universität wien

MAGISTERARBEIT / MASTER'S THESIS

Titel der Magisterarbeit / Title of the Master's Thesis

„Data Analytics for Smart Product Configuration – Statistical Analysis of Operational Log Data in a Smart City Platform"

verfasst von / submitted by

Tomislav Maruščak

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Magister der Sozial- und Wirtschaftswissenschaften (Mag. rer. soc. oec.)

Wien, 2018 / Vienna 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 951
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Magisterstudium Statistik
Betreut von / Supervisor:	ao. Univ.-Prof. Dipl.-Ing. Dr. Erhard Reschenhofer
Mitbetreut von / Co-Supervisor:	

Abstract

The analysis of log files has attracted an unprecedented attention of scientific community in the last decade. While usage of log files for debugging purposes is common, the analysis for optimization and further purposes is an open hot topic in literature. Unleashing the full potential of log file analysis is prevented by a number of obstacles, e.g. lack of standards, heterogeneity of features (mostly nominal, but also natural text), etc.

This thesis proposes a statistical approach for automatizing the analysis of log files. In particular, we present a set of methods for the extraction of relevant metrics from semi-structured log files and structural break tests to detect abrupt changes in these metrics. In addition we introduce means to identify deviations from expected system behaviour and to dynamically tune thresholds for the detection of anomalous system conditions. Proposed methodology was successfully tested by using actual log file from a real-world implementation.

The work performed in this thesis is a part of the H2020 EU-funded project "SUPER-SEDE". Parts of this thesis will be included in the upcoming project deliverable *D2.5DataAnalysis_v2*.

Keywords: data analysis, smart city platform, rule evaluation, KPI extraction, structural break tests, kernel density estimation, operational log data, nominal variables

Abstract

Die Analyse von Log-Daten bekommt im letzten Jahrzehnt zunehmende Aufmerksamkeit in der Wissenschaft. Während die Verwendung von Log-Daten für Debugging schon üblich ist, ist deren Analyse zur Optimierung und zu sonstige Zwecken ein offenes Problem in der Literatur. Das volle Potential der Analyse von Log-Daten wird eingeschränkt durch unterschiedliche Hindernisse, wie z.B. keine (fehlende) Standards oder heterogene Variablen (sie können, unter anderem, nominal, aber auch alltägliche Sprache sein).

Diese Magisterarbeit schlägt eine statistische Vorgehensweise für die Analyse von Log-Daten vor. Insbesondere stellen wir eine Menge von Verfahren dar, sowohl zur Ableitung von wichtigen Maßen aus semi-strukturierten Log-Daten, als auch Strukturbruchtests zur Feststellung von plötzlichen Änderungen in diesen Maßen. Zusätzlich dazu führen wir Wege zur Erkennung von Abweichungen vom erwarteten Verhalten des Systems ein, und zur dynamischen Anpassung der Grenzen zur Erkennung eines atypischen Zustands des Systems. Die vorgeschlagene Methodik wird an Daten einer Implementierung aus der realen Welt erfolgreich getestet.

Die Arbeit, die in dieser Magisterarbeit präsentiert worden ist, ist ein Teil des H2020 EU-geförderten Projektes "SUPERSEDE". Teile davon werden im kommenden Deliverable *D2.5DataAnalysis_v2* enthalten sein.

Stichwörter: datenanalyse, smart-city-platform, rule-evaluation, KPI extrachierung, nominale variablen, strukturbruchtests, kerndichteschätzer, operative log daten

Acknowledgement

I would first like to thank my thesis advisor ao. Univ.-Prof. Dipl.-Ing. Dr. Erhard Reschenhofer of the Faculty of Business, Economics and Statistics at University of Vienna. The door to Prof. Reschenhofer's office was always open whenever I ran into a trouble spot or had a question about my research or writing. He allowed this paper to be my own work, yet pointed me in the right direction.

I would also like to thank Dr. techn. Danilo Valerio from SIEMENS AG Österreich, who mentored me from a corporate point of view, and enabled my embarking on a career at SIEMENS AG. He was also a second reader of this thesis, and I am gratefully indebted to his for his very valuable comments on this thesis.

Without their passionate participation and input, the thesis could not have been successfully conducted. Thank you.

Author

Tomislav Maruščak

Contents

List of Figures

List of Tables

1 Introduction

The activity of billions of electronic devices and software systems is monitored, traced, and saved in so called log files. These log files are of crucial importance in case a problem occurs and the cause has to be identified, but can also be used for additional purposes. The information contained in a log file can be used to optimize already operational systems, to recognize design flaws, to detect security threats, to identify bottlenecks, and many more. A growing trend relates to the usage of log files to predict upcoming errors before they occur.

Formally, a **log file** in the information science context is defined as the automatically produced and time-stamped documentation of events relevant to a particular system [1]. Log files can be divided into several families, depending on their nature. Each family has its own peculiarities, which makes it impossible to talk about log data analysis as a unique problem. A thorough review of the common problems in the log file analysis can be found in [7].

In a test setting a debugger can exploit the information from the logs manually. This works when the target of the analysis is defined. Due to the quantity of the data coming from a real, live system, it is impossible to approach it manually. The quantity of the data may be addressed by abstraction [14, 25], which is a process of grouping together the similar log entries arising in the short time, due to the fact that they could be coming the same *fault* which propagates to the system, therefore not bringing any new information. Zheng et al. [28] were able to filter similar events and compress the file by as much as 99.97%. Another big topic in the log file analysis is the lack of format, in the general case. Software developers may be writing the events in natural language, possibly with time stamp and some variables. Log parsing [15] may be necessary in order to achieve structure.

1.1 On the Statistical Analysis of Log Files

Sensor data, that could be generated by a variety of relatively inexpensive sensors installed on any type of machinery, can record valuable information about a machine's state, like operating temperature, vibration, RPM, etc., making it easier to directly apply statistical tools and methods. Data collected from sensors is usually

straightforward to analyse as it comes as a series of unidimensional readings over the time, forming a time series.

Log files differ considerably from sensor data. The files may be very heterogeneous, highly depending on the use case, as well as the way the software developer designed and implemented them (meaning that there is no unique standard). Most messages are written in plain text, and it is possible that not all of the possibly relevant information is logged. What is often missing is the context of the event logged. If the software developer imagined that the main use of the logs would be for debugging, it is likely that he wouldn't keep track of the successful events. Apart from the issue that software log data are mostly textual, they are also coming at **random times**, which are depending on the users' activity. Due to the current lack of standards, the developers have the freedom to choose how the logs are generated. That can mean that the messages are stored as text messages with a time stamp, generating a semi-structured output, or as a series of nominal variables, outputting a structured log. This prevents the direct application of machine learning approaches, as they require numeric inputs. As a result it is necessary to pre-process log files to extract the right metrics from logs in a meaningful way, before applying desired statistical tools on it.

The difference in the analysis of the log data is not only huge among the sensor log data and software usage log data, but also among the sensor log data for various machines, or for the software logs, among the types of software. The analysis is also depending on the goals (use) of the data - printer logs could be used for troubleshooting, while web browsing data could be used to monitor and optimize network flows, improve ads or recognize intrusion attempts [21].

Apart from the issues stated before, metrics extracted from the log data may not (always) fulfil the formal statistical assumptions of the methods we would like to use, etc. Furthermore, the same use cases among different companies could be generating different types of data, or their needs from the data could vary, making it impossible to simply directly apply the same approach, and automate the analysis in that way.

Oliner et al. [21] have provided an overview of typical problems while dealing with log data, as well as some suggestions.

1.2 What is SUPERSEDE?

The SUPERSEDE [1] project is based upon the assumption that the current software engineering methods and tools are still poorly exploiting the increasing volume of available user feedback and context data, hindering the creation, evolution and adaptation of software services and applications that fulfil end-user expectations on quality of experience (QoE) and quality of service (QoS). The overall project objective is to deliver methods and tools to support decision-making in the evolution and adaptation of software services and applications by exploiting end-user feedback and runtime data, with the overall goal of improving end-users' QoE.

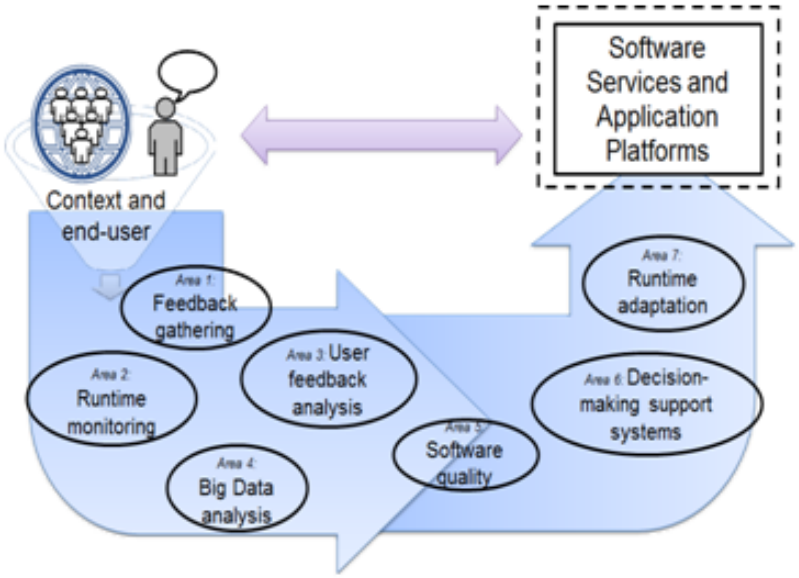


Figure 1: The SUPERSEDE vision [2]

The vision of the project is summarized in Figure 1. End-user feedback is available in online forums, app stores, social networks and novel direct feedback channels, which connect software applications and service users to developers. Runtime data can be gathered from environmental sensors, monitoring infrastructure, usage logs, etc. End-user feedback and data will be analysed to support decision-making tasks both for **(i)** *software evolution* and **(ii)** *adaptation*. Software evolution decision-making performed by analysts, system architects, developers and project managers will facilitate faster innovation cycles that deliver early value to a wide target of

[1] This section is partially following the official project documentation published at `https://www.supersede.eu/project/objectives/`

potential customers. Examples are the identification of new business use cases; the discovery and prioritization of new requirements; the identification of issues to be solved through software evolution; the definition of test cases; and strategic / release planning, based on the analysis of past series of software evolution cycles.

Software adaptation will occur at runtime not only to match the personal characteristics of the end-user (either as individual or as instance of a type of user), but also to respond timely to quickly changing context conditions [2].

