Journal of Web Engineering, Vol. 14, No. 5&6 (2015) 414–442
© Rinton Press

IDENTIFYING WEB PERFORMANCE DEGRADATIONS THROUGH SYNTHETIC AND REAL-USER MONITORING

JÜRGEN CITO[1] DEVAN GOTOWKA[2] PHILIPP LEITNER[1] RYAN PELETTE[2]
DRITAN SULJOTI[2] SCHAHRAM DUSTDAR[3]

[1] *s.e.a.l. – software evolution & architecture lab, University of Zurich, Switzerland*
{cito, leitner}@ifi.uzh.ch

[2] *Catchpoint Systems, Inc., New York, USA*
{devan, ryan, drit}@catchpoint.com

[3] *Distributed Systems Group, Vienna University of Technology, Austria*
dustdar@dsg.tuwien.ac.at

The large scale of the Internet has offered unique economic opportunities, that in turn introduce overwhelming challenges for development and operations to provide reliable and fast services in order to meet the high demands on the performance of online services. In this paper, we investigate how performance engineers can identify three different classes of externally-visible performance problems (global delays, partial delays, periodic delays) from concrete traces. We develop a simulation model based on a taxonomy of root causes in server performance degradation. Within an experimental setup, we obtain results through synthetic monitoring of a target Web service, and observe changes in Web performance over time through exploratory visual analysis and changepoint detection. We extend our analysis and apply our methods to real-user monitoring (RUM) data. In a use case study, we discuss how our underlying model can be applied to real performance data gathered from a multinational, high-traffic website in the financial sector. Finally, we interpret our findings and discuss various challenges and pitfalls.

Keywords: Web services; performance engineering; application performance monitoring; changepoint analysis

1 Introduction

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. Research has shown that nowadays 47% of online consumers expect load times of *two seconds or less* [7, 21, 29]. 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 (Quality of Service) a challenge for large scale Internet applications. Generally, providing a consistent QoS requires continually collecting data on Web performance on the Web service provider side, in order to observe and track changes in desired metrics, e.g., service response time. Reasons to observe these changes are different in their nature, and range from detecting anomalies, identifying patterns, ensuring service reliability, measuring performance changes after new software releases, or discovering performance degradation.

Early detection and resolution of root causes of performance degradations can be achieved through monitoring of various components of the system. Monitoring can be classified as either *active* or *passive* monitoring, and, orthogonally, as *external* or *internal*. In active monitoring, monitoring agents are actively trying to connect to the target system in order to collect performance data, whether the system is accessed by real end-users or not. Passive monitoring, on the other hand, only collects measurements if the system is actively used. Internal and external monitoring differentiate in whether the measurements are obtained in systems within the organization's data center or through end-to-end monitoring over the network outside the data center. This has ramifications in terms of what the level of detail of monitoring data that is available. In our experiments, we make use of active, external monitoring, which provides a way of capturing an end user perspective and enables the detection of issues before they affect real users [24]. In this paper we actively focus on external monitoring and omit the notion of internal monitoring in our experiments, because for many use cases (e.g., monitoring solution providers) internal data is not always available. Thus, we are presenting an approach to identify performance issues in the absence of internal monitoring data.

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

This paper is an invited extension of work [13] that originally appeared in Proceedings of the 14th International Conference on Web Engineering, ICWE'14. In the original research paper, we investigated how changepoint analysis can be applied to web performance data based on traces gathered through simulation. In this extension, we expand our analysis and apply the changepoint detection method to real-user monitoring data and discuss how our underlying model can be applied to real performance data. Furthermore, we report on a statistical phenomenon - Simpson's paradox - that we detected within our real-life use case.

The main contributions of this paper are now two-fold: (1) A model for understanding performance data via analyzing how common underlying root causes of web performance issues manifest themselves in data gathered through *active, external* monitoring, and (2) how this model translates to performance data gathered through *passive, real-user* monitoring.
The rest of the paper is structured as follows: Section 2 provides an overview on the basics of web performance monitoring. It will also give an introduction to the statistical methods used throughout the paper. In Section 3, we introduce a taxonomy of root causes in server performance degradations, which serves as the basis for our experiments. In Section 4, we describe the methods and steps we take to obtain our results and explain 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 based on exploratory visual analysis and statistical changepoint analysis. In Section 5, we use these insights and apply our underlying model and statistical method to real-user data collected through *passive, real-user monitoring.* Again, we interpret

the results and compare exploratory visual analysis with the results obtained by statistical analysis. Furthermore, we present a case where we have observed Simpson's Paradox within the collected performance data. Section 6 contrasts our analysis on synthetic monitoring with real-user monitoring and discusses the respective benefits and pitfalls. Section 9 puts our work in the context of the larger scientific community. Finally, we conclude the paper in Section 10 with an outlook on future work.

2 Background

The following sections will briefly introduce monitoring and analysis techniques employed in this paper.

2.1 Web Performance Monitoring

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.

2.1.1 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 monitoring service. In the following, we will describe the basics of a web transaction over the HTTP protocol. This section is largely based on [16, 17, 26].

HTTP is an application layer protocol built on top of the TCP protocol. TCP 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.

HTTP Transaction We consider a simple HTTP transaction, where the client (either a web browser or another application in case of a web service) issues a single request for HTTP content to the *web server* hosting the web service or web application. Figure 1 depicts the workflow of such a HTTP transaction. In the following, we describe the steps taken in this workflow from starting the request to receiving the response.

1. DNS Lookup: The DNS lookup resolves a domain name for the request. The client sends a DNS query to the local ISP DNS server.[a] The DNS server responds with the IP address

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

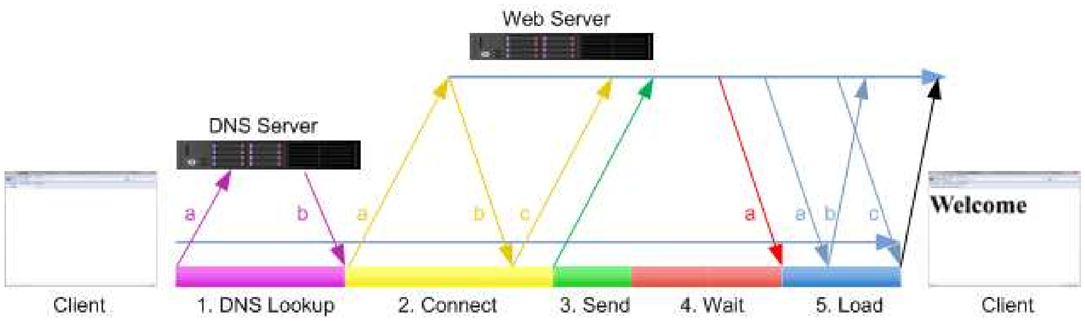


Fig. 1: Anatomy of a simple HTTP transaction

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.

These are the basic steps involved in a simple HTTP transaction. The HTTP 1.1 protocol further supports *persistent* and *parallel* transactions [16].

