both changepoint in the mean and variance analysis.

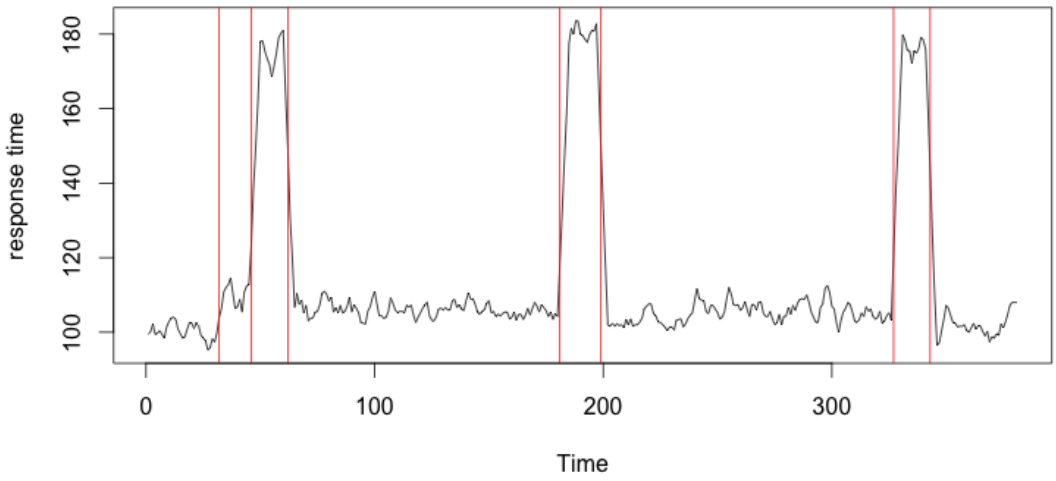

Figure 4.13: Results of the periodic delay scenario - red lines indicate changepoints in the variance

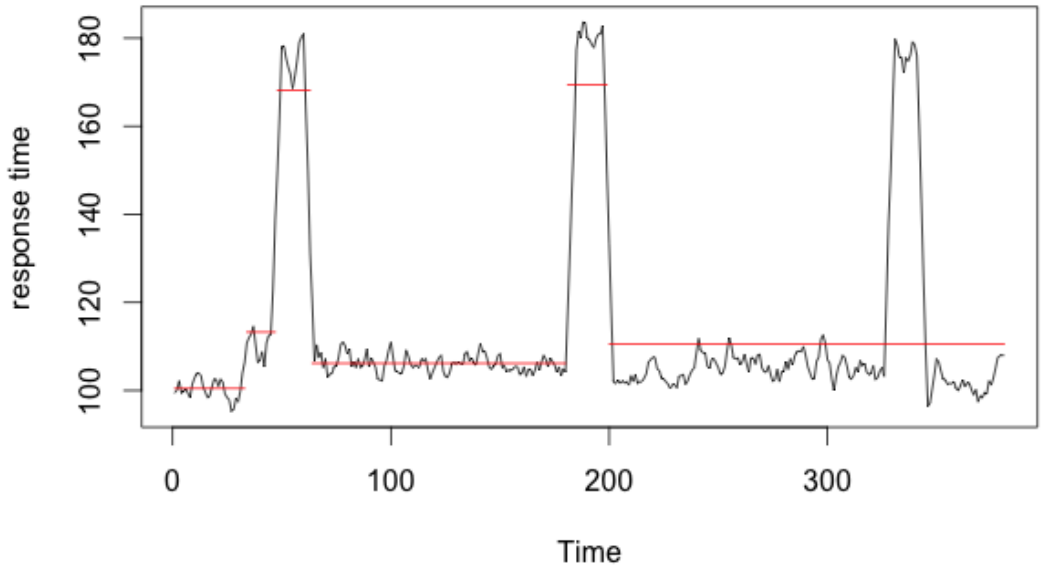

Figure 4.14: Results of the periodic delay scenario - red lines indicate changepoints in the mean

CHAPTER 5

Web Performance Benchmarking

“No man is happy but by comparison."