Working Package	Description
WP1	Feedback and Data Collection
WP2	Feedback and Data Analysis
WP3	Requirements For Methods and Tools
WP4	Enacting the Decisions
WP5	Software Tool Suite
WP6	Use Case Validation
WP7	Dissemination and Exploitation
WP8	Project Management

Table 1: Description of the SUPERSEDE working packages

1.2.1 Thesis Contribution to SUPERSEDE

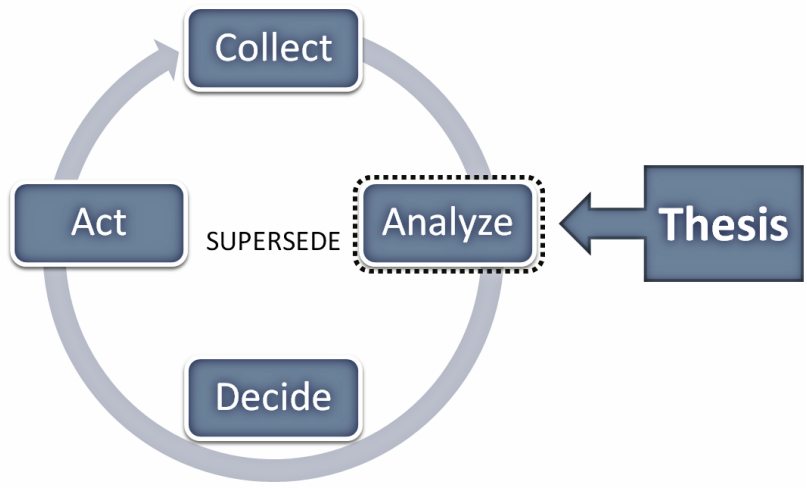


Figure 2: The proposed SUPERSEDE cycle

The **collect** step consists of the gathering of the data from the end-users, the execution context and usage logs. The data is passed on to **WP Analyse** to provide results and visualizations extracted from the maintenance logs and the

feedback data, ideally automating the generation of user models from the patterns of usage, and deriving the indicators for the QoE. The **decide** step takes the outputs of my analyses in order to derive appropriate decision-making models that can be fed by user feedback and monitoring logs to enable automated and semi-automated decision making– either by defining the software evolution tasks (by e.g. identifying new users' requirements, or issues to be solved through software maintenance - *in the next versions*) or by performing dynamic software adaptation (e.g. getting recommendations about actions to be implemented to keep the QoS at a good level). The **act** step would then implement the decided changes at the right moment (i.e. schedule and assess the impact of the executed actions). This is displayed in Figure 2.

This thesis falls in the context of the WP Analyse. The goal is to **improve** the Event-Condition-Action (ECA) rules, which are currently mostly threshold-based, whereby the thresholds mostly have to be **manually defined**. They are used to raise **alerts** from the software monitoring as well as feedback data to the upcoming working packages. We wish to enrich these rules with more sophisticated methods, in a way that the current marked events would still be captured, as well as some further ones. Predicting the inadequate system states, which could negatively reflect on the users' QoE, allows us to raise an alert to the software developers, allowing them to react, either in a way of software runtime adaptation, or by directing the evolution of the software according to the (predicted) users' needs. The descriptive analysis will allow to correctly identify the most interesting use cases, which will be described in the coming chapters (2,3).

Ideally, an automated rule extraction system would be developed, that generates rules and thresholds automatically, based on feedback and monitoring data. The feasibility of this will not be covered in the scope of this thesis. Due to the lack of the reliable feedback data, the focus will be on detecting of anomalies in the monitoring data. The more concrete application of SUPERSEDE in the SIEMENS use case will be described in Section 2.1.

Parts of this thesis will be published in the scope of the project's documentation and deliverables.

1.3 Problem Definition

This thesis tackles most of the problems described above. In particular:

- too many method names for meaningful clustering of data
- extraction of relevant numerical KPIs (key performance indicators) from categoric data
- definition of response duration as a time difference between users request and system's response
- extraction of trends in the response duration data
- response duration forecasting
- detection of abrupt changes in these trends
- identification of deviations of periodicity.

2 Domain Description

2.1 Smart City Information Ecosystem

As a part of the Aspern Smart City Research group [6], Siemens AG Österreich has developed the "smart city information ecosystem", which is a platform between API providers, which provide access to specific datasets, and app developers, who utilize the data to provide a service to their users.

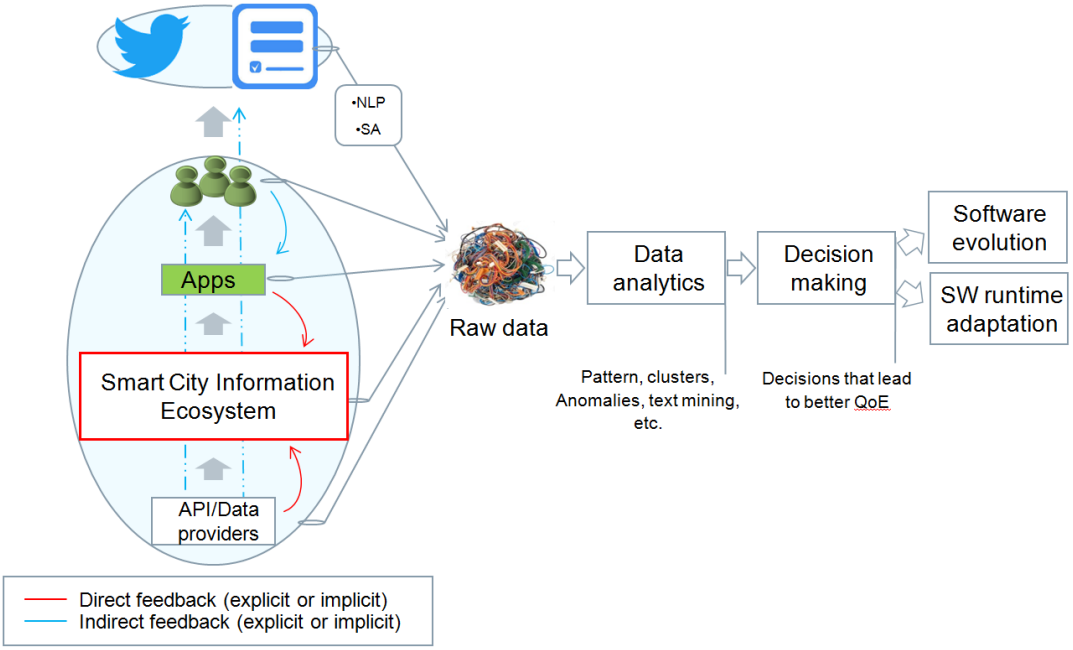


Figure 3: Smart City Information Ecosystem

The interaction between API providers and app developers occur via REST API. Every (in-)direct call of the APIs is logged. We monitor this activity at different places within the ecosystem (see Section 2.2 and Figure 4), which generates different logs, that are semantically interconnected.

This represents one source of our data. API providers, app developers or their end-users may further provide feedback data. After it is preprocessed with natural language processing and semantics analysis tools, is it joined to the rest of the data.

The idea is to use the data from these two separate sources to extract some insights about the problems that may arise. This describes the collect step of the SUPERSEDE proposed iterative cycle, described in Section 1.2.1 and depicted in Figure 2.

Another goal is to extract user intentions from their comments, perform customer segmentation and derive metrics from the data that would be good indicators for QoS.

Then comes the decide step, in which we would like to derive (semi-) automated decision making models that can be fed by the outputs of my analysis. They would be able to identify the users needs, that could be implemented in the next versions, or perform a dynamic software adaptation meaning that it maintains the software's QoS at a good level while running.

The last step of the cycle implements the decided changes at the right moment, i.e. performs scheduling of the executed actions.

2.2 The Data

A snapshot of the original log data is shown in Figure 4. The data contains method calls, the time stamp, and some other parameters that are mainly categorical. Since most of the approaches I intend to use rely on numbers rather than on unordered categories, the first challenge we face is to deduce metrics from these categories in a meaningful way. Since we deal with software usage logs, a non-constant Δt (meaning a constant frequency), prevents us to (directly) use time series approach.

Timestamp	SRC	Class	Usr	Rol	SessionID	EvType	Direct	MethodNameOrig	ContentType	Content
2016-01-05 14:52:44.966	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	REQUEST	/getCountForPropertiesCount	application/json	0
2016-01-05 14:52:44.966	MANAGMENT	controllers.APIsControllerV1	0	0	20fb3a6dd8667c036020551a7b07	GET	REQUEST	/getCountForPropertiesCount	application/json	0
2016-01-05 14:52:45.170	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	RESPONSE	/getCountForPropertiesCount	application/json	SUCCESS
2016-01-05 14:52:45.170	MANAGMENT	controllers.APIsControllerV1	0	0	20fb3a6dd8667c036020551a7b07	GET	RESPONSE	/getCountForPropertiesCount	application/json	SUCCESS
2016-01-05 14:52:45.185	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	REQUEST	/getTagsFrequency	application/json	0
2016-01-05 14:52:45.201	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	RESPONSE	/getTagsFrequency	application/json	SUCCESS
2016-01-05 14:52:45.201	MANAGMENT	controllers.APIsControllerV1	0	0	20fb491d719682a920ade6bcbb80	GET	REQUEST	/getTagsFrequency	application/json	0
2016-01-05 14:52:45.201	MANAGMENT	controllers.APIsControllerV1	0	0	20fb491d719682a920ade6bcbb80	GET	RESPONSE	/getTagsFrequency	application/json	SUCCESS
2016-01-05 14:52:45.232	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	REQUEST	/getGeoCoordinates	application/json	0
2016-01-05 14:52:45.232	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	RESPONSE	/getGeoCoordinates	application/json	SUCCESS
2016-01-05 14:52:45.232	MANAGMENT	controllers.APIsControllerV1	0	0	20fb4b0a427b75477b5030012392	GET	REQUEST	/getGeoCoordinates	application/json	0
2016-01-05 14:52:45.232	MANAGMENT	controllers.APIsControllerV1	0	0	20fb4b0a427b75477b5030012392	GET	RESPONSE	/getGeoCoordinates	application/json	SUCCESS
2016-01-05 14:52:45.248	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	REQUEST	/getAll	application/json	0
2016-01-05 14:52:45.263	MANAGMENT	controllers.APIsControllerV1	0	0	20fb4cf83c1b4affe12aa119e7e2	GET	REQUEST	/getAll	application/json	0
2016-01-05 14:52:45.263	MANAGMENT	controllers.APIsControllerV1	0	0	20fb4cf83c1b4affe12aa119e7e2	GET	RESPONSE	/getAll	application/json	SUCCESS
2016-01-05 14:52:45.279	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	RESPONSE	/getAll	application/json	SUCCESS
2016-01-05 14:52:46.644	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	REQUEST	/getLastnElement/2	application/json	0
2016-01-05 14:52:46.644	MANAGMENT	controllers.APIsControllerV1	0	0	20fba34e3e6d2ef6777e37c02ea0	GET	REQUEST	/getLastnElement/2	application/json	0
2016-01-05 14:52:46.660	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	RESPONSE	/getLastnElement/2	application/json	SUCCESS
2016-01-05 14:52:46.660	MANAGMENT	controllers.APIsControllerV1	0	0	20fba34e3e6d2ef6777e37c02ea0	GET	RESPONSE	/getLastnElement/2	application/json	SUCCESS
2016-01-05 14:52:47.448	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	REQUEST	/getCountForPropertiesCount	application/json	0
2016-01-05 14:52:47.448	MANAGMENT	controllers.APIsControllerV1	0	0	20fbd58ac8eb4aa37b251b3ae120	GET	REQUEST	/getCountForPropertiesCount	application/json	0
2016-01-05 14:52:47.495	MANAGMENT	controllers.APIsControllerV1	0	0	20fbd58ac8eb4aa37b251b3ae120	GET	RESPONSE	/getCountForPropertiesCount	application/json	SUCCESS
2016-01-05 14:52:47.511	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	RESPONSE	/getCountForPropertiesCount	application/json	SUCCESS
2016-01-05 14:52:47.511	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	REQUEST	/getTagsFrequency	application/json	0
2016-01-05 14:52:47.511	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	RESPONSE	/getTagsFrequency	application/json	SUCCESS
2016-01-05 14:52:47.511	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	REQUEST	/getGeoCoordinates	application/json	0
2016-01-05 14:52:47.511	MANAGMENT	controllers.APIsControllerV1	0	0	20fbd97c405351092652816095 97	GET	REQUEST	/getTagsFrequency	application/json	0
2016-01-05 14:52:47.511	MANAGMENT	controllers.APIsControllerV1	0	0	20fbd97c405351092652816095 97	GET	RESPONSE	/getTagsFrequency	application/json	SUCCESS
2016-01-05 14:52:47.526	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	RESPONSE	/getGeoCoordinates	application/json	SUCCESS
2016-01-05 14:52:47.526	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	RESPONSE	/getAll	application/json	0
2016-01-05 14:52:47.526	MANAGMENT	controllers.APIsControllerV1	0	0	20fbda64dc1c38bd9128a718770f	GET	REQUEST	/getGeoCoordinates	application/json	0
2016-01-05 14:52:47.526	MANAGMENT	controllers.APIsControllerV1	0	0	20fbda64dc1c38bd9128a718770f	GET	RESPONSE	/getGeoCoordinates	application/json	SUCCESS
2016-01-05 14:52:47.542	WEBAPP	server.RESTfulAdministrator	0	0	0	GET	RESPONSE	/getAll	application/json	SUCCESS
2016-01-05 14:52:47.542	MANAGMENT	controllers.APIsControllerV1	0	0	20fbdb6c66784adc7edf67d344df	GET	REQUEST	/getAll	application/json	0
2016-01-05 14:52:47.542	MANAGMENT	controllers.APIsControllerV1	0	0	20fbdb6c66784adc7edf67d344df	GET	RESPONSE	/getAll	application/json	SUCCESS
2016-01-05 14:53:35.861	WEBAPP	server.RESTfulAdministrator	0	0	0	POST	REQUEST	/addUser	application/json	{username:52aded1653
2016-01-05 14:53:35.877	MANAGMENT	controllers.APIsControllerV1	0	0	2107a85cdbbdea1ca2c9b0b974d00	POST	REQUEST	/addUser	application/json	{username:52aded1653
2016-01-05 14:53:38.002	WEBAPP	server.RESTfulAdministrator	0	0	0	POST	RESPONSE	/addUser	application/json	SUCCESS