HTTP/2. HTTP/2 is the next generation of the HTTP network protocol with a focus on performance. It is not supposed to be a full rewrite, but it rather uses the same APIs and semantics (e.g., status codes, HTTP methods) to represent the protocol.
The foundation for these improvements is SPDY [15], an application-layer web protocol, which started out as an experimental protocol designed by Google that has been quickly adopted by major browsers,[b] as well as middleware and server vendors. The overall goal is to reduce

[b] At the time of this writing: Chrome, Firefox, Opera, Internet Explorer 11, Safari, and Amazon Silk

end-user perceived latency by providing mechanisms such as *multiplexing*, *header compression*, *server push/hint*, and *content prioritization* [15].

Prior to HTTP/2 a given connection could only handle a single request at a given time. Multiplexing allows for multiple requests to exist on the same connection simultaneously. This increases parallelism and therefore the efficiency is increased as well. Header compression reduces the total bytes for each request, which improves performance since less TCP packets are sent on the wire. Server push/hint improves performance because the server indicates to the client what resources are required (hint) or simply sends the resources directly without the client being required to parse HTML upon receiving it in order to issue requests as in HTTP/1.X. Content prioritization allows the client to specify to the server priority for important resources that should load first. This reduces contention by effectively throttling resources that are less important thereby improving user-perceived load time if used correctly.

2.1.2 Synthetic Monitoring.

Due to the complexity of modern distributed systems, the underlying infrastructure 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 the 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. It is also referred to as *active monitoring* or *active probing* [12], 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. 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.

2.1.3 Real-User Monitoring.

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. Therefore, if end users cannot access the application, or there is a low/no traffic period, or not enough data captured, performance and availability issues might go unnoticed.

RUM 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. Additionally, RUM can also capture device information, operating system, browser information and geographic origin.

2.2 Statistical Analysis

2.2.1 Changepoint Analysis.

Changepoint analysis deals with the identification of points within a time series where 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 distribution 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. The detection of these points in time generally takes the form of hypothesis testing. The null hypothesis, H_0, represents no changepoint and the alternative hypothesis, H_1, represents existing changepoints.
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 1.

$$H_0 : F_1 = F_2 = ... = F_n \tag{1}$$

where $F_1, F_2, ..., F_n$ are probability distribution functions associated with the random variables of our segments. The alternative hypothesis would then be defined as in Equation 2.

$$H_1 : F_1 = ... = F_{k_1} \neq F_{k_1+1} = ... = F_{k_m} \neq F_{k_m+1} = ... = F_n \tag{2}$$

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. 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.

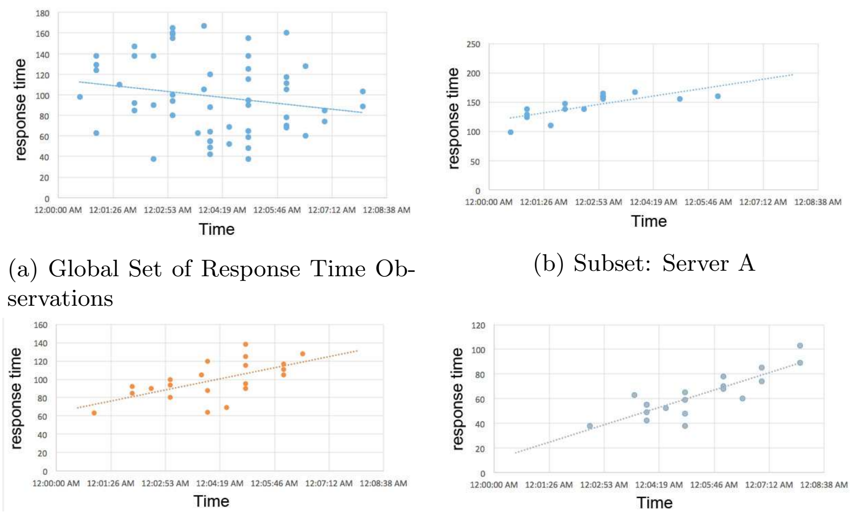
For further information, we refer to the original papers [8,20], as well as to work providing a more comprehensive review [31].

In Section 4 we apply changepoint analysis to web performance data to detect fundamental distribution shifts in our data that reflect longer standing performance degradations.

2.2.2 Simpson's Paradox.

Analyzing data made up of distinct subsets can lead to a statistical phenomenon known as Simpson's Paradox [32], in which a trend detected to unique sets of data (i.e., a subpopulation) is actually opposite the trend detected when the same analysis is applied to the combined sets of the data.

In Figure 2 we briefly demonstrate this statistical phenomenon in a web performance context. We plot response times over time, as well as a trend line calculated through linear regression. In the global data set containing all metric observations in Figure 2a there appears to be a negative trend, as indicated by the trend line. Looking at each server individually in Figures 2b, 2c, 2d, response times actually slow down (trend positively) in each server group.



(a) Global Set of Response Time Observations

(b) Subset: Server A

(c) Subset: Server B

(d) Subset: Server C

Fig. 2: Simpson's Paradox illustrated in four simple figures

This response time slowdown within each group is the opposite the trend detected across the combined set of groups in Figure 2a.

In Section 5 we discuss a case study in which Simpson's paradox can be observed in web performance data.

3 Root Causes of Server Performance Degradation

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 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. In case of extreme oversaturation, the system stops reacting entirely. 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 of response times x_t and x_{t+1} where $\lambda = |x_t - x_{t+1}| > c$ and 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.

The underlying causes of performance degradations in Web application and Web service backends are diverse. They differ significantly in the way they manifest themselves in performance data gathered through active, external monitoring. We propose a simple taxonomy of root causes in Web performance degradation. First, we assign a possible root cause (e.g., Slow I/O, High CPU utilization, Log rotation, as seen in Figure 3) to one of the three main performance degradations effects, which can be determined through external monitoring: *global delay*, *partial delay*, and *periodic delay*. Further classifications and proper assignment to the main effects in the taxonomy have been derived together with domain experts in the area of Web performance monitoring and optimization. This classification supports providers of application monitoring solutions to guide their alerting efforts.

In the following, we provide a brief explanation of the general causes (global, partial, and periodic) of performance delays in computer systems. We then classify the main effects of our taxonomy by grouping the root causes by the distinguishable effect they have. The taxonomy is depicted in Figure 3, and explained in more detail the following.

Note that: 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.

3.1 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. We distinguish between global delays that are caused through resource contention and code or configuration issues. Global delays caused by resource contention include, for instance, delays due to a bottleneck in disk I/O. In this case, the flow of data from the disk to the application (e.g., via a database query) is contended. This means the query takes longer to perform and return data which causes an overall performance delay. Global delays caused by problems in the application code can for instance be caused by the introduction of logical units or algorithms with high computational complexity in the backend service. Alternatively, such global delays may be caused by overzealous synchronization in the application code, which may even lead to temporary service outages when a deadlock situation cannot be immediately resolved by the underlying operating system or virtual machine.