— Thomas Shadwell, English poet

Benchmarking is a term used throughout multiple fields in science, engineering and business, and in its basic form describes a way of comparing certain indicators. In the context of computing, benchmarking is an essential method for performance evaluation on actual physical machines [51]. Performance benchmarking refers to the execution of a set of tests on different computing units and measuring the results that are used to evaluate the performance of a system. We differentiate between benchmarks that evaluate systems under well-defined, synthetic workload and systems in production.
Web performance benchmarking, as described in this chapter, aims on comparing web applications and web services in production in order to conduct a comparative analysis between the subjects of the benchmark. A definition in [1] captures the idea of benchmarking as we will devise it in this thesis rather accurately:

> *"Benchmarking is the process of continuously measuring and comparing an organization against business leaders anywhere in the world to get information that will help the organization take action to improve its performance"*

It suggests a structured process in order to learn from competitors through evaluation of performance metrics. Furthermore, it suggests that the results from the observation should derive appropriate actions for improvement. While this quote is a definition for benchmarking from a business perspective, it strongly correlates with the notion of benchmarking that we will use.
We develop a general methodology for web performance benchmarking based on statistical analysis of finite collections of observed data from production systems of immediate competitors

through synthetic monitoring. Our approach is then applied in a case study conducted at Catchpoint Systems, Inc.[1] to demonstrate the advantages of the method. Finally, the results of the case study are presented and briefly discussed.

5.1 Methodology

We define a general methodology for benchmarking based on data sampling through synthetic monitoring of target systems. The defined steps are illustrated in Figure 5.1, and are explained in the following.

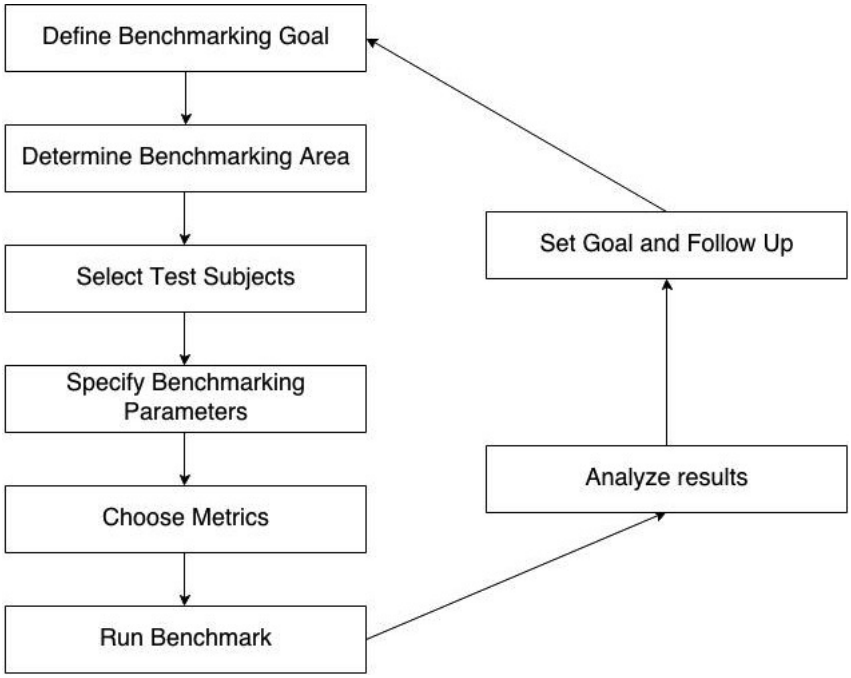

Figure 5.1: A methodology for web performance benchmarking

Define Benchmarking Goal

The most important step is to determine what the objective of performing the benchmark study is. This decision has a strong influence on the whole process, as many of the following steps and parameters depend on how the outcomes of the final analysis are used. Example of benchmarking objectives are as follows:

- Analysis of direct competitors and industry leaders.
  - Set baseline for performance metrics and analyze impact of optimization.

[1] http://www.catchpoint.com

  - Build business cases for strategic and operative decision-making.

- Performance Analysis of third-party services.
  - Competitive analysis prior to a buying-decision.
  - Monitoring and controlling of metrics defined in SLAs or other contractual. agreements

- Identify bottlenecks and observe anomalies (see Chapter 4)

Determine Benchmarking Area

Depending on the overall goal of the benchmark, the area of the benchmarking must be considered carefully. Area in this context refers to the aspect of the application or system:

- Network/DNS – comparison of infrastructure
- Backend Performance – comparison of server processing times
- Frontend Performance – how fast is the site built as soon as the markup is fully loaded. (Load times from third party services loaded through scripts or images are also taken into account)
- Single web request vs. Process/Transaction (multiple successive requests)

Select Test Subjects

The type of test subjects is predetermined by the benchmarking goal. The tests either benchmarks our own application against competition and industry leaders, third party services.