Figure 4: A fragment of the monitoring log

Monitoring logs trace the user's activity and the corresponding system responses. While response times mostly depend on user enquiries (together with the overall system load), the users' enquiries come at random times. A variety of approaches will be tested in order to address these challenges.

Further properties of the data will be investigated as well, since the deviation from any found regularities in the data might also give us an insight into the overall system health.

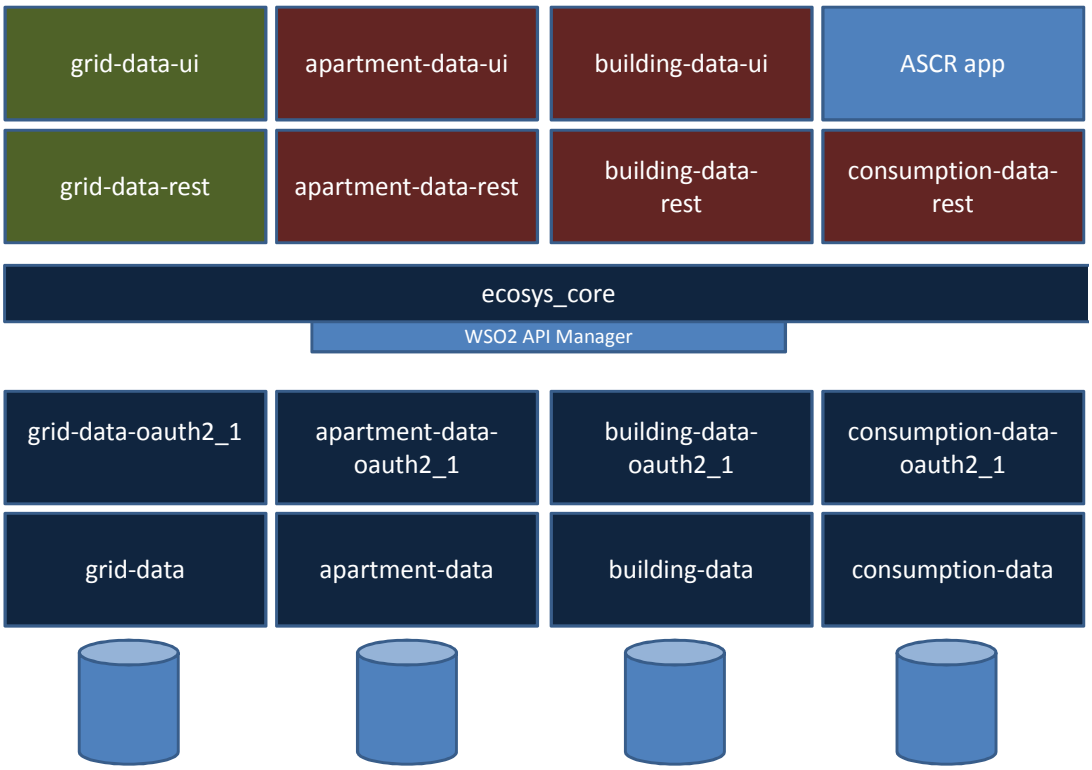


Figure 5: A schematic display of the information ecosystem

The implemented ecosystem schematics is displayed in Figure 5. E.g., a user clicks on a button in his app, which implicitly calls `grid-data-ui` service in our ecosystem. The request is prepared and forwarded to the service `grid-data-rest`. All of the requests go through the `ecosys_core` service, before (if necessary) they go through `grid-data-oauth2_1` and `grid-data` services and access the database. The response then follows the same path back to the user. Each of the services is

monitors inputs and outputs, in the features defined in the coming chapter. This generates several log files that are interconnected, so I merge them before the data preprocessing.

2.2.1 Features

For each row, we record the following:

- TIMESTAMP – in the format: `yyyy-mm-dd TZ hh:mm:ss.ms`,
- SOURCE – which one of the services in Figure 5 is an event coming from,
- CLASS – the java class where the event originated,
- USER – MD5 of the user name if the user is logged in,
- ROLE – the role of the user (publisher or developer),
- SESSION ID – a unique value for a pair request–response,
- EVENT TYPE – either get or post, login or logout,
- DIRECTION – either request or response,
- METHOD NAME – full path of the method being called,
- CONTENT TYPE – the format of the body (JSON, TXT, XML),
- CONTENT – if the event was successful, the body in the predefined format, else one of the error messages (error, failure, network error, success with error message etc.)

3 Data Preprocessing

Due to the issues defined earlier, we cannot use the data directly, so we pre-process it in order to extract the meaningful metrics, to apply our proposed tools on.

3.1 MethodName Clustering

The method name (see previous section) is the name of the REST API being called (e.g. `getWeather`). It may or may not include some parameters (e.g. `getWeather/city=VIENNA`) and attributes (e.g. `getWeather/city=VIENNA?temp=-celsius`). While the attributes are easier to strip (being (almost) always after the "`?`" sign in the url, the parameters may be masked on the left side of it. To make it easier to inspect, I plotted the method names in a hierarchical dendrogram, with clearly set boundaries between clusters, where the distance measure for clustering was given by Levenstein's string (dis-)similarity measure as defined in equation (2). The illustrative clustering for the first 100 methods is displayed in Figure 6.

Formally, Levenshtein distance [19] $\mathrm{lev}_{a,b}(|a|,|b|)$ between two strings a, b (of lengths $|a|$ and $|b|$ respectively) is given recursively by

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j), & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + \mathbb{I}_{\{a_i \neq b_j\}} \end{cases} & \text{otherwise.} \end{cases} \tag{1}$$

This is basically a count of either character replacements, insertions, or deletions to convert one string into the other. From distance function, we can easily derive the Levenshtein similarity measure ($\in [0,1]$):

$$\texttt{levenshteinSim}(a,b) = 1 - \frac{\mathrm{lev}_{a,b}(|a|,|b|)}{\max(|a|,|b|)} \tag{2}$$

Based on method names, a cut-off point was determined for which the strings would be clustered together. From the 570 unique method names (without the parameters), 132 clusters with similarity of more than 18% have been found. This is a reasonable amount that can further be cross-checked and edited manually, and adjusted if necessary. It can further be manually grouped to 32 groups of methods,

Figure 6: Clustered method names using the Levenstein's similarity measure, for the first 100 unique method names

based on frequency of calls and similarity of paths. The clustering dendrogram can be seen in Figure 6. Each color is representing a new cluster, and Levenshtein distance between the clusters is proportional to the length of the branches.

These methods may further be grouped in meaningful APIs, by analysing the nested structure of method paths.

3.2 SessionID

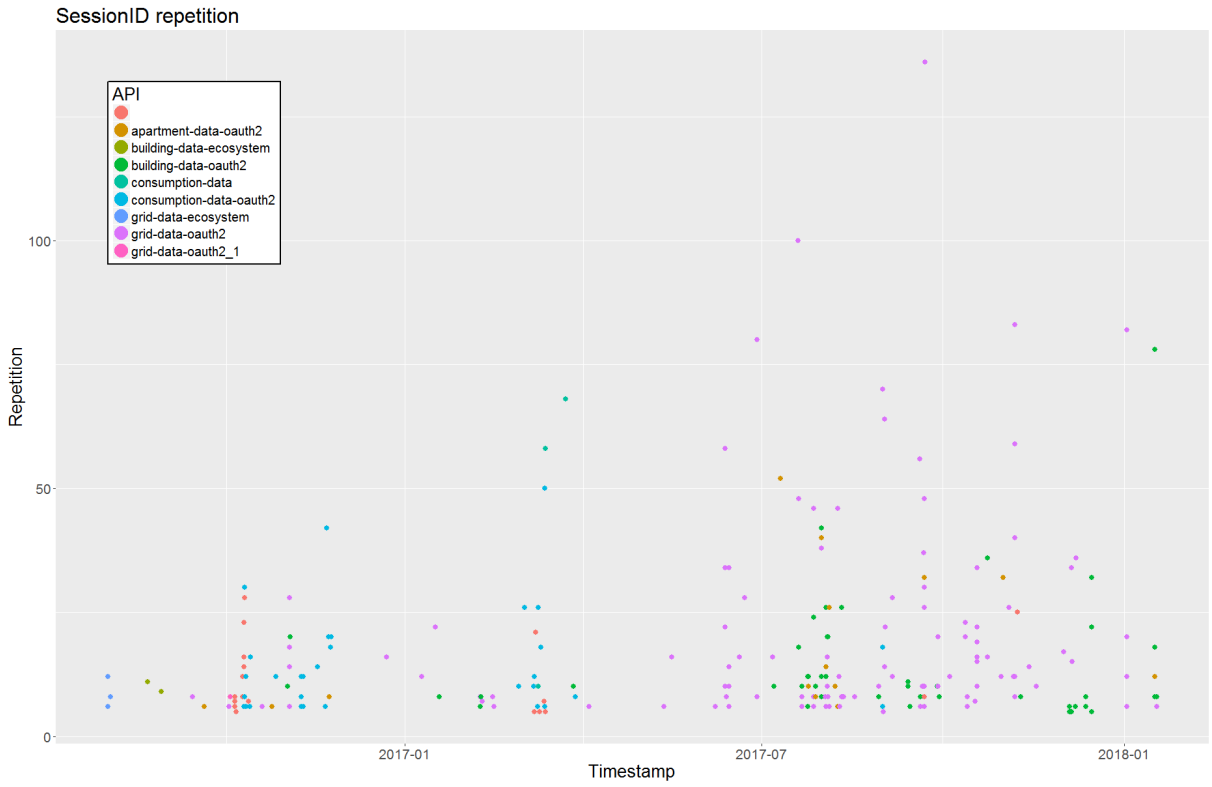


Figure 7: Unexpected behaviour in the SessionID variable

As presented earlier, user activity is being logged in a form of a `REQUEST` entry, followed by a `RESPONSE` as a system's reaction. They are two separate lines in the log file, which can be matched by the unique identifier `SessionID`. Note that they may or may not be stored as consecutive lines, for example if requests are coming faster than its corresponding response can be delivered (e.g. the request is requiring a database access that, depending on the form, in extreme cases, could take up to several minutes).