Lastly, global delays can also be caused by misconfiguration on the responding server. An example of this is a *misconfigured cache* on a server. Traditionally, caching is used to optimize performance by storing frequently accessed data locally rather than on a (remote) disk. This helps reduce the overhead of repeated requests and makes accessing the most frequently used data much faster. There are other benefits as well, such as reducing CPU, disk and memory usage each time the request hits the responding server. Caching the most frequently requested data also frees up the responding server to handle other requests from more users. However, if the cache is misconfigured and the server is trying to access something from cache that is in fact not in cache, then there is a cache miss. This will cause the server to stall or not return the requested data. Since the most frequently accessed data is the data that is cached, misconfigurations will impact a majority of the users, causing a global delay.

3.2 Periodic Delay

Periodic delays are global delays that are not continuous, but happen, for instance, a few times a day. A periodic delay is mostly not induced through a new deployment or release,

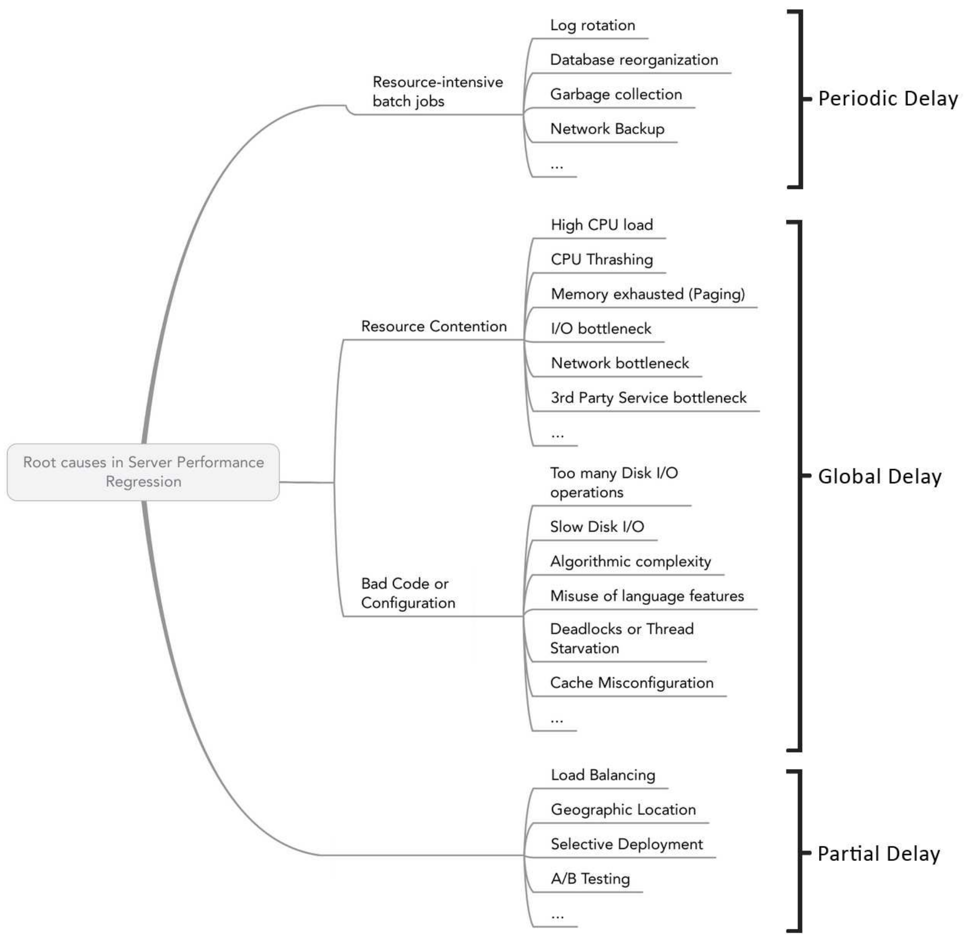


Fig. 3: Taxonomy of Root Causes in Server Performance Degradation. *Causes* (on the left side) lead to *effects* (periodic, global, partial delay) on the right side

but rather through a background process causing the system to use an increased amount of resources. One practical example of such a background job is log rotation. Log rotation is an automated process in server systems administration, where log files that exhibit certain characteristics are archived. The process is usually configured to run as a periodic cronjob to be fully automated. Log rotating often includes archiving and transferring large text files, and, hence, can oversaturate the backend system for a short period of time.

3.3 Partial Delay

Partial delays are global delays that occur only for a subset of all requests, e.g., for a subset of all customers of the Web service. This situation can occur if the application employs a form of redundancy, for instance, load balancing or content distribution networks (CDN). In such scenarios, any problems that lead to global delays can potentially be inflicting only one or a subset of all backend servers, hence delaying only those requests that happen to be handled by one of the affected backends. Partial delays are interesting, as they are hard to detect (especially if the number of affected backends is small).

4 Identifying Performance Degradations through Synthetic Monitoring

After introducing externally visible classes of effects of performance degradation (global, partial, periodic), we want to identify characteristics in performance data associated with each class. Furthermore, we want to present statistical methods that are well suited for identifying such changes. Automatic classification of these effects support application monitoring solutions to guide their alerting. Our approach is to generate realistic performance traces for each class through a testbed Web service, which we have modified in order to be able to inject specific *performance degradation scenarios* associated with each class. We collect data through active monitoring as described in Section 4.1. The specific scenarios we used are described and defined in Section 4.2. Afterwards, we apply the methods described in Section 4.3.1 to the traces we generated, in order to be able to make general statements about how these methods are able to detect root causes of performance degradation.

4.1 Simulation Design

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*, and *Dynamic Web Service Component*. Synthetic agents send out HTTP GET requests every n minutes from m agents and collect response times. In the simulation, we sample every 1 minute resulting in 1 new observation of our system every minute. Each observation is stored in a database with the corresponding timestamp.

The simulation design and its communication channels are depicted in Figure 4. We consider a system with this architecture where requests incoming from the synthetic nodes are governed by a stochastic process $\{Y(t), t \in T\}$, with T being an index set representing time.

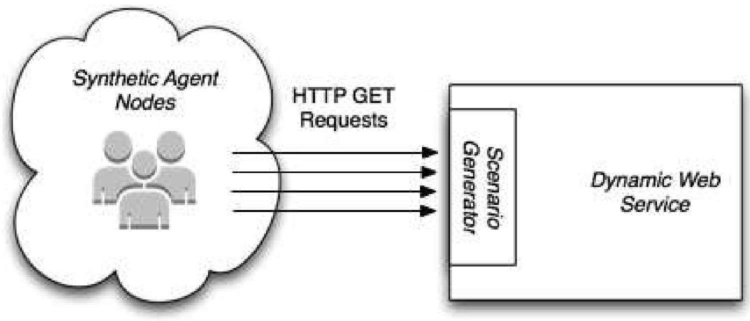


Fig. 4: Simulation Design

4.1.1 Synthetic Agent Nodes

We gather data from the *Dynamic Web Service Component* via active, periodic monitoring through *Synthetic Agent Nodes*. 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: (1) URL of the Web service that should be tested; (2) sampling interval, e.g., every n minutes; (3) test duration, i.e., how many sample requests to issue in each run of the test agent.