Specify Benchmarking Parameters

Based on the purpose of the benchmark, and benchmark areas, you can determine how and where to test from. For instance, we can choose to benchmark DNS performance from key US states that account for the majority of end users. We might decide to run the tests every 5 minutes, if major changes are planned and detailed analysis is important, or every few hours if the results are merely used for a business case. The specific parameters heavily depend on the decisions that have been made in the previous steps. More generally, these are the parameters that should be considered:

- Sampling frequency: how often to actively probe the target systems
- Geographic locations: where will the test will be conducted
- Duration: determines the length of the observation period of the target systems. Duration, sampling frequency, and number of test subjects determine the number of total data items.
- Synthetic agent:

- Browser emulation: determine what browser(s) to use for emulation
- Connectivity: the traditional option in terms of connectivity are software agents running within a data center with high speed internet connectivity (DSL or similar). Depending on the benchmarking testing infrastructure, and due to the growing importance of mobile devices, some vendors offer agents with 3G or 4G internet connectivity to emulate mobile experience.

All parameters listed are usually constrained by matters of cost and feasibility.

Choose Metrics

The benchmarking area directly maps to the kind of metrics should be chosen in the data collection phase to be analyzed in the following steps: network/DNS area requires basic metrics of HTTP within a request (DNS, Connect, Send). Backend area requires all further request component metrics (Wait, SSL, TTFB, Load, Response time). And the frontend area requires all frontend metrics (DOM Load time, Content load, Render start, Document complete, Time to title, Webpage response).

Run Benchmark/Data Collection

Running the benchmark is the process of data collection over the given duration and sampling frequency, considering all parameters specified in the previous steps of the methodology. Depending on the metrics that are to be collected, we may employ two possible collection strategies as defined in [7]: record storage and data set processing.
In the record storage approach every observation is stored. Further statistics and analysis on the collection of data is delegated to the data analysis step. This approach requires high amounts of storage in the data collection step, as well as computation power for the analysis of the data. It is simple to implement, as it does not require any intermediary steps during collection.
The data set processing approach does not store the observations directly, but continually updates a data set with statistics of interest. These statistics have to be chosen beforehand, since their calculation is integrated in the data collection process. Hence, this approach requires less storage. Some statistics, like the median or percentiles, need specific techniques in order to be dynamically calculated without the presence of all observations [37, 45]. The downside of this approach is that as soon as the benchmark study has been completed, only the predetermined statistics are available for analysis. Whenever possible, it is recommended to store all observations in a repository in order to have all data available for further analysis.

Analyze Results

The collected data should now be properly analyzed and visualized in order to derive useful conclusions and actions. Analysis can range from simple histograms and summary statistics to sophisticated time series analysis.

Set Goal and Follow up

After conducting the analysis, the resulting outcomes should be incorporated into goals and should propose further action. This step is especially important when setting a baseline for improvement of the own application's performance. Benchmarking competition is usually done to assess a status-quo, i.e., where is the application performance compared to competition and industry leaders, and where should it be. In order to drive improvement within a development team, we present a simple process in the following.

Process

A simple set-goal and follow-up process can be summarized as follows:

- Determine specific performance goal (benchmarking): The results of the benchmark determines a rank between competitors. Set a mark within the benchmark to determine your goal (i.e., performance should be in the top 20%)
- Set performance goal in specific metrics (response time, webpage response, etc.)
- Integrate the goal in the development process through visualization
- Follow up on the goal over time

The last to steps in the process concern the development team of the application and are important for reaching the goal. A proper visualization technique in two-dimensional graphs has been developed and is illustrated in Figure 5.2: The goal (as a specific metric) and the percent of users that should reach that goal are fixed within the graph. The x-axis represents time, the y-axis represents the percentage of users that have reached the fixed goal. In the example in Figure 5.2 we can follow how performance has developed over time to reach our set goal.

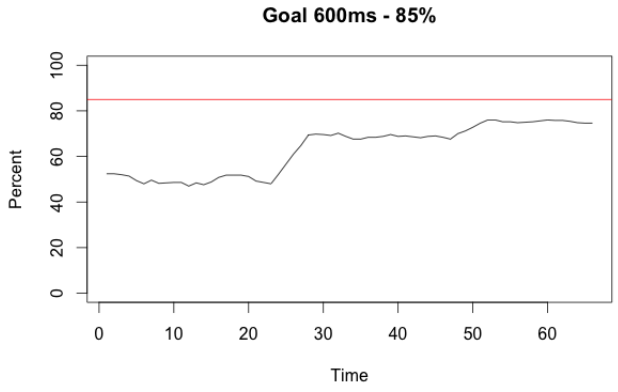

Figure 5.2: Suggested graph for a follow up process - Goal is set at 600ms for 85% of users

5.2 Case Study

During a research internship at Catchpoint Systems, Inc., a case study has been performed employing the previously described methodology. The benchmark in this case study will compare search engine providers and provide a rank of specific metrics in the result section. The following sections will go through all the steps of the methodology and set the proper parameters along the way.