This means that each SessionID should appear exactly twice, which is not always

the case. It happens that there exists a unique value of SessionID, corresponds exclusively to the request or to the response, which could mean that its matching pair wasn't logged. It also happens that the same SessionID is held for hundreds of lines, as can be seen in Figure 7, which shows the number of repetitions of a unique SessionID value, if it's greater than 2.

It was among the first indicators of unexpected system behaviours visible in the data, which is forwarded to the system developers for further investigation. The causes have been traced back to some not elegant implementations, that were adequate at an earlier time point, but aren't sufficient for current versions and use cases. Mind that this is exactly what the SUPERSEDE proposes, to make use of the monitoring logs in order to assist the software evolution and adaptation processes.

In 99.2% of the entries, the SessionID feature is behaving as expected. This means that only a few data has to be discarded.

3.3 Response Duration

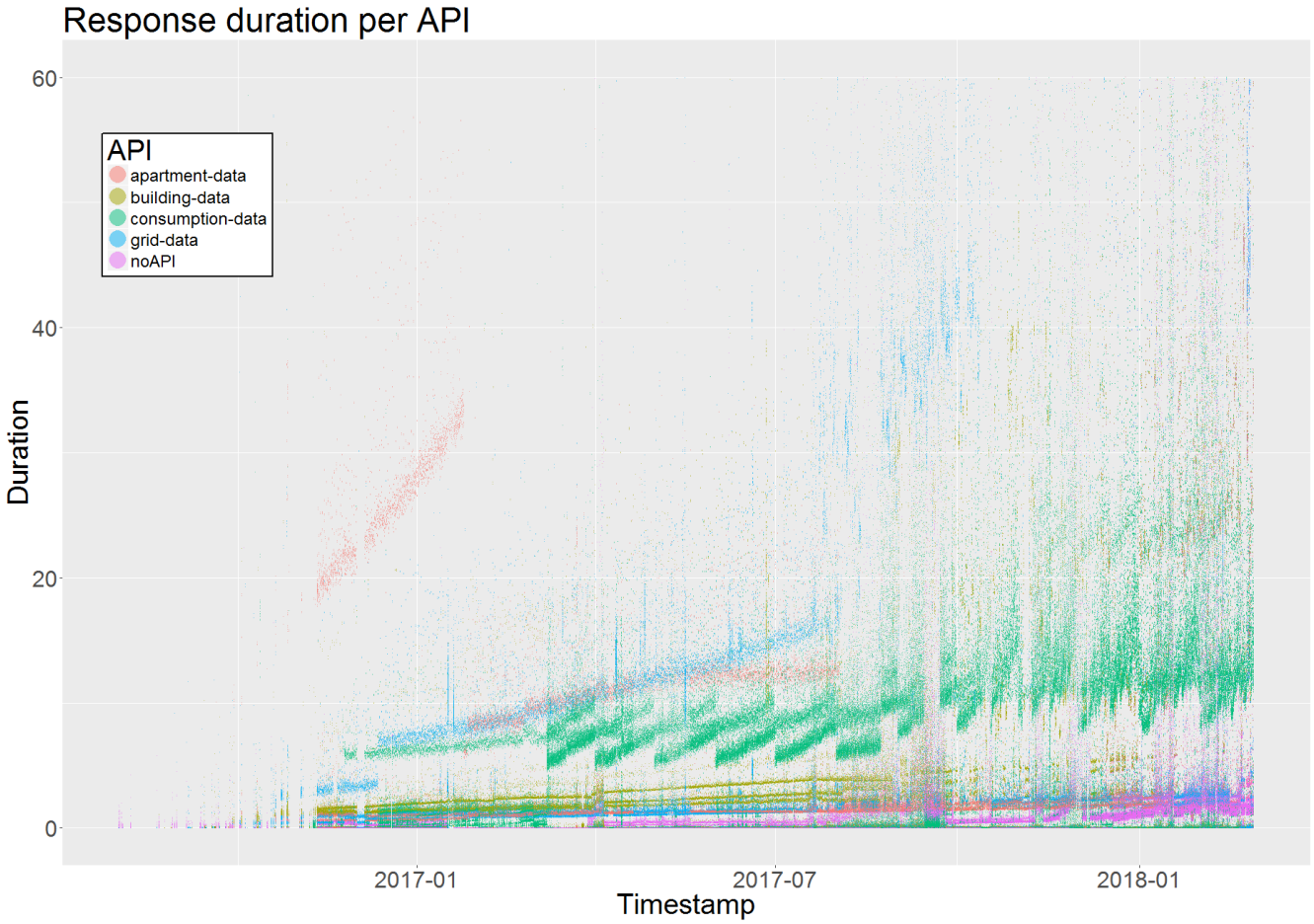


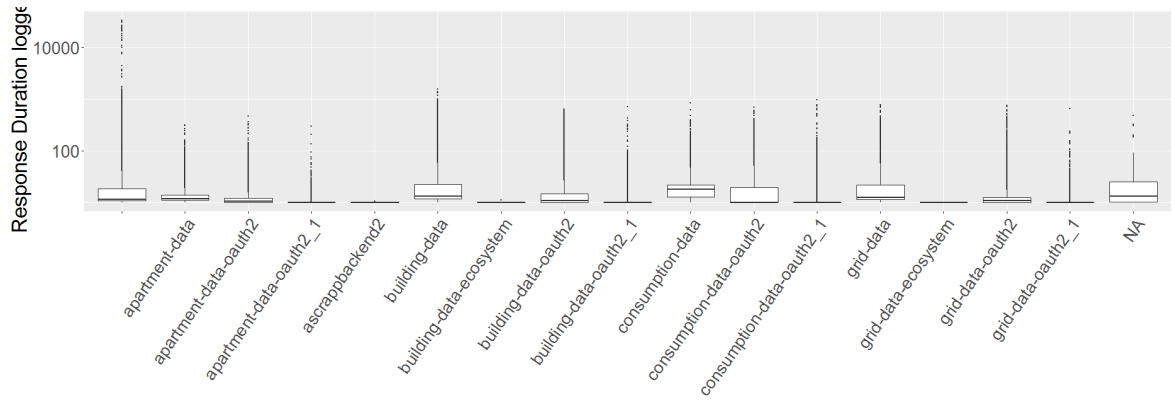
Figure 8: Response duration over the APIs, cropped at 60 seconds

After initial preprocessing of SessionIDs, it is possible to compute response duration by taking the time difference from a request to its corresponding response, matched by a (now) unique SessionID.

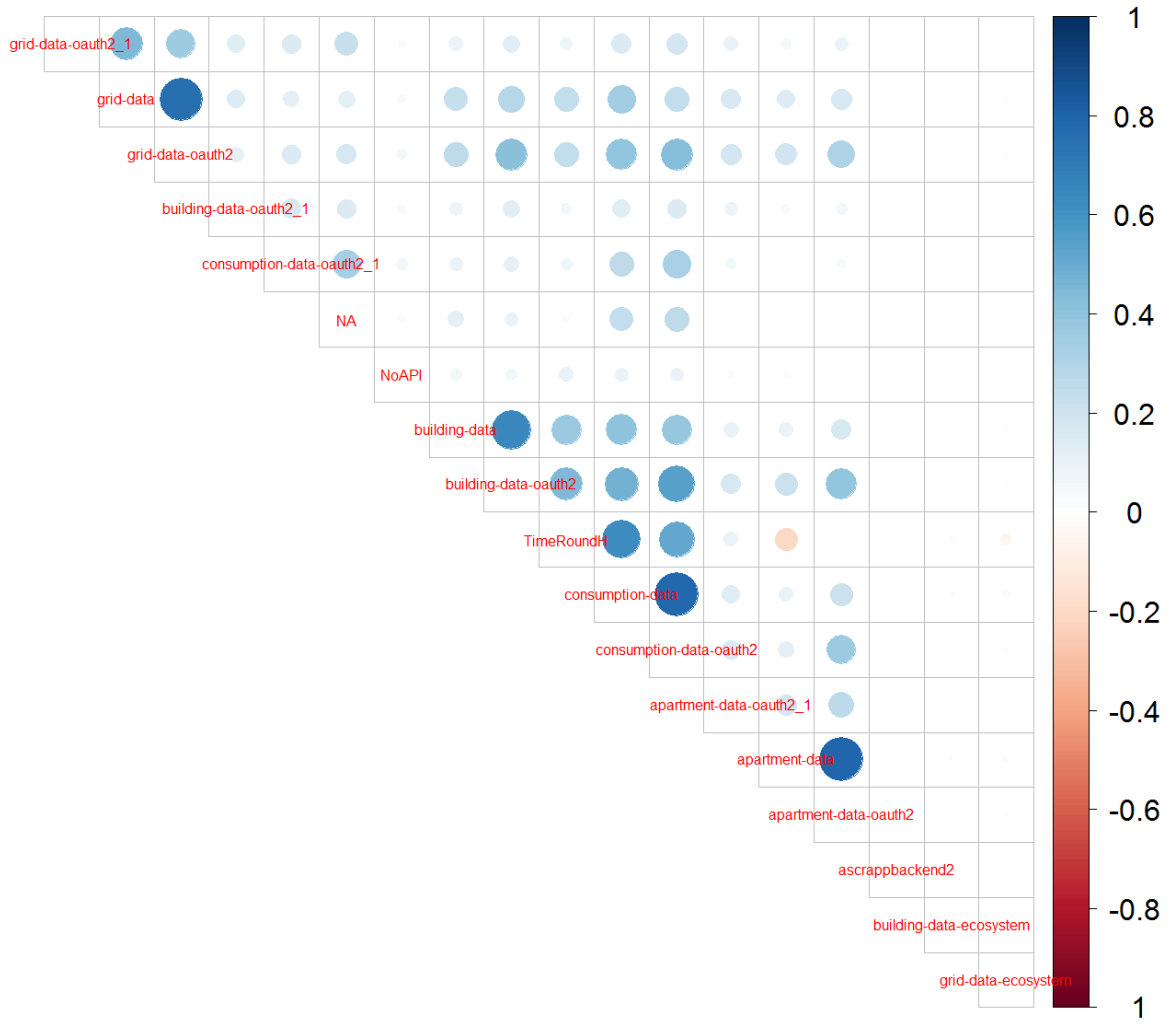
It is interesting to observe rising trends in response durations for different APIs, but then again, there are certain "cut-off" points, different across the APIs and their methods, where the trends seem to *reset*, as displayed in Figure 8. The points are response durations, plotted over time, and the $y-$axis is cropped at 60 seconds. The furthest outliers go up to 9 hours, but they are extremely rare.

Response durations somewhat correlate (Figure 9), mainly because they all depend on the time variable. It is also visible that correlations between the APIs that lay on the same path (in a sense of the request propagation described in Section

2.2, Figure 5)) are higher. These multiple objects, that I consider to be different APIs (because by definition of domain experts they are), sometimes call the same database, which is a bottleneck for the information flow. Databases grow over time, queries of some APIs last proportionally longer, which is seen in the correlation with the time stamp variable.