4.1.2 Scenario Generation Component

As only the main effects can be identified through synthetic monitoring, changes following the primary notions of *global delays*, *partial delays*, and *periodic delays* (see Section 3) are injected in 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 test 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. The Scenario Generation Component also needs to take into account possible geographic distribution of the agents, as well as load balancing mechanisms. In the following, we introduce a formal model for defining scenarios that reflects these notions that can be used to formally describe scenarios.

We consider a given set of parameters to compose a complete scenario within our simulation model. Within a scenario, we need to be able to specify how performance metrics (i.e., response times) develop over time, as well as synthetic agents that are actively probing our target system.

- 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 3).

$$D : T \mapsto \mathcal{X} \tag{3}$$

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

- On the top-level, we define a scenario $\mathcal{S}$ that describes how the target system that is observed by each synthetic agent $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ develops over time. Each agent observes a development in performance $D_i \in \mathcal{D}$, with $\mathcal{D}$ being the set of all possible developments (Equation 4).

$$\mathcal{S} : \mathcal{A} \mapsto \mathcal{D} \tag{4}$$

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 5.

$$[X_0, t_1, X_1, ..., t_n, X_n](t) = \begin{cases} X_0 & 0 \leq t < t_1 \\ X_1 & t_1 \leq t < t_2 \\ \vdots & \\ X_n & t_n \leq t \end{cases} \tag{5}$$

  This allows us to easily define developments X_i in terms of time spans $t_{i+1} - t_i \geq 0$ 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 6.

$$[X_0, n, X_1, p]^*(t) = \begin{cases} X_1 & \text{for } kp + n \leq t < (k+1)p \\ X_0 & \text{otherwise} \end{cases} \tag{6}$$

  where $k \geq 0$.

Figure 5 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".

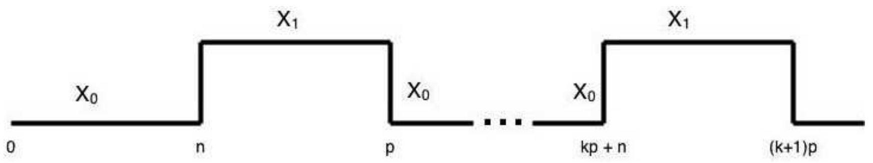


Fig. 5: Depiction of the periodic development scheme defined in Equation 6

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 performance at certain points in time in a declarative way, as well as define changes in periodic time intervals that result in changes in response times for a specific amount of time.

4.1.3 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 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.

4.2 Simulation Scenarios

Here, we formally define the parameters that actually form a certain scenario, and give an example of a real-life situation that would lead to such a performance behavior. The aim of each scenario is to properly represent one of global, partial or periodic delay.

4.2.1 Global Delay

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.

Example Scenario Use Case A new feature needs to be implemented for a new release. A 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 (Quality Assurance) and the new release is deployed to all users.

Scenario Parameters The parameter for this delay is given in Equation 7. Further, for every index i, we define the initial response time range as in Equation 8, as well as the range for the global change over all agents in Equation 9.

$$\mathcal{S}_G = \{a_i \mapsto [X_{a_i,0}, 420, X_{a,1}] \mid a_i \in \{a_1, a_2, a_3, a_4, a_5\}\} \tag{7}$$

$$\begin{split} X_{a_1,0} \sim \mathrm{U}(90,115), X_{a_2,0} \sim \mathrm{U}(100,130), X_{a_3,0} \sim \mathrm{U}(110,140), \\ X_{a_4,0} \sim \mathrm{U}(95,110), X_{a_5,0} \sim \mathrm{U}(100,110) \end{split} \tag{8}$$

$$X_{a,1} \sim \mathrm{U}(150,175) \tag{9}$$

4.2.2 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.

Example Scenario Use Case 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 “Round Robin” as its load balancing algorithm [30]. 20% of all users perceive the application with higher response times.

Scenario Parameters The parameter for this delay is defined in Equation 10. For every index i we define the initial response time range as in Equation 11, as well as the range for the partial change for agent a_5 in Equation 12.

$$\mathcal{S}_P = \{a_i \mapsto [X_{a_i,0}, \infty] | a_i \in \{a_1, a_2, a_3, a_4, a_5\}, a_5 \mapsto [X_{a_5,0}, 360, X_{a_5,1}]\} \tag{10}$$

$$\begin{aligned} X_{a_1,0} \sim \mathrm{U}(115, 125), X_{a_2,0} \sim \mathrm{U}(115, 120), X_{a_3,0} \sim \mathrm{U}(120, 145), \\ X_{a_4,0} \sim \mathrm{U}(105, 115), X_{a_5,0} \sim \mathrm{U}(110, 120) \end{aligned} \tag{11}$$

$$X_{a_5,1} \sim \mathrm{U}(140, 165) \tag{12}$$

4.2.3 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.

Example Scenario Use Case 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 13. The response time ranges are defined as in Equation 14.

$$\mathcal{S}_{PD} = \{a_1 \mapsto [X_0, 45, X_1, 65]^*\} \tag{13}$$

$$X_0 \sim \mathrm{U}(95, 115), X_1 \sim \mathrm{U}(160, 175) \tag{14}$$

4.3 Experiments

We now describe the methods of analysis and execution of the experiments introduced in Section 4, as well as the concrete results we achieved.

4.3.1 Methods of Analysis

The specified scenarios are executed within a physical testbed (described in Section 4.3.2) and results are analyzed and interpreted. The following sections outline the methods of analysis that are applied to the results.

Exploratory Data Analysis. Our first analysis approach is to examine the time series of monitored response times over time 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[c]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.

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 [28] package "changepoint". This approach makes sense, as we are not interested in the detection of spikes or other phenomena that can be considered as outliers in the statistical sense, 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. For this, we recall the simple view on delays we introduced in our taxonomy: $\lambda = |x_t - x_{t+1}| > c$. We adapt this model of change as follows. Instead of comparing two consecutive data points x_t and x_{t+1}, we compare the changes in the mean at the time point where changepoint have been detected by the algorithm. 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 denoted as λ, and can be defined as $\lambda = |\mu_{<\tau} - \mu_{>\tau}| > c$ via replacing two variables.

The threshold c is challenging to determine optimally. When c 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. When c is defined unnecessarily sensitive, the monitoring system is prone to 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. 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 $c = \mu_{<\tau} \cdot 0.4$. This means that if the new metric after the changepoint $\mu_{>\tau}$ is 40% above or below the old metric $\mu_{<\tau}$, the change is considered a *real change* as opposed to a *false positive*. 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: $c = \min(\mu_{>\tau}, \mu_{<\tau}) \cdot 0.4$.

Note that, in the case of this paper, 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 application, SLAs, and other factors and is usually determined by own empirical studies.