5.2.1 Goal and Area

The goal of this web performance benchmark is to obtain a current state on performance of selected search engines throughout the US for a business case.
In this case study we are interested in *backend performance* and *frontend performance* of the *homepage* of the search engines (i.e., a single web request).

5.2.2 Test subjects

The elected test subjects are the most popular search engines: AOL[2], Ask[3], Bing[4], Google[5], Yahoo[6]

5.2.3 Parameters

The following benchmarking parameters have been set:

- Sampling frequency: Every 8 hours from *each* geographic location
- Geographic locations: 45 synthetic agent nodes throughout the US and Canada[7]
- Duration: 1 month
- Synthetic agent:
  - Emulated Browser: Internet Explorer 9
  - Connectivity: DSL Connectivity within data centers

5.2.4 Metrics

The chosen metrics for the defined area of frontend and backend performance are: *response time*, *render start*, and *webpage response*. We recall the definition of these metrics:

[2] http://www.aol.com
[3] http://www.ask.com
[4] http://www.bing.com
[5] http://www.google.com
[6] http://www.yahoo.com
[7] The exact node locations can be found up in the appendix

- Response time: Network time it takes to get the base HTML from the web server, starting with the initial request and ending at the last received response packet.
- Render start: Time it takes the browser to start rendering objects on the page.
- Webpage response: Time it takes to load every element of the webpage. The whole application is then fully loaded

5.2.5 Data Collection and Analysis

The defined metrics are to be observed over the defined time span and, should all be collected within a data repository. The analysis will be narrowed to the median values of the desired metrics, as the primary goal of the benchmark is to calculate a rank within the search engine industry.

5.2.6 Results

The benchmark collected 25549 observations over 45 synthetic agent nodes throughout the US and Canada within 1 month of 5 search engines. The results of the web performance benchmark have been summarized as *ranks* on how these 5 sites have compared in the metrics response time, render start time, and webpage response. The results are presented in the Figures 5.3, 5.4, 5.5, as well as in Table 5.1.
A brief interpretation of the results is outlined in the following.

Conclusion

Google ranks first across all metrics and can therefore be declared the 'winner' of this web performance benchmark. The ranks are pretty straight-forward and usually follow the same order across all metrics, with one interesting exception. At the metric *render start time* Yahoo (who ranks last in all other metrics) now ranks third after Google and Bing.
Render start time is important, as it gives the user visual feedback that the site is processing and the site is still responsive. From this we can conclude that Yahoo's server resources load faster than those of Ask and AOL. When looking further at the results for *webpage response*, we can see that Yahoo then loses to all the competitors in overall performance. This means that Yahoo, although it had fast server responses, loses out in frontend performance. If we look at the specific results it takes Yahoo 2833ms to render all objects on the homepage, from the moment the HTML source was already loaded.
From this information, we can take away that Yahoo should restructure their frontend in order to gain better overall performance. Ask and AOL should observe their server performance more closely to identify potential bottlenecks or other ways to improve.