(a) Boxplots of log transformed response duration per API



(b) Response duration correlation per API

Figure 9: Distributions of request durations

Raising alerts should be based on the averaged values (or some other statistic),

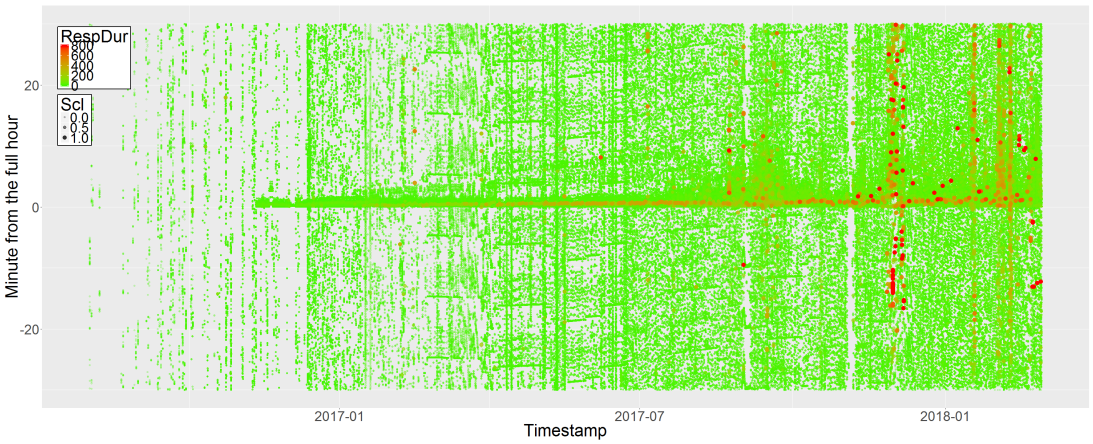
since it's always possible that a single user (unintentionally) generates a complex enquiry which may result in longer operation. Such an isolated event should not be considered alarming, and no action should be taken in that case.

Response duration, when observed per Java method (see Figure 8), also shows interesting changes (**breaks**) in trends. We would like to be able to detect these trends, as well as those changes automatically. Knowing this, we can also base the alerts on forecasted response duration more reliably, in order to launch an alert, before the response duration goes critical. In this case, since the bottleneck for the response duration is access to the database, and the database is growing linearly over time, the increasing trend in response duration is also linear. This is a good sign, because of simplicity of forecasting models, i.e. it enables raising a *yellow alert* **before** response duration becomes critical. Clearly, a *red alert* could be raised if the actual response duration became critical. Regardless of that, the mere use case can easily be implemented in other SUPERSEDE partners or future users, who also monitor systems' response duration and take actions if a response is too slow (e.g. change resolution in video stream if the network connection shows signs of congestion etc.).

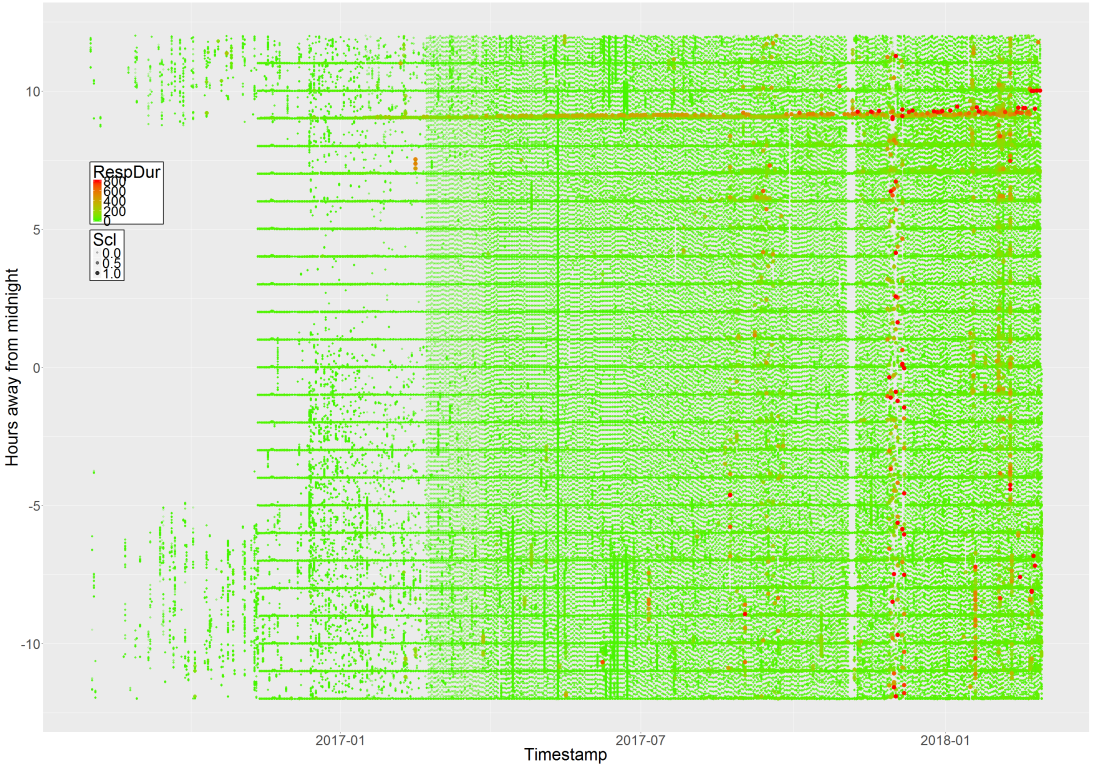
Thresholds for response time are computed by extracting the trends in response duration data and estimating expected range for future measurements. The goal is to raise alerts if a current threshold is exceeded or if it is about to be exceeded. Response times may vary depending on an exact method that is monitored, meaning that each method will have its own dynamic threshold.

Even though this system is developed on a specific use case, the solution is portable, as long as the measured points refer to a signal that shows long-term trends and short-term fluctuations (e.g., response times, network throughput, bit error rate, video frame errors, etc.). This is particularly important for the application of the method into the project use cases.

3.3.1 Descriptives



(a) Distribution of requests and durations over time, during the hour

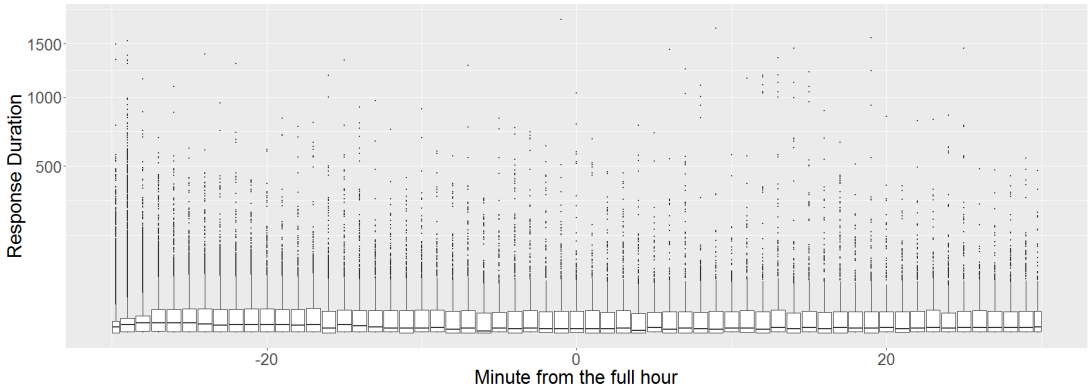


(b) Distribution of requests and durations over time, during the day

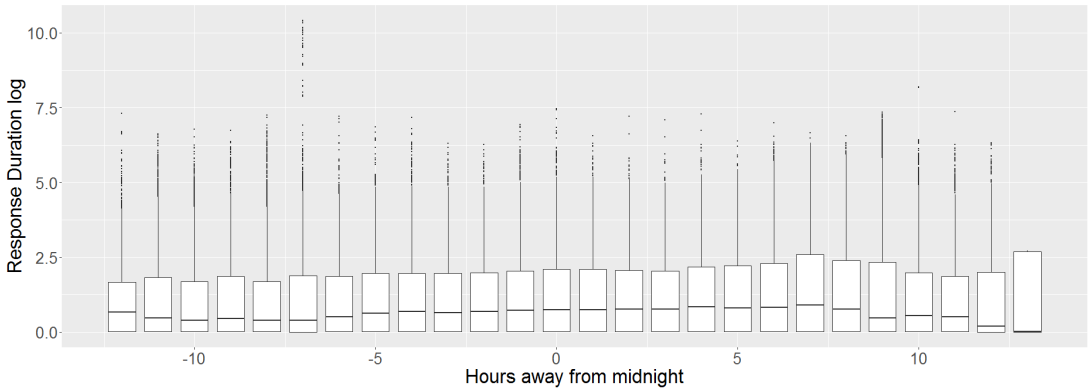
Figure 10: Distribution of requests and durations

As a part of initial analyses of the response duration behaviour, distributions of method calls has been checked per minute, hour, day, etc. We notice an increase in average response duration, as well as in total method calls every hour around a full hour, and every day at 9 AM. This is depicted in Figures 10 and 11, and is a

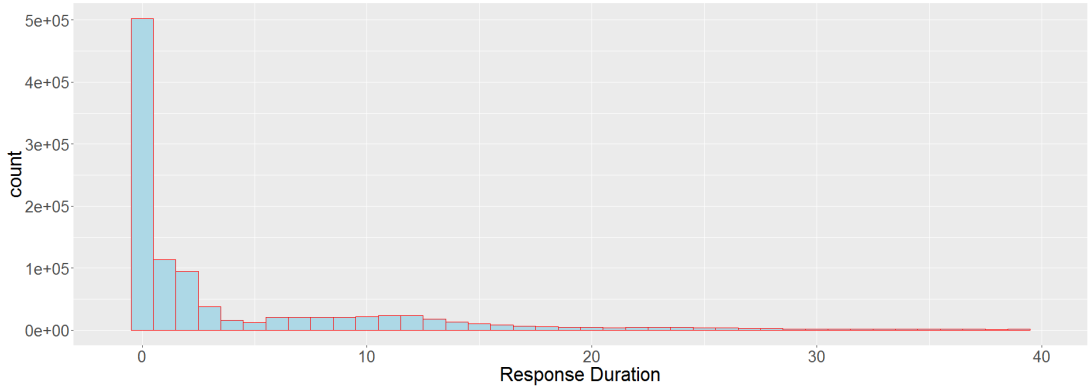
first indication that there could be a periodicity in method calls. This topic will be covered in Section 4.2.



(a) Boxplots of response duration over hour



(b) Boxplots of response duration over day



(c) Histogram of response durations, cropped at 40s

Figure 11: Distributions of request durations

In Figure 10 points represent requests, that come over time ($x-$axis), and on the $y-$axis is difference from the full hour, and midnight respectively. Colour and size of each point represent response duration, for the request that started at that

particular time, and all responses that take 800 seconds or more are of the same shade of red. The maximum amount of response duration is 9.19 hours. It is interesting to see that during the busiest time of an hour, is where the worst values of the response durations lay. This is visible as horizontal "lines", but there are also several days that were especially problematic, where the reddest points form vertical "lines". The effect of hourly influence is however not visible on the box-plots of response duration per hour, or per day (Figure 11). Further changes in the system behaviour are also apparent, e.g. the point where the hourly checks have started (start of the green *lines* in the figure), as well as 10-minute checks (time point where the dots suddenly become more dense).