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

4.3.2 Testbed Setup

The experiments based on the described simulation model were executed in a local testbed consisting of 5 *Synthetic Agent Nodes* and 1 *Dynamic Web Service Component*. The agents operate in the local network as well. Each 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 an agent that records the data. The physical node setup consists of Intel Pentium 4, 3.4 GHz x 2 (Dual Core), 1.5 GB RAM on Windows and is running a Catchpoint agent node instance for monitoring. The Web service running on the target server has been implemented in the Ruby programming language and runs over the HTTP server and reverse proxy nginx and Unicorn.The physical web server setup is the same as the synthetic agent node setup, but is running on Ubuntu 12.04 (64 bit). During simulation, a random number generator is applied to generate artificial user behavior as specified in the distributions, which can be represented efficiently with common random numbers [18]. 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 we use MT19937 (Mersenne twister) [25] as a random number generator.

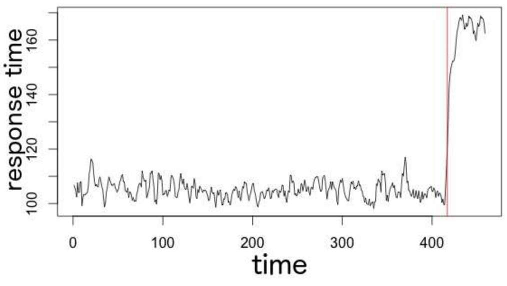
4.3.3 Results and Interpretation

We now discuss the results of our scenario data generation, and the results of applying the statistical methods described in Section 4.3.1. At first, we display a raw plot of the resulting time series data without any filters and interpret its meaning. Further, we apply a moving average smoothing filter (with a window size $w = 5$) to each resulting time series and conduct a changepoint analysis.

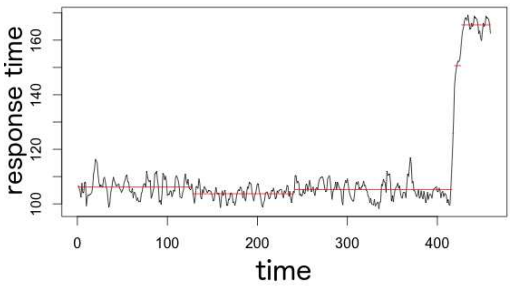
Global Delay The global delay forms the basis of our assumptions on how performance changes can be perceived on server backends. Both, the partial and periodic delay, are essentially variations in the variables time, interval and location of a global delay. Hence, the findings and interpretations of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

Exploratory Visual Analysis. In Figure 6c, we see the results for the 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.

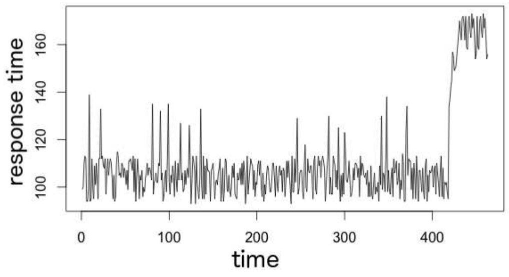
Statistical Changepoint Analysis. We apply changepoint analysis to the smoothed time series with a moving average window size of 5. In Figure 6a, we can immediately see how



(a) Variance Changepoint



(b) Mean Changepoint



(c) Global delay results raw time series

Fig. 6: Global Delay results

the smoothing affected the chart, compared to the chart with the raw data in Figure 6c: 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 6a is the changepoint in the variance, denoted by a vertical red line at the changepoint location. Table 1 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.
Next, we look at Figure 6b, where the change in the mean is indicated by horizontal lines 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 6b due to the very small change in the mean. The estimated numerical values for the changepoints are listed in Table 2. The table also lists the mean values, as well as the calculated threshold $c = \mu_{<\tau} \cdot 0.4$. The last column also states whether or not a detected changepoint in the mean is an *actual changepoint* (CP) as defined by the notion of *significant change* where $\lambda = |\mu_{<\tau} - \mu_{>\tau}| > c$, or a *false positive* (FP). Only one of the estimated changepoints has been identified as CP 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 1: Variance CP for $\mathcal{S}_G$

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

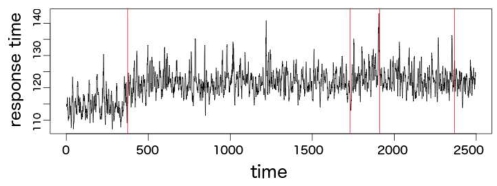
Table 2: Mean CP for $\mathcal{S}_G$

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.

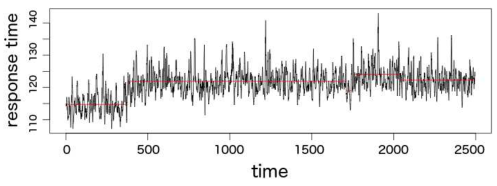
Exploratory Visual Analysis. As before, we plot the time series data and look for changes in our performance. In Figures 7a and 7b, we see rather stable time series charts, relatively volatile (spiky), due to the higher amount of conducted tests and random network noise. Around the time points 460-500 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 7c, we plot the same aggregation as in Figures 7a and 7b, 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 7d, 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.

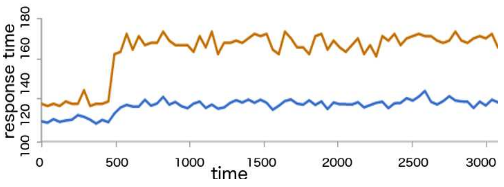
Statistical Changepoint Analysis. Analyzing both the changepoints in the variance in Table 3 and Figure 7a, as well as the changepoints in the mean in Table 4 and Figure 7b yields no surprise following our initial exploratory visual analysis. The changes that have been identified are not significant 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



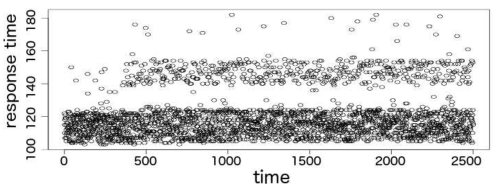
(a) Variance Changepoints



(b) Mean Changepoints



(c) 90th percentile and mean plot – the brown line (with higher response time values) denotes the 90th percentile



(d) Partial Delay Scatterplot

Fig. 7: Partial Delay results

by our post-processing step of checking the threshold. 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	$\sigma^2_{<\tau}$	$\sigma^2_{>\tau}$
374	12.88	24.25
1731	24.25	12.43
1911	12.43	6.63

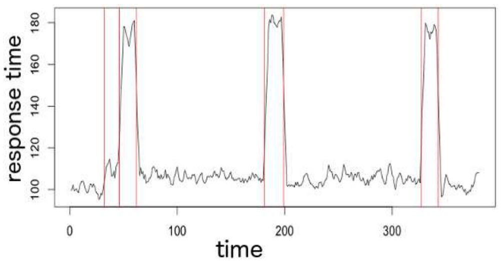
Table 3: Variance CP for $\mathcal{S}_P$

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

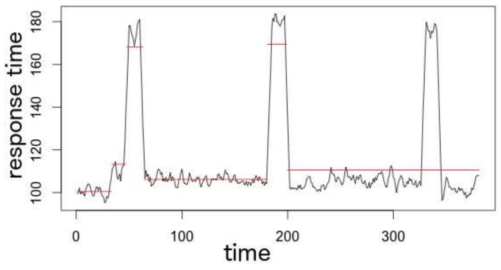
Table 4: Mean CP for $\mathcal{S}_P$

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 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



(a) Variance Changepoint



(b) Mean Changepoint

Fig. 8: Periodic Delay results

returning back the system returns to its normal operation. The intuition we have on periodic delays can be clearly observed in Figure 8. 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.