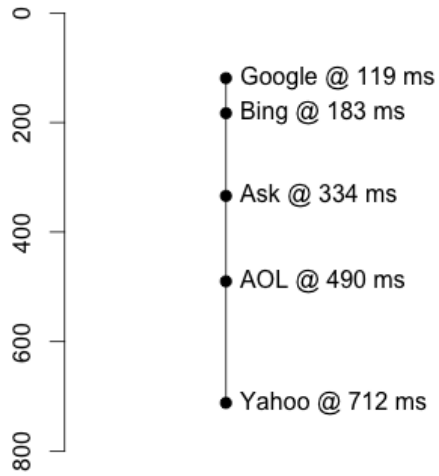

Figure 5.3: Response times ranking

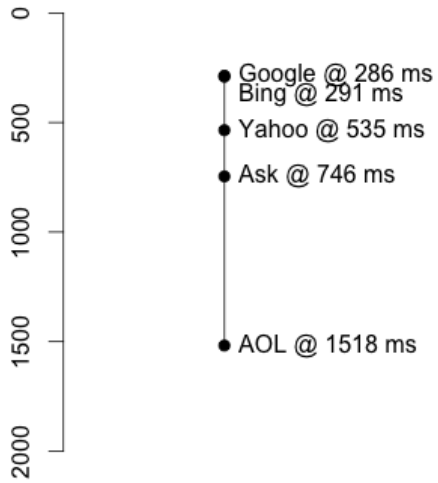

Figure 5.4: Render start time ranking

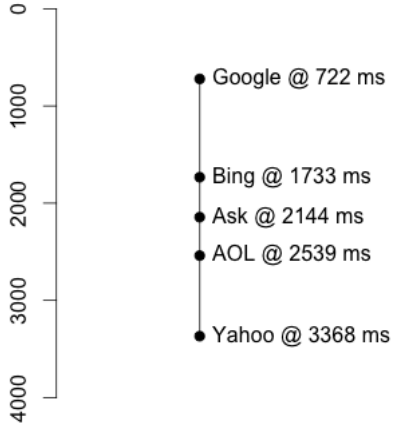

Figure 5.5: Webpage response ranking

	Response time	Render start ime	Webpage response
Google	119ms	286ms	722ms
Bing	183ms	291ms	1733ms
Ask	334ms	746ms	2144ms
AOL	490ms	1518ms	2539ms
Yahoo	712ms	535ms	3368ms

Table 5.1: Results of median response time, render start time, and webpage response time

CHAPTER 6

Conclusion

This chapter concludes this thesis with a brief summary of the methods and results that have been presented and discussed. Moreover, the research questions, that have been introduced in Section 1.2, will be revisited to answer how these questions were addressed throughout the thesis. Finally, possible future work that we were not able to sufficiently address is summarized.

6.1 Summary

This thesis starts off with providing motivation for the importance of properly managing web performance in large scale web applications. It is followed by summarizing background information on web performance monitoring, as well as statistical methods employed in performance analysis. Furthermore, it has presented research work related to the topics and methods of this thesis.

A taxonomy of root-causes in performance regressions in web systems has been introduced, which was further used to construct scenarios to simulate issues in web performance within an experimental setup. In a series of simulations, we measured how performance metrics develop over time and presented the results. Furthermore, we provided analysis and interpretation of the results.

A general methodology for web performance benchmarking based on synthetic monitoring of target systems has been outlined. A case study on benchmarking search engine providers was performed by applying the presented methodology. Ultimately, the results of the benchmark were presented and discussed.

6.2 Research Questions Revisited

We recapitulate the results of this thesis by returning to the research questions introduced in Section 1.2, and summarize the work that has been accomplished to address them.

How do performance issues manifest in performance data gathered through external synthetic monitoring? How can these changes in performance be observed, both visually and through statistical methods and algorithms?

The first research question was addressed in Chapter 4. The first step was to characterize what kind of performance issues are we able to observe through external synthetic monitoring. Therefore, we developed a taxonomy of root-causes in performance regressions in section 4.1. The taxonomy serves as the basis to construct scenarios within a developed simulation model. In the simulation model, we introduced a formal model for the specification of scenarios in Section 4.2.2. The second part of the research question was addressed in the evaluation of the results. We apply exploratory visual analysis and statistical changepoint analysis to the results. Furthermore, we provide interpretation of the results in order to understand how the simulated performance changes can be properly observed.

How do we determine a performance goal? When a certain performance goal is determined, how can the progress of achieving this goal be actively tracked and communicated?

The second research question was addressed in Chapter 5. A possible way to determine a performance goal is to establish a baseline metric that is obtained through web performance benchmarking of immediate competition. We develop a more general approach to web performance benchmarking in Section 5.1, which can also be applied to benchmark and compare third party services. In the presented method, we introduce a feedback loop that describes how the progress of a determined performance goal can be communicated. We suggest a specific form of visualization, which tracks the progress of a performance goal over time.

6.3 Future Work

Following the work presented in this thesis, there are possible improvements and further work we were not able to address sufficiently:

- Performance data gathered through external synthetic monitoring only allows for a black box view of the system and is often not sufficient for in-depth root-cause analysis of performance issues. Combining data from external monitoring and internal monitoring in order to automate or assist in root-cause analysis and correlation of issues is a possible approach that should be considered.
- The simulation and analysis in this thesis is limited to performance issues on the server. Further work might include extending the taxonomy of root-causes and simulation scenarios to also represent frontend performance issues. Furthermore, the presented statistical methods could be applied and analyzed on frontend metrics as well.
- For statistical changepoint analysis, this thesis employed the Binary Segmentation algorithm. An interesting addition would be an evaluation on different algorithms for changepoint analysis. A comprehensive review on further changepoint algorithms can be found in [74].