3.4 Extracting Further Metrics

Since each line of the log file includes internal success evaluation of the event, stored in the **Content** feature, the first and the most obvious thing is to check how do they appear over time. A surge of failures in a short time might be a good indication of problems in the system, network, database etc. Kernel density estimates could be proven useful for this use case as well: for each of the possible values in the **Content** feature, we sub-select them from the data and *count* their appearances in the given time window (by using KDE and adjusting the *bandwidth* parameter - see Section 4.2.1). The higher (*broader*) the bandwidth, the *smoother* the resulting function. An example of that can be seen in Figure 19. Red markers at the bottom indicate appearance of an event, and the curves are estimating the probability distribution based on the observed events. In this case it is done with a Gaussian kernel estimator. Since the formal output is a probability distribution function, the y scale is defined so that the function integrates to 1. This condition is, however, not necessary for this application, since the gotten numbers will be scaled and weighted anyway in future steps.

This is one possible way to convert discrete categorical variable into a continuous, numerical one. Instead of working with original nominal values, for each of the levels, we are working with the probability of its appearance in a certain time, basing these probabilities on real observations.

4 Data Analysis

By this point, we already addressed some of the problems defined in Section 1.3.

A way to tackle the problem of changing *linear* trends (as described in Section 3.3) are structural break tests.

I will go around the categorical features *and* random times of the events **both** with kernel density estimation, as announced in Section 3.4. I will also use KDE to automate the periodicity detection of the methods (see Section 4.3), and to re-imagine anomaly detection in detection of a deviation from the periodicity, since the structural break tests do not seem to be powerful enough in this use case.

4.1 Structural Breaks Tests

A structural break is any change in time series that may cause errors in prediction. E.g. it occurs when linear regression coefficients aren't independent of time. This kind of break may happen in various ways - the slope may change or the intercept may change, while variance either changes or stays the same. We can know the time of the break, or it can be unknown.

We can group hypothesis of breaks in several different categories:

- a single break in mean with a known breakpoint: we use it to test for a break on the historic data,
- a known number of breaks in mean with unknown breakpoints,
- an unknown number of breaks in mean with unknown breakpoints: will be useful for automation of testing of the future data,
- breaks in variance.

in Figure 12 there was only one break at the point $x = 2$. On second subfigure we see an example of joined data coming from two different programs, where again the Chow's test as defined in equation (7) can be used to confirm the existence of two intercepts.

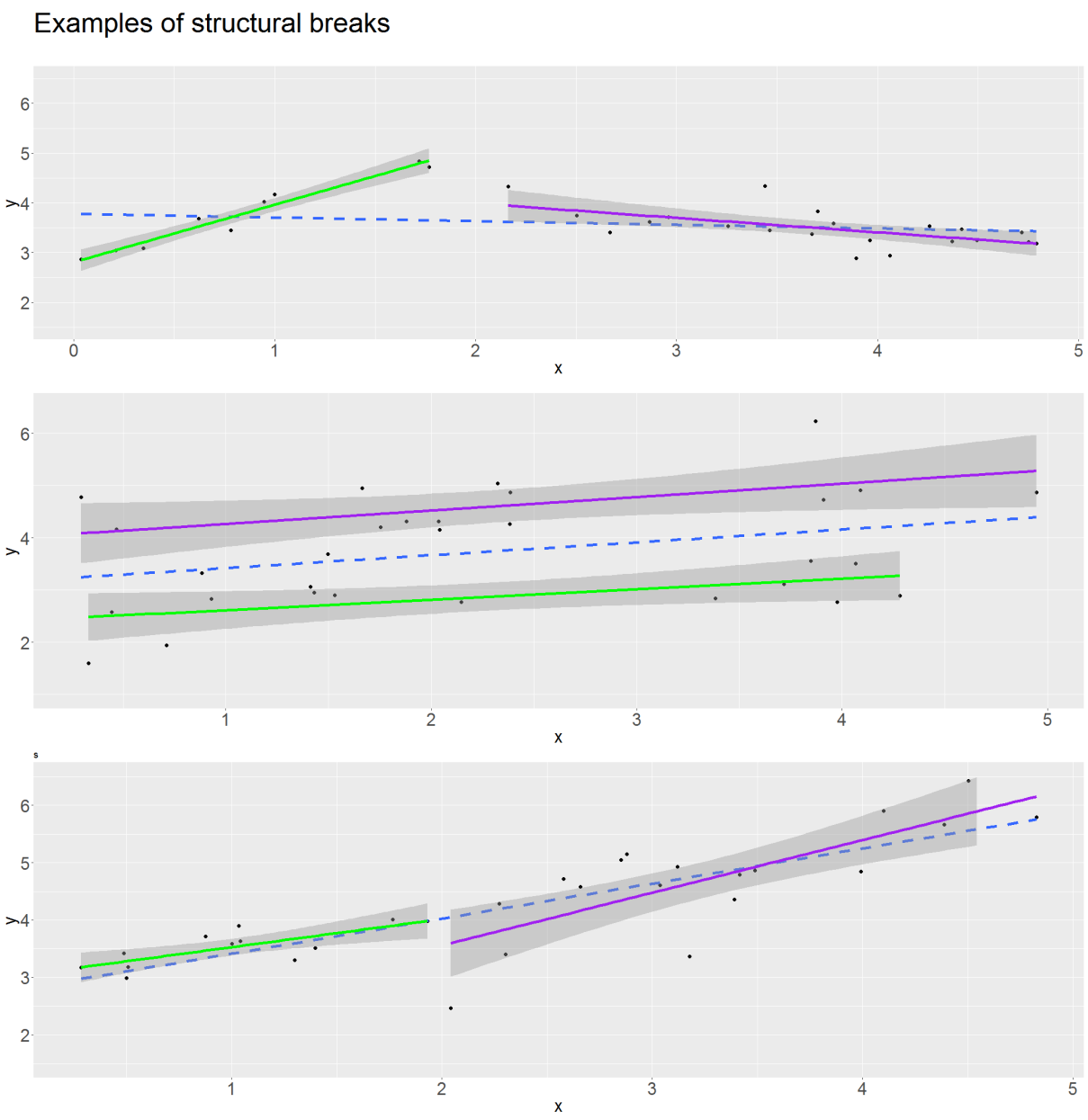


Figure 12: Artificial examples of structural breaks

4.1.1 F-Tests

Consider a general setup: linear regression model for which we assume that a break in one or more parameters occurred in period τ:

$$\begin{aligned} y_t &= x_t'\beta_1 + \varepsilon_t, \text{ if } t \in [1, \tau] \\ &= x_t'\beta_2 + \varepsilon_t, \text{ if } t \in [\tau + 1, T] \end{aligned} \tag{3}$$

or, more compactly:

$$y_t = x_t'\beta_1 + \mathbb{I}_{\{t\leq\tau\}}x_t'\gamma_t + \varepsilon_t, \text{ if } t \in [1,T] \tag{4}$$

where

$$\begin{aligned} \mathbb{I}_{\{t>\tau\}} &= 0, && \text{if } t \leq \tau, \\ \mathbb{I}_{\{t>\tau\}} &= 1, && \text{if } t > \tau, \\ \gamma &= \beta_2 - \beta_1. \end{aligned}$$

Assume that the x's are weakly exogenous and the ε are homoskedastic and not autocorrelated. Under additional regularity conditions regarding the joint x and ε process, the OLS (ordinary least squares) estimator of 4 is consistent, asymptotically normal and asymptotically efficient.

Tests of structural change are testing the null hypothesis of no structural change, meaning the coefficients are stable, vs. the alternative that the coefficients are time dependant.

$$\begin{array}{lcll} H_0 : \gamma = 0 & & \beta_t = \beta_0 & \forall t \in \{1,\ldots,n\} \\ H_1 : \gamma \neq 0 & \text{or} & \exists t_0, t_1 : \beta_{t_0} \neq \beta_{t_1} & t_0, t_1 \in \{1,\ldots,n\} \end{array}$$

The first such a test was proposed by Chow [11] in 1960, as a test of whether the coefficients on different data (sub)sets were equal. It is most commonly used in the analysis of time series to test for the presence of a structural break at a known time τ.

Let S_C be the sum of squared residuals from combined data, S_1 be the sum of squared residuals from the first group, and S_2 be the sum of squared residuals from the second group. N is the total number of observations, k is the number of parameters. The Chow's test statistic [11] is:

$$S_C = \varepsilon'\varepsilon \tag{5}$$

$$S_j = \varepsilon_j'\varepsilon_j, \; j = \begin{cases} 1, & \text{for } t \leq \tau, \\ 2, & \text{for } t > \tau \end{cases} \tag{6}$$

$$F = \frac{\frac{S_C - (S_1+S_2)}{k}}{\frac{S_1+S_2}{N-2k}} \sim F(k, n-2k) \tag{7}$$

The test is, however, only powerful in the situation where the breaking point is known, and unique, while the residual variance remains constant before and after the break.

Over the years, several generalizations of the test have been proposed. A brief overview can be found in Table 2.

Quandt [23] first proposed the sup F test as the likelihood ratio test for detecting structural change with an unknown breakpoint. The idea is to compare the Chow's F statistic over all possible breakpoints t_i, and take their supremum. Although the idea is simple, its optimality properties, as well as the asymptotic distributions have only been shown 33 years later by Andrews [5]. This test, as well as some others, mostly assume homoskedasticity.

Maasoumi et al. [20] developed the likelihood based MZ test for simultaneous change in regression coefficients and error variances at a fixed and known breakpoint.

A natural generalization is the sup MZ test, which compares the MZ scores over all possible breakpoints. According to Ahmed et al. [4], it can be used to test for a single unknown simultaneous break in mean and variance. The authors compared it to the sup F test proposed by Quandt [23], concluding that the loss of power by testing for structural change in variance is negligible, while the gain in power under heteroskedasticity is huge [4]. However, their proof was a series of Monte-Carlo simulations, followed by an empirical example.

		Assumptions		
Test	Comment	Breaking point	Variance	Source
F		known	homoskedastic	Chow [11]
sup F		unknown	homoskedastic	Quandt [23], Andrews [5]
MZ	simultaneous change in regression coefficients and error variances at a known point	known	heteroskedastic	Maasoumi et al. [20]
sup MZ	proof by Monte-Carlo simulations	unknown	heteroskedastic	Ahmed et al. [4]

Table 2: Summary of the development of the structural break tests

All of the tests presented in Table 2 assume that the data is linear, and that there is only one breakpoint. There are further tests that are based on generalized fluctuation tests as proposed by Leisch et al. [18], that are able to detect various types

of structural changes. Some of these approaches and tests have been implemented in R package `strucchange` by Zeileis et al. [26].

Structural break tests for non linear models is a subject that hasn't been thoroughly studied yet. Kapetanios [17] relies on neural networks to approximate the conditional expectation of dependent variable, and testing the residuals with standard residual based structural break tests as described in Section 4.1.1. They offered a Monte Carlo proof. Hoyo et al. [16] propose a more advanced approach for testing of parameter consistency of non-linear models, as well the asymptotic properties for their method. This will, due to unnecessary, not be covered in the scope of this thesis. However, it may be relevant for partners where response duration doesn't show linear trends.

4.1.2 Generalized Fluctuation Tests

Fluctuation tests can be based on estimates (mostly OLS) or on residuals.