Statistical Changepoint Analysis. While the exploratory visual analysis in periodic delays was straight forward, the changepoint analysis brings some interesting insights. Figure 8a and Table 5 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.
The changepoint in the mean yields more interesting results, as seen in Table 6 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 8b, 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 analyses for changepoint in the mean and variance.

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 5: Variance CP for $\mathcal{S}_{PD}$

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

Table 6: Mean CP for $\mathcal{S}_{PD}$

5 Identifying Performance Degradations through Real-User Monitoring

Real User Monitoring (RUM) is a passive, external approach to monitoring web performance. This approach collects a more distributed set of data than the methods described in Section 4, so we adapt our analysis in accordance to RUM data and determine whether it is also a suitable method for identifying web performance degradation.

5.1 Real-User Monitoring

Active monitoring requires continuously running automated transactions that emulate real user behavior. This is effective for identifying performance degradations, but it requires additional resources: (1) infrastructure to support the synthetic agents, and (2) manpower to configure and maintain the tests. Real-User Monitoring leverages browser APIs to collect data specific to each end-user transaction, thus no additional resources are needed aside from storage and initial implementation of the client-side code. This means that any future code changes to the application being monitored require no adaptation to monitoring configuration, and all aspects of the application that users interact with can easily be measured. Additionally, varying populations of users interact with real world services, websites, and applications. The hyper-specific nature of synthetic testing can account for some of this variance by testing core functionality and vital aspects of a service. However, measuring everything is often not feasible, so important aspects of the service and certain populations' experience when using the service are often not measured. RUM complements Synthetic Monitoring by either capturing a significant sample of real traffic to gain adequate perspective of various populations or by capturing all traffic. As such, RUM can detect many more micro-outages in comparison to Synthetic Monitoring. The need to detect the outages is not necessary for all service providers, but for many it is critical to business especially if the outages occur in major markets.

If all traffic is sampled with Real-User Monitoring, all user environment variables will be collected, effectively dividing users into different populations including, but not limited to, geo-location, ISP, device, operating system, browser, responding server and datacenter, and multivariate test page version. This allows accounting for performance issues that are population specific and may not be detected with other monitoring methods. For example, users who speak different languages or reside in specific regions are often delivered different content from others, thus the code powering the service differs. As a result, the variation is code that can introduce performance degradations caused by bugs. Additionally, network peering constantly changes, and this can be seen as performance shifts with certain autonomous systems or ISPs.

However, the measuring of more unique data points causes the body of real user performance data to be complex and distributed. There are now many possible combinations of user environment characteristics such as geo-location, ISP, device, OS, etc. which makes accurately identifying performance degradations much harder than relying on active, external monitoring data. Since RUM data consist of many sub-populations, or subsets, we must adapt our analysis to take into account analyzing the full set of data, as well as particular user populations, or subsets.

5.2 RUM Methodology

We collected data utilizing a JavaScript beacon that leverages the World Wide Web Consortium Navigation Timing API[d] to collect performance data and MaxMind[e], a Geo-IP database in conjunction with ”User Agent” detection to determine user environments. The JavaScript is loaded on the page asynchronously to not impact the browser onload event or the loading of other resources required in construction of the page.

5.2.1 Data Collection

The data collected for our experiment was stored in a custom database designed to store a large amount data without doing any aggregations. By storing the data in this format we were able to conduct various statistical calculations on the entire population as well as the underlying subsets and combinations of those subsets e.g., end users geography by the quality of the connection illustrated as bandwidth and the response time. For our experiment we sampled traffic from a multinational website with a daily average number of 2 million pageviews with 7 pages viewed per visit by each unique visitor. The traffic to the website primarily came from the United States, but also originated from the United Kingdom, Canada, Australia, India, and China. We collected samples of traffic from various ISPs with end users connecting via the entire spectrum of available consumer level bandwidths. The data was sampled at a maximum of 500,000 unique pageviews per day for 5 days. Those measurements were not filtered and were aggregated to averages at an hourly interval (i.e., moving average window of 1 hour). An overview of the data collection parameters is given in Table 7.

Data Collection Method	Real-User Monitoring (RUM)
Sampling Period	10/22/2014 - 10/26/2014 (5 days)
Aggregation	Hourly
Sampling Limit	Max. 500,000 unique pageviews per day
Traffic (page views)	4,3 million views (100%)
United States/Canada	2,91 million views (68%)
International	1,39 million views (32%)
(UK, Australia, India, China)	

Table 7: Data Collection Parameters for the Real User Monitoring Experiment

5.3 Experiments

We now discuss how we applied changepoint analysis to web performance data collected via the RUM methodology as well as present the results from our investigation.

[d] http://www.w3.org/TR/navigation-timing-2/
[e] https://www.maxmind.com

5.3.1 Methods of Analysis

The following scenario is applied to a body of real user data gathered as described above. Changepoint analysis is applied to the "Document Complete" metric (which indicates the time it took until the JavaScript "onLoad" event has been fired.) as it is a good indication of webpage readiness for user interaction on a webpage. Increases in "Document Complete" times signify increases in wait time for the user until she or he can begin to use the webpage. A spike usually signals that something is failing in the rendering and execution of the webpage and causing it to become unusable.

We employ two methods of changepoint analysis on RUM data: *Global Analysis* and *Subset Analysis*, to account for the complexity and distribution of RUM data and then analyze the results of each.

Global Analysis. Global Analysis applies the changepoint detection on the whole set of RUM data, without any regard of individual populations or subsets within the collection. We perform the changepoint analysis on the "Document Complete" metric.

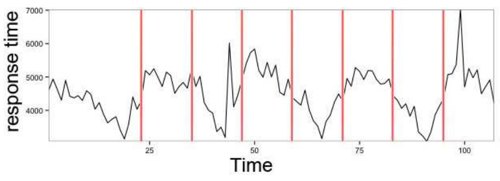
Subset Analysis. Subset Analysis applies the same changepoint detection analysis as described above, but to unique subsets of the data rather than the full set. We define our subset types based on user environment characteristics that we are able to capture (e.g., browser or ISP). Then, we read all of the data collected during the study to determine the specific subsets. Every unique value captured for each characteristic across the entire data set is a unique subset. Each individual user measurement that matches the defining value of the subset and subset type gets added to that subset. Subsets within each subset type are proper, meaning individual user measurements can only be added to one and only one subset of each subset type. However, the same individual user measurement may belong to different subsets of unique subset types.