APPENDIX A

Acronyms

APM Application Performance and Resource Management

DNS Domain Name System

DOM Document Object Model

ISP Internet Service Provider

OED Oxford English Dictionary

QoE Quality of Experience

QoS Quality of Service

pdf Probability density function

SLA Service Level Agreements

SPC Statistical Process Control

SPEC Standard Performance Evaluation Corporation

SUT System Under Test

TCP Transfer Control Protocol

TPC Transaction Processing Performance Council

APPENDIX B

List of Synthetic Agent Locations

Location	Data Center
Las Vegas	Level 3
New York	Level 3
New York	VZN
Toronto CA	Bell Canada
Denver	Level 3
Washington DC	NTT
Washington DC	AboveNet
Washington DC	Cogent
Los Angeles	GLC
Denver	AboveNet
Atlanta	Cogent
Atlanta	Level 3
Atlanta	NTT
Los Angeles	Level 3
Atlanta	AboveNet
Denver	XO
Los Angeles	VZN
Denver	TWTC
Chicago	Level 3
Phoenix	AboveNet
Phoenix	Internap
San Francisco	TWTC
Seattle	GTT

Location	Data Center
Chicago	AboveNet
Seattle	Level 3
San Francisco	NTT
Chicago	NTT
Miami	GTT
Miami	GLC
Miami	Level 3
San Francisco	VZN
San Francisco	Level 3
Washington DC	Level 3
New York	Savvis
Los Angeles	Savvis
Seattle	Savvis
Dallas	Cogent
Dallas	GLC
Dallas	Telx
Dallas	Level 3
New York	GTT
Montreal	Basic
Vancouver	Level 3
Toronto	Bell Canada
Montreal	Bell Canada
Calgary	Bell Canada

Bibliography

[1] *The Benchmarking management guide*. Productivity Press, Cambridge, Mass, 1993.

[2] ecommerce web site performance today, an updated look at consumer reaction to a poor online shopping experience, August 2009.

[3] Marcos K Aguilera, Jeffrey C Mogul, Janet L Wiener, Patrick Reynolds, and Athicha Muthitacharoen. Performance debugging for distributed systems of black boxes. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 74–89. ACM, 2003.

[4] John Aitchison and James Alan Calvert Brown. The lognormal distribution. 1963.

[5] Akamai. The impact of web performance on e-retail success, 2004.

[6] Mauro Andreolini, Valeria Cardellini, and Michele Colajanni. Benchmarking models and tools for distributed web-server systems. In *Performance Evaluation of Complex Systems: Techniques and Tools*, pages 208–235. Springer, 2002.

[7] Mauro Andreolini, Valeria Cardellini, and Michele Colajanni. Benchmarking models and tools for distributed web-server systems. In *Performance Evaluation of Complex Systems: Techniques and Tools*, pages 208–235. Springer, 2002.

[8] N Balakrishnan. *Order statistics : theory & methods*. Elsevier, Amsterdam New York, 1998.

[9] Sandjai Bhulai, Swaminathan Sivasubramanian, Rob Van Der Mei, and Maarten Van Steen. Modeling and predicting end-to-end response times in multi-tier internet applications. In *Managing Traffic Performance in Converged Networks*, pages 519–532. Springer, 2007.

[10] Leszek Borzemski. The experimental design for data mining to discover web performance issues in a wide area network. *Cybernetics and Systems*, 41(1):31–45, 2010.

[11] Leszek Borzemski and Maciej Drwal. Time series forecasting of web performance data monitored by mwing multiagent distributed system. In *ICCCI (1)*, pages 20–29, 2010.

[12] Leszek Borzemski and Anna Kaminska-Chuchmala. Knowledge discovery about web performance with geostatistical turning bands method. In *KES (2)*, pages 581–590, 2011.

[13] Leszek Borzemski and Anna Kaminska-Chuchmala. Knowledge engineering relating to spatial web performance forecasting with sequential gaussian simulation method. In *KES*, pages 1439–1448, 2012.

[14] Leszek Borzemski and Anna Kaminska-Chuchmala. Spatio-temporal web performance forecasting with sequential gaussian simulation method. In *CN*, pages 111–119, 2012.

[15] Leszek Borzemski and Anna Kaminska-Chuchmala. Distributed web systems performance forecasting using turning bands method. *IEEE Trans. Industrial Informatics*, 9(1):254–261, 2013.

[16] Leszek Borzemski, Marta Kliber, and Ziemowit Nowak. Using data mining algorithms in web performance prediction. *Cybernetics and Systems*, 40(2):176–187, 2009.