Tests based on estimates rely on the idea that a structural change would be manifested in the measurable difference between regression coefficients of the whole data and the regression coefficients of a subset of the data. If there are no structural changes, the coefficients remain (almost) constant over time. An empirical process can here be derived by taking the differences between subsample estimates and the overall estimate. The subsamples are either chosen recursively, meaning that they start with the first k observations, and including the coming observation stepwise, or by a moving constant-width window.

Under the null hypothesis the processes should not stray too far away from zero. Asymptotic properties of these processes are known [18], meaning that the bounding processes can be calculated, which are only crossed with a desired probability α. If an empirical process fluctuates heavily and crosses the boundaries at a specific time, there is enough evidence that a structural break occurred at that time. In this case, recursive estimates process will have a peak around the breakpoint, whereas the moving estimates (ME) process would have a strong shift [26]. For simplicity, we apply the principle of the structural break tests on a sensor data of yearly temperature averages in New Haven, CT. It can be seen in Figure 13.

Fluctuation processes can be calculated based on cumulative or moving sums

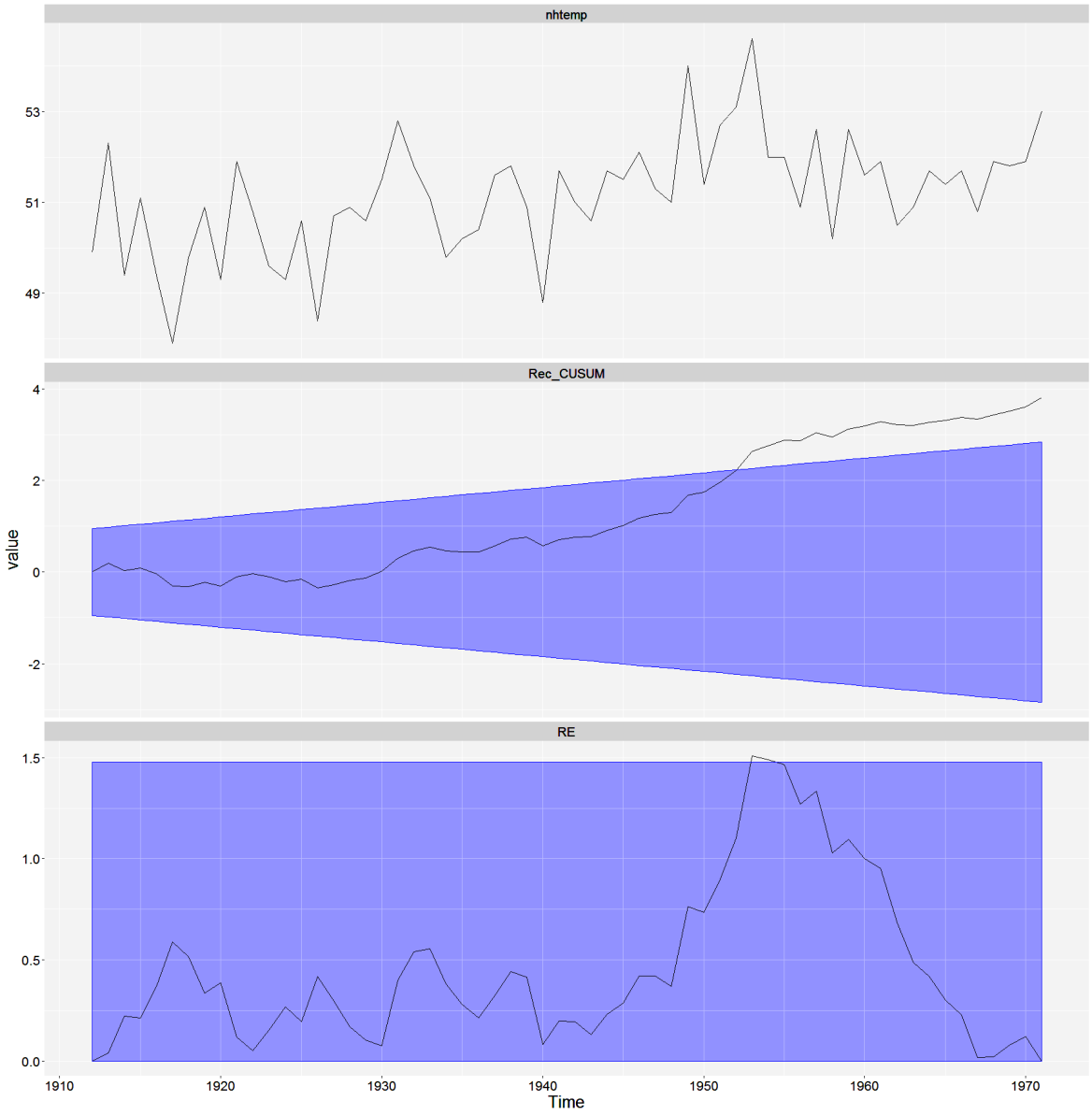


Figure 13: An example of structural break detection on the New Haven temperature data

(abbreviated: CUSUM or MOSUM respectively) of either the OLS-residuals or recursive residuals (one step forecasting errors).

The CUSUM test, as proposed by Page [22], Brown et al. [8] is based on the cumulative sum of recursive residuals. If there is a change in the structure, the process will leave its zero mean around the breakpoint, because the one step forecasting errors would grow. The OLS based tests have similar properties to the estimates-based processes and under the alternative of a unique change point, the OLS-CUSUM would show a peak, while the OLS-MOSUM would show a shift

around the change point.

By using the `strucchange` package for **R** Zeileis et al. [26] we implemented a methodology to address the following tests:

- Rec-CUSUM
- OLS-CUSUM
- Rec-MOSUM
- OLS-MOSUM

The `efp()` function from `strucchange` takes as input a signal and returns a one-dimensional empirical process of sums of residuals. The process can either be based on recursive residuals or on OLS residuals and it will contain cumulative sums or moving sums of residuals in a certain window based subset of the data. For the processes based on moving sums, all estimations are done for observations in a moving data window, whose width is determined by h and which is moving over the whole sample [27].

For the types:

- RE
- ME

a k-dimensional process will be returned, if k is a number of regressors (features) in the model, as it is based on recursive OLS estimates of the regression coefficients or moving OLS estimates respectively. Recursive estimates test is also called fluctuation test, therefore setting type to "fluctuation" was used to specify it in earlier versions of `strucchange`. It still can be used now, but will be forced to "RE" [27].

And for the types:

- Score-CUSUM
- Score-MOSUM

a $k+1$-dimensional process will be returned, one for each score of the regression coefficients and one for the scores of the variance. The process gives the decorrelated cumulative sums of the ML scores (in a Gaussian model) or first order conditions respectively (in an OLS framework) [27].

If there is a single structural change point τ, the recursive CUSUM path starts to depart from its mean 0 at τ. Brownian bridge type paths will have their respective

peaks around τ. Brownian bridge increments type paths should have a strong change at τ [27].

There is a `plot()` function for the output objects of `efp()`, i.e. a class `efp`, that plots the process' path together with the corresponding boundaries, that defaults to $\alpha = 0.05$. The boundaries may be extracted separately by using a `boundary()` function. The most relevant function is the significance test, which returns a p-value, is implemented under `sctest()` [26, 27].

4.1.3 Use Cases

Focus of my analyses will mainly lie on various aspects of response duration and periodicity of some of the methods. As described in Section 3.4, it is possible to extract more metrics from the logs, as well as from user behaviour (by including the feedback data), but they will not be considered in the scope of this thesis.

4.1.3.1 Structural Break Based Alerts

Before we begin with response duration modelling, we need to focus on observed trends in response duration, and breaks in their structure. Response duration data has linear trends, but they break, and before we use it for forecasting, we need to address this. This will be an alert for itself, as well as a base for better forecasting of response duration (see Section 4.1.3.3).

For validation we apply statistical tests described in Section 4.1.2 on method `/consumption-data/comparison`. The data is split in weekly subsets (*windows*) and structural break tests are performed every week. In case of no breaks, the window was appended by the data from the coming week, and the test was performed again. In case of a structural break, the data before the break was removed, meaning the starting period for the next test is the last (tested) structural break, and the process repeats. The reason for this is that none of the tests were UMP (uniformly most powerful) for testing of hypotheses of more than one breaks in the data, so this way we make sure that, per each test performed, there is at most one structural break in the data. It implicitly enables approximation of a time point of structural breaks, even for the tests that do not return it by default.

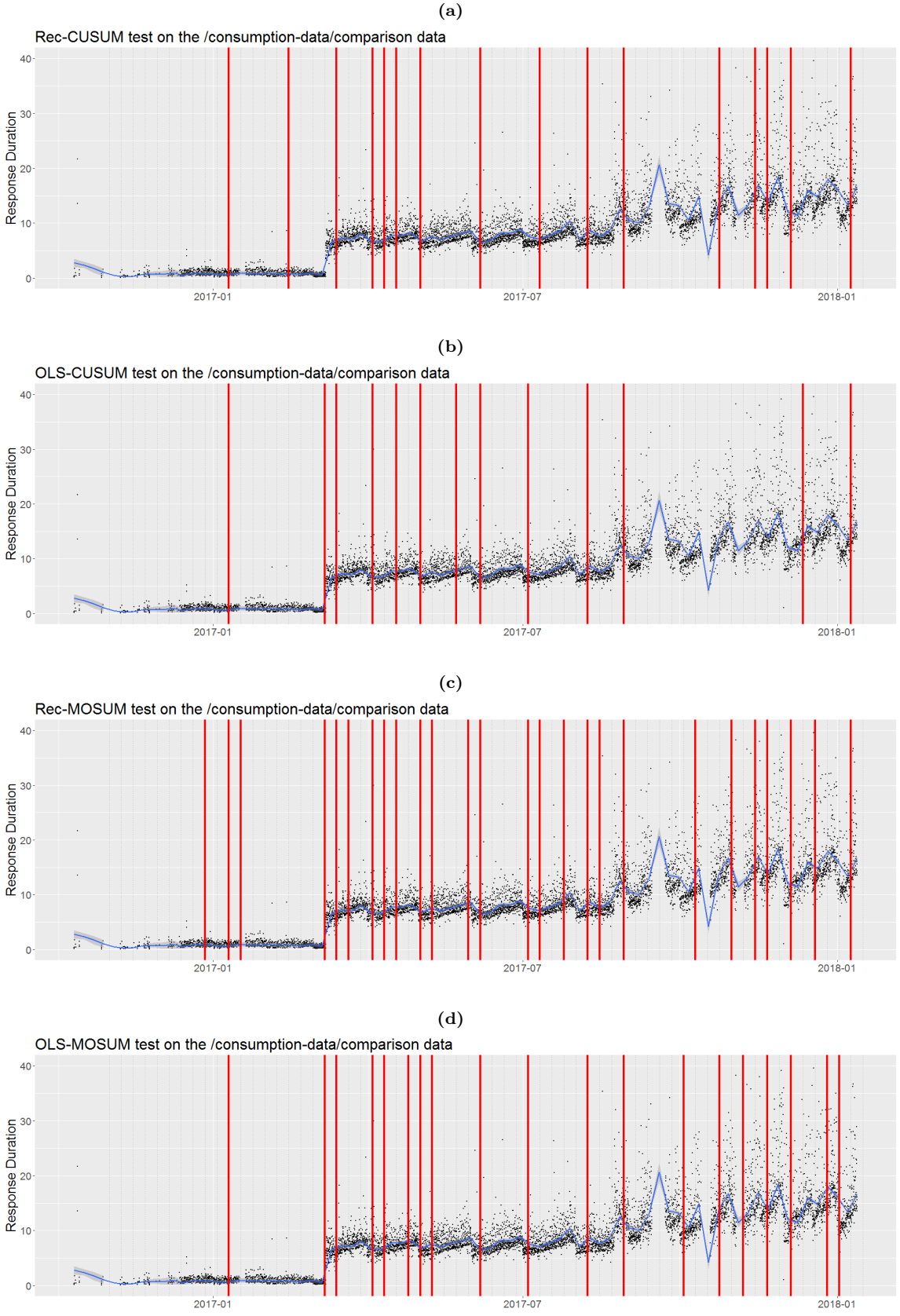


Figure 14: Alerts raised by the structural break tests for the `/consumption-data/comparison` method