As geography is a leading cause for increased latency across user populations, we use *user location* as a subset type and the specific *geo-location* to define our subsets. We predetermined our subsets to be user measurements from the United States and user measurements from everywhere else (International) prior to this experiment. We chose this as the largest population of the visitors to the website with the JS beacon is United States and there were not enough visitors from one unique comparative country for good sample set on its own. For each subset, we perform the changepoint analysis on the "Document Complete" metric and compare the detected changepoints in each subset to each other.

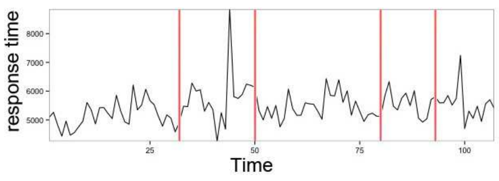
After performing both *global analysis* across the whole user population and *subset analysis* on two different user populations (US and International), we determine whether either method or a combination of both is sufficient to accurately detect web performance degradations caused by partial delays that we have already classified in Section 3.

5.4 Results and Interpretations

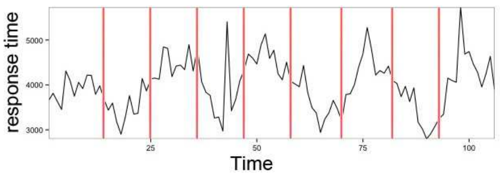
Similar to the analysis we did in section 4.3.3, we now apply changepoint analysis to real-life data where periodic and partial delays coincide. Figure 9a shows the periodic delay (cf. Section 3.2) in the global set of observations with a phase time of approximately 10-12 hours. The US subset in Figure 9c paints a very similar picture; just the changepoint detection time



(a) Global Set of RUM Observations



(b) RUM Subset: International



(c) RUM Subset: United States

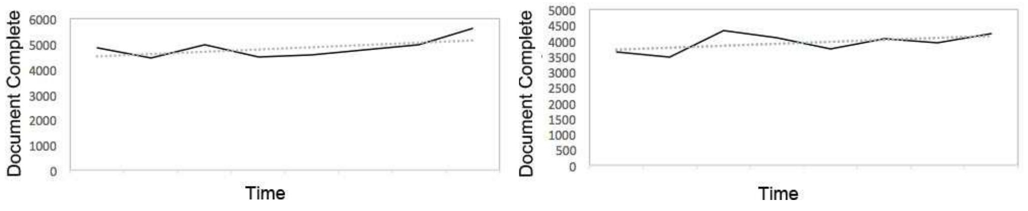
Fig. 9: RUM Observations

is slightly shifted. This is not very surprising, since the majority of visitors came from the US.

However, when it comes to the International subset in Figure 9b we see a lot less changepoints at different points in time that were not detected in the global set. The data presented here showcases a partial delay that could have not been detected in the global analysis, but rather needs a separation into viable subsets where the observed performance disparity becomes clear. It is best to avoid drawing conclusions based on high level, or aggregation of data with different characteristics, without investigating the underlying subsets of measurements as well as the user components that are driving the data.

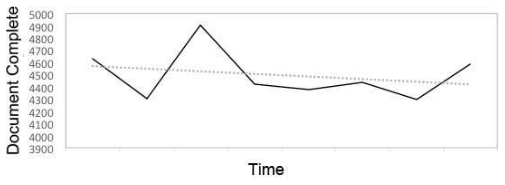
Simpson's Paradox in RUM. Additionally, when examining the differences between the subsets (US and International) and the global set, we are able to detect instances of the Simpson's Paradox. One particular case is found by focusing on a smaller period of time within the same data set (7 hours worth of measurements). Within this smaller timeframe, we are able to detect a trend of "Document Complete" measurements increasing in each subset, but in the whole set for the same timeframe, the "Document Complete" measurements were decreasing. In Figures 10, we illustrate the Simpson's Paradox by showing a decreasing trend in the global set, but an increasing trend in both the international and US subset.

This means, in addition to changepoint misalignment and inaccuracy we detect in comparing the whole set to the subset results, the presence of the Simpson's Paradox can also cause another difference in the results from global and subset analysis: the direction of the changepoint can differ at the same moment in time for the whole set and each subset. These three differences (misalignment, inaccuracy, opposite trends) require that changepoint analysis be performed on both a global set scale, as well as with regard to particular subsets in order to truly identify web performance delays and degradations within a complex distributed data set, such as one collected by RUM.



(a) International subset: Increasing trend in Document Complete for the same time frame

(b) United States subset: Increasing trend in Document Complete for the same time frame



(c) Global Set: Document Complete time is decreasing

Fig. 10: Simpson's Paradox in RUM data

6 Discussion

Real-User Monitoring complements Synthetic Monitoring. Synthetic and Real-User Monitoring are used by many website and application operators to ensure quality experiences for the users of their services. Both methodologies come with advantages and disadvantages as well as differences in how the data provided should be interpreted. While Synthetic Monitoring is beneficial for maintaining a constant schedule to reliably gauge availability and performance, its limitations in what can be effectively monitored in terms of both aspects of the service and attributes of the users interacting with the service are suitably complimented by real-user monitoring. However, since real-user monitoring is passive, measurements are only made when end users interact with the service, thus a lack of visibility into the root causes of availability loss prevents Real-User Monitoring from being an effective method for monitoring for availability. Consequently, Real-User Monitoring is predominantly implemented in the interest of measuring performance from a global perspective.

Adapting Methods of Analysis. As such, the data gathered using this method, when looked at collectively, requires the additional step of determining what key population subsets exist in the global data before an accurate analysis can be made. Since it is expected that certain subsets will show better performance than others, and fluctuations in traffic based on time of day or day of week often impact certain subsets, patterns that can only be explained by examining subsets separately can appear in the global data. This primarily occurs in geographic subsets but can exist in other data attributes depending on the service being monitored. The same is not true for synthetic data since the subsets are controlled and synthetic monitoring employs probing performed in uniformly distributed time intervals from the same testing locations

7 Threats to Validity

In the first part of our research, we employ a simplified simulation model in order to show how significant performance 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. Most importantly, in our simulation design, we do not actively inject the notion of web traffic workloads (as has been, for instance, taken in consideration in [23]), but rather simulate the variability of workloads (and therefore response times) in web services through specific scenarios parameters. Further, 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 in our experiments.

In the second part of our research, we analyze data collected from real users through Real User Monitoring (cf. 2.1.3). The subject used for the experiments is a multinational website in the financial sector. Data was collected over a period of 5 days with a representative amount of traffic (daily average number of 2 million pageviews). However, the specific industry chosen, as well as the specific dates to collect our data might limit the generality of our results.

8 Limitations of the Approach

The research presented in this paper makes use of active, external monitoring to identify performance degradations in web applications, which leads to certain limitations of our approach.