[17] Anna Bouch, Allan Kuchinsky, and Nina Bhatti. Quality is in the eye of the beholder: meeting users' requirements for internet quality of service. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 297–304. ACM, 2000.

[18] George Box. *Time series analysis : forecasting and control*. John Wiley, Hoboken, N.J, 2008.

[19] Christian Callegari, Sandrine Vaton, and Michele Pagano. A new statistical approach to network anomaly detection. In *Performance Evaluation of Computer and Telecommunication Systems, 2008. SPECTS 2008. International Symposium on*, pages 441–447. IEEE, 2008.

[20] Catchpoint. Anatomy of an http transaction. catchpoint anatomy. Accessed December 26, 2013.

[21] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[22] Jie Chen and Arjun Gupta. *Parametric statistical change point analysis with applications to genetics, medicine, and finance*. Boston, 2012.

[23] Mike Y Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 595–604. IEEE, 2002.

[24] Ludmila Cherkasova, Kivanc Ozonat, Ningfang Mi, Julie Symons, and Evgenia Smirni. Automated anomaly detection and performance modeling of enterprise applications. *ACM Transactions on Computer Systems (TOCS)*, 27(3):6, 2009.

[25] Thiam Kian Chiew. *Web page performance analysis*. PhD thesis, University of Glasgow, 2009.

[26] David M Ciemiewicz. What do you mean?-revisiting statistics for web response time measurements. 2001.

[27] Ira Cohen, Steve Zhang, Moises Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. Capturing, indexing, clustering, and retrieving system history. In *ACM SIGOPS Operating Systems Review*, volume 39, pages 105–118. ACM, 2005.

[28] SD Committee et al. Specweb. Technical report, Tech. Rep., April 1999, http://www. specbench. org/osg/web.

[29] Compuware. Ten best practices for benchmarking web and mobile site performance. `http://www.compuware.com/content/dam/compuware/apm/assets/whitepapers/WP_Ten_Best_Practices_Benchmarking.pdf`, 2013. Accessed: 2014-01-16.

[30] Alistair Croll and Sean Power. *Complete Web Monitoring - Watching your visitors, performance, communities and competitors.* O'Reilly, 2009.

[31] Idris A Eckley, Paul Fearnhead, and Rebecca Killick. Analysis of changepoint models. 2011.

[32] A W EDWARDS and L L CAVALLI-SFORZA. A METHOD FOR CLUSTER ANALYSIS. *Biometrics*, 21:362–375, 1965.

[33] Dan Farber. Googleś marissa mayer: Speed wins. `http://www.zdnet.com/blog/btl/googles-marissa-mayer-speed-wins/3925`, 2006. Accessed: 2014-01-16.

[34] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*. The Internet Society, June 1999.

[35] Piotr Fryzlewicz. Wild binary segmentation for multiple change-point detection. 2012.

[36] Everette S Gardner Jr. Exponential smoothing: The state of the art—part ii. *International Journal of Forecasting*, 22(4):637–666, 2006.

[37] Anna C Gilbert, Yannis Kotidis, S Muthukrishnan, and Martin J Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 454–465. VLDB Endowment, 2002.

[38] David Gourley. *HTTP : the definitive guide*. O'Reilly, Beijing Sebastopol, CA, 2002.

[39] Ilya Grigorik. *High-performance browser networking*. O'Reilly, Beijing Sebastopol, CA, 2013.

[40] Neil Gunther. *Analyzing computer system performance with Perl:PDQ*. Springer, New York, 2011.

[41] Russell G Heikes, Douglas C Montgomery, and Ronald L Rardin. Using common random numbers in simulation experiments—an approach to statistical analysis. *Simulation*, 27(3):81–85, 1976.

[42] Robert Hogg. *Introduction to mathematical statistics*. Prentice Hall, Englewood Cliffs, N.J, 1995.

[43] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

[44] Mukesh Jain. Performance testing and improvements using six sigma. In *28th Annual Proceeding of the PNSQC Conference*, 2010.

[45] Raj Jain and Imrich Chlamtac. The p 2 algorithm for dynamic calculation of quantiles and histograms without storing observations. *Communications of the ACM*, 28(10):1076–1085, 1985.

[46] Rebecca Killick and Idris A. Eckley. changepoint: an R package for changepoint analysis. *R package version 0.5*, 2011.

[47] Rebecca Killick, Paul Fearnhead, and IA Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, 2012.

[48] Andrew King. *Speed up your site : Web site optimization*. New Riders, Indianapolis, Ind, 2003.

[49] Jonathan Klein. Email Interview, September 2013.

[50] Ron Kohavi, Randal M Henne, and Dan Sommerfield. Practical guide to controlled experiments on the web: listen to your customers not to the hippo. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 959–967. ACM, 2007.

[51] Umesh Krishnaswamy and Isaac D Scherson. A framework for computer performance evaluation using benchmark sets. *Computers, IEEE Transactions on*, 49(12):1325–1338, 2000.

[52] Zhen Liu, Nicolas Niclausse, Cesar Jalpa-Villanueva, and Sylvain Barbier. Traffic Model and Performance Evaluation of Web Servers. Technical Report RR-3840, INRIA, December 1999.

[53] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[54] Daniel A Menascé. Tpc-w: A benchmark for e-commerce. *Internet Computing, IEEE*, 6(3):83–87, 2002.

[55] Carolina Milanesi, Lillian Tay, Roberta Cozza, Ranjit Atwal, Tuong Huy Nguyen, Tracy Tsai, Annette Zimmermann, and CK Lu. Forecast: Devices by operating system and user type, worldwide, 2010-2017, 1q13 update. *Dataquest Market Insights: Computing Hardware Worldwide*, (5):1–6, April 2013.

[56] Amitava Mitra. *Fundamentals of quality control and improvement*. Wiley. com, 2012.

[57] Theophano Mitsa. *Temporal data mining*. Chapman & Hall/CRC, 2010.

[58] Paul. Mockapetris. *RFC 1034: Domain names-concepts and facilities*. The Internet Society, November 1987.

[59] Thanh HD Nguyen, Bram Adams, Zhen Ming Jiang, Ahmed E Hassan, Mohamed Nasser, and Parminder Flora. Automated detection of performance regressions using statistical process control techniques. In *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering*, pages 299–310. ACM, 2012.

[60] OED Online. metric, n., Oxford English Dictionary Online, 2nd edition. http://www.oed.com, January 2014.

[61] Adam B Olshen, E S Venkatraman, Robert Lucito, and Michael Wigler. Circular binary segmentation for the analysis of array-based DNA copy number data. *Biostatistics (Oxford, England)*, 5:557–572, 2004.

[62] Hoang Pham. *System software reliability*. Springer, Berlin London, 2006.

[63] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[64] Radware. State of the union: Ecommerce page speed & web performance. `http://www.radware.com/stateoftheunion-fall2013/`, 2013. Accessed: 2014-01-16.

[65] Amit P Sawant, Matti Vanninen, and Christopher G Healey. Perfviz: A visualization tool for analyzing, exploring, and comparing storage controller performance data. In *Electronic Imaging 2007*, pages 649507–649507. International Society for Optics and Photonics, 2007.

[66] A J Scott and M Knott. A Cluster Analysis Method for Grouping Means in the Analysis of Variance. *Biometrics*, 30:507–512, 1974.

[67] Luca Scrucca. qcc: an r package for quality control charting and statistical process control. *R News*, 4/1:11–17, 2004.

[68] Howard J. Seltman. *Experimental Design and Analysis*. Pittsburgh, PA, 2013.

[69] Ashish Sen and Muni S. Srivastava. On Tests for Detecting Change in Mean, 1975.

[70] Walter A Shewhart. Economic control of quality of manufactured product. *New York*, 501, 1931.

[71] Robert Shumway. *Time series analysis and its applications with R examples*. Springer, New York, 2011.

[72] Yingying Chen Ratul Mahajan Baskar Sridharan and Zhi-Li Zhang. A provider-side view of web search response time. 2013.

[73] Abe Stanway. Skyline, open source project. `https://github.com/etsy/skyline`, 2013. Accessed: 2014-01-16.

[74] Amanda Strong. A review of anomaly detection with focus on changepoint detection, 2012.

[75] Drit Suljoti. Private Interview, August 2013.

[76] Nicole Sullivan. Designing fast websites. `http://www.slideshare.net/stubbornella/designing-fast-websites-presentation`, 2009. Accessed: 2014-01-16.

[77] Andrew Tanenbaum. *Distributed systems : principles and paradigms*. Pearson Prentice Hall, Upper Saddle River, NJ, 2007.

[78] Reinhold Weicker. Benchmarking. In *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures*, pages 179–207, London, UK, UK, 2002. Springer-Verlag.

[79] Yottaa. 9 tips to benchmark your web performance. `http://www.yottaa.com/9-tips-to-benchmark-your-web-performance-ebook`, 2013. Accessed: 2014-01-16.