In the first part, our approach is limited to consider server side performance in our presented model and further in our experiments. Future research could address this issue by extending the model to consider frontend performance through metrics provided by modern browsers. These would incorporate notions of the Document Object Model (DOM) and other events triggered in JavaScript. This has been partially implemented in our approach in Section 2.1.3, by incorporating the Document Complete metric from browsers. However, there are several more metrics that need to be taken into account to provide a full view on frontend performance.

Furthermore, when conducting our experiments, the synthetic agents only considered response payloads, but assumed no request payloads (i.e., all synthetic requests happened through HTTP GET). Request payloads could be introduced when users submit data through, e.g., form submission or file uploads (i.e., HTTP POST).

Lastly, we have addressed misconfiguration on servers, especially misconfigured caches, as *global delays*. However there are not explicitly stated as part of our model. Future work could help to identify these more specific misconfigurations by incorporating its characteristics in our proposed model.

9 Related Work

Analyzing performance data observed through continuous monitoring in distributed systems and web services has produced a large 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. For instance, Pinpoint [9] 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 [19]. [14] 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. [1] proposes an approach which identifies root causes of latency in communication paths for distributed systems using statistical techniques. An analysis framework to observe and differentiate systemic and anomalous variations in response times through time series analysis was developed in [10]. [27] suggests an approach to automated detection of performance degradations using control charts. The lower and upper control limits for the control charts are determined through load testing that establish a baseline for performance testing. The baselines are scaled employing a linear regression model to minimize the effect of load differences. [11] proposes an automated anomaly detection and performance modeling approach in large scale enterprise applications. A framework is proposed that integrates a degradation based transaction model reflecting resource consumption and an application performance signature that provides a model of runtime behavior. After software releases, the application signatures are compared in order to detect changes in performance. A host of research from Borzemski *et al.* highlights the use of statistical methods in web performance monitoring and analysis [2–6]. The research work spans from statistical approaches to predict Web performance to empirical studies to assess Web quality by employing statistical modeling. [24] conducted an experimental study comparing different monitoring tools on their ability to detect performance anomalies through correlation analysis among application parameters. In the past, we have also applied time series analysis to the prediction of SLA violations in Web service compositions [22].

These approaches differ from ours mainly in two ways. Firstly, these systems usually either instrument the middleware or the communication layers (i.e., some sort of internal monitoring approach) to generate performance data. This leads to the situation that if users cannot access the application, the instrumentation (and thus system observation) and therefore the proposed detection of performance issues never happens in the first place. We make use of external, synthetic monitoring that allows us to detect and alert on performance issues before potential users of an application notice. Secondly, our approach uses an automated statistical analysis process based on a well-known, robust method (*changepoint analysis*) to detect fundamental shifts in our (performance) data distribution to provide on-line analysis of production systems. This analysis guides the detection and following alerting efforts for application monitoring solutions. However, there are certain aspects of our approach that are subject to improvement. We discuss those aspects in the following section.

10 Conclusion and Future Work

A taxonomy of root causes in performance degradations 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.

Following the work presented in this paper, 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. To counter this issue to a certain extent, we extended our analysis to data coming from real-user monitoring and analyzed how our initial model applies.

In our experiments, we focused on the response payload that resulted from the server side. It would be an interesting addition to extend our model to include request payloads (e.g., in the case of file uploads) and observe the effects. When considering future technologies discussed in Section 2, while performance optimizations and gains resulting from SPDY and HTTP/2 are likely to be seen in the absolute results, it is highly unlikely that they would impact the conclusions of our study.

Combining data from external monitoring and internal monitoring in order to automate or assist in root cause analysis and correlation of issues is another possible approach that should be considered. The simulation and analysis in this paper 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. In real-user monitoring, a move towards automated subset selection and analysis has been discussed as a possible extension.

Acknowledgements

The research leading to these results has received partial funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement 610802 (CloudWave), as well as from the European COST programme under Action identifier IC1304 (ACROSS). Numerical simulation was carried out during a research visit of the first author at Catchpoint Systems, Inc., New York, USA. The authors would like to thank Prof. Gernot Salzer for his comments on earlier versions of this work.

References

1. 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.
2. 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.
3. 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.
4. Leszek Borzemski and Anna Kaminska-Chuchmala. Knowledge engineering relating to spatial web performance forecasting with sequential gaussian simulation method. In *KES*, 1439-1448, 2012.
5. Leszek Borzemski and Anna Kaminska-Chuchmala. Knowledge discovery about web performance with geostatistical turning bands method. In *KES (2)*, pages 581–590, 2011.
6. Leszek Borzemski, Marta Kliber, and Ziemowit Nowak. Using data mining algorithms in web performance prediction. *Cybernetics and Systems*, 40(2):176–187, 2009.
7. 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.
8. Jie Chen and Arjun Gupta. *Parametric statistical change point analysis with applications to genetics, medicine, and finance.* Boston, 2012.

9. 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.
10. Yingying Chen, Ratul Mahajan, Baskar Sridharan, and Zhi-Li Zhang. A provider-side view of web search response time. *SIGCOMM Comput. Commun. Rev.*, 43(4):243–254, August 2013.
11. 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.
12. Thiam Kian Chiew. *Web page performance analysis*. PhD thesis, University of Glasgow, 2009.
13. Jürgen Cito, Dritan Suljoti, Philipp Leitner, and Schahram Dustdar. Identifying root causes of web performance degradation using changepoint analysis. In *Web Engineering, 14th International Conference, ICWE 2014, Toulouse, France, July 1-4, 2014. Proceedings*, pages 181–199, 2014.
14. 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.
15. Yehia Elkhatib, Gareth Tyson, and Michael Welzl. The effect of network and infrastructural variables on spdy's performance, 2014.
16. 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.
17. David Gourley. *HTTP : the definitive guide*. O'Reilly, Beijing Sebastopol, CA, 2002.
18. 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.
19. Paul Jaccard. The distribution of flora in alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.
20. Rebecca Killick and Idris A. Eckley. changepoint: an R package for changepoint analysis. *R package version 0.5*, 2011.
21. Andrew King. *Speed up your site : Web site optimization*. New Riders, Indianapolis, Ind, 2003.
22. Philipp Leitner, Johannes Ferner, Waldemar Hummer, and Schahram Dustdar. Data-Driven and Automated Prediction of Service Level Agreement Violations in Service Compositions. *Distributed and Parallel Databases*, 31(3):447–470, 2013.
23. 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.
24. Joao Paulo Magalhaes and Luis Moura Silva. Anomaly detection techniques for web-based applications: An experimental study. In *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on*, pages 181–190. IEEE, 2012.
25. 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.
26. Paul. Mockapetris.
27. 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.
28. R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.
29. Forrester research. Ecommerce web site performance today: An updated look at consumer reaction to a poor online shopping experience, August 2009.
30. Behrooz A. Shirazi, Krishna M. Kavi, and Ali R. Hurson, editors. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos,, 1995.
31. Amanda Strong. A review of anomaly detection with focus on changepoint detection, 2012.
32. Clifford H Wagner. Simpson's paradox in real life. *The American Statistician*, 36(1):46–48, 1982.