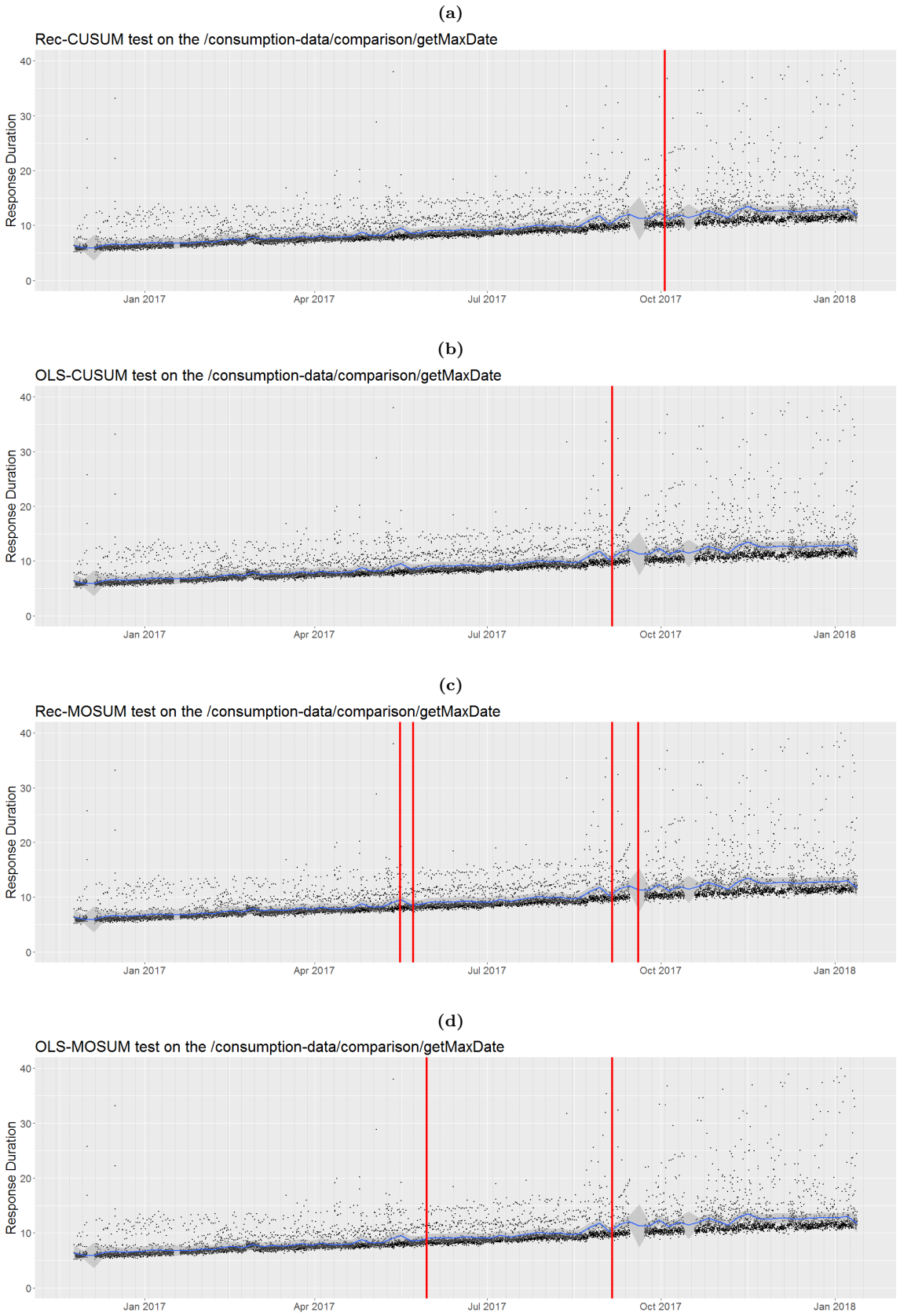


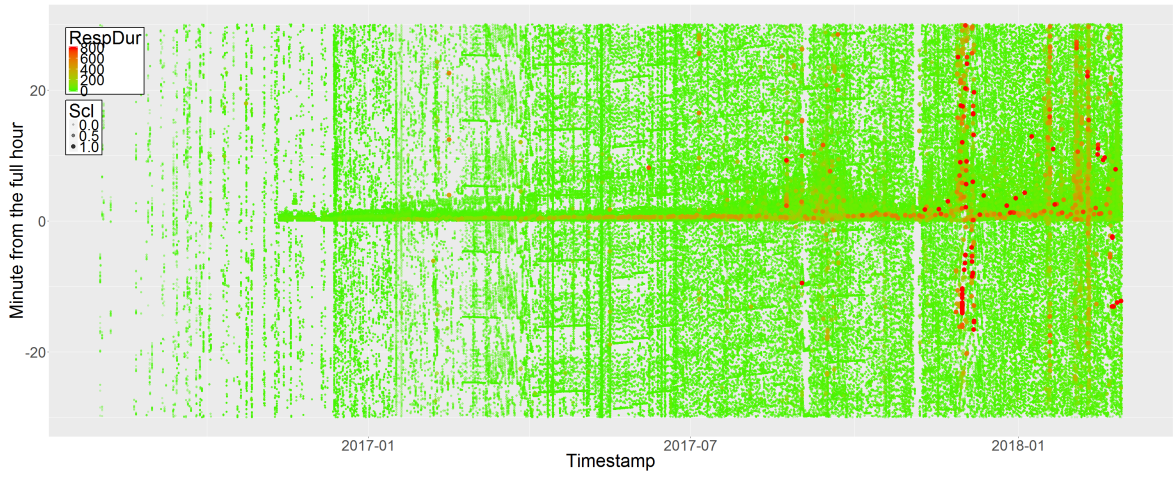
Figure 15: Alerts raised by the structural break tests for the `/consumption-data/comparison/getMaxDate` method

Tests from the function `strucchange::efp()` were performed at $\alpha = 0.05$ level of significance. $\alpha = 0.1$ was tried out, while output for some of the tests wasn't very different, for others it was too sensitive. From Figure 14 it is clearly visible that only a couple of weekly delimiters were detected as structural breaks by all of the tests at this level of significance. It is interesting to note that none of the tests gave exactly the same output. Additionally, a cold-start phenomenon was present: before working reliably, most of the tests needed some time (i.e. more data) to gather enough evidence (and reduce the $p-$value enough) in order to successfully reject the null hypothesis of no structural change.

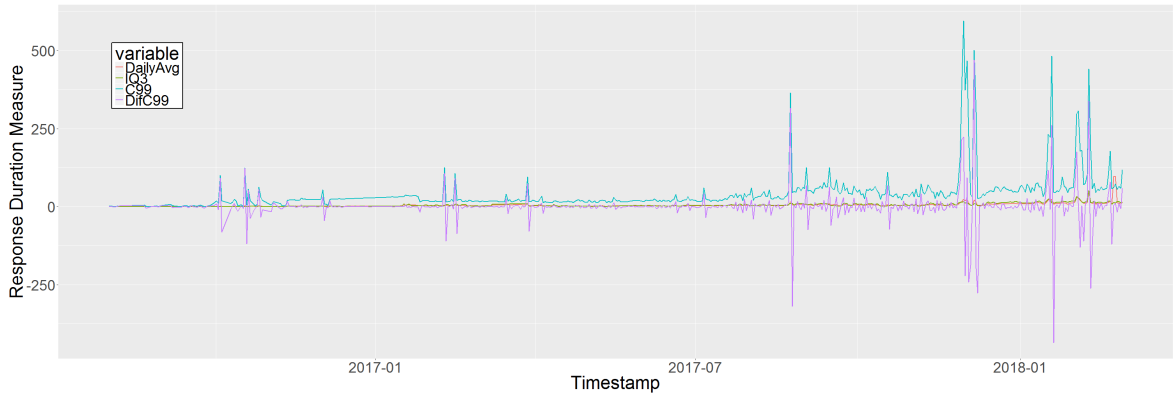
To see how the tests behave on a method with no apparent structural changes, a series of *benchmarking* tests was performed on the method `/consumption-data/comparison/getMaxDate`. Results are depicted in Figure 15. Here, as expected, less break points are detected. The detected break points may be caused by change in variance, which is less obvious to the naked eye than the change in intercept or slope.

4.1.3.2 Response Duration Based Alerts

Raising an alarm is only possible when having a predefined threshold. This is also something we would like to have automatised, rather than having to define it manually, as covered in Section 3.3 It is only then possible to check if the trends seem to cross the threshold in near future, which will be covered in the next section.



(a) Distribution of requests and durations over time, during the hour



(b) Daily measures of response duration

Figure 16: Extraction of metric

Due to the high amount of requests, most of which being processed fairly quick, averaged daily/ hourly response duration is not a good enough measure to detect bad time windows. We need a measure that would be more sensitive to outliers, but not too sensitive, since they do last up to 9 hours. In Figure 16b daily average, daily 3^{rd} interquartile and daily 99^{th} percentile are compared. There is an indication that these could be a better statistic than the mean, because they capture the skewness of the data better. Since we want to be able to detect a sudden change in its behaviour, we differentiate the daily 3^{rd} interquartile, and its peaks do seem to capture well the most critical days of the overall response duration. This means that this measure is good enough to base our threshold computation on, and it enables staying tuned with changes in the system, while eliminating the need of having to define thresholds manually. We do however set the boundaries for such

a limit, not allowing it to fall less than 7 sec, or rise above 60 sec, meaning - if the dynamically extracted threshold is less than 7 seconds, do not raise alerts, and if it's above 60 seconds, raise it unconditionally (independent of the system's state).

We replayed the historical data collected during the project, to test proper functioning of threshold computation. The data refers to the response time of an API that shows heavy short-term fluctuations. Figure 17 shows response time signal in a test period (second half of February 2018). Black lines connect the hourly mean response duration values. The blue line connects threshold values. Green line in the bottom is the lower bound for response duration, manually set to 7 seconds, meaning we would not react to thresholds that are lower than 7 seconds. If the hourly based metric exceeds the extracted thresholds, an alarm is raised, and marked with the red cross on the x-axis. We see that there are less long-term fluctuations of the signal, causing the dynamic threshold to stay almost constant. It is interesting to note that in this case the alarms seem to occur regularly every day. In Subfigure 17b we plot the occurrence of alarms by time-of-day. This plot uncovers another phenomenon: most of the alarms occur between 6 and 9 am. The reason for this is that, in this time window, maintenance works in the databases are active, leading to overall decrease of system performance. This shows that dynamic thresholding is robust against regular maintenance windows and continues to provide proper alarms. It is also interesting to notice that for some days (towards the end of February), alarms are raised for almost every hour. The reason for this is that the hourly averages are higher than in the weeks before, and that even though the threshold is adapting, alarms are still raised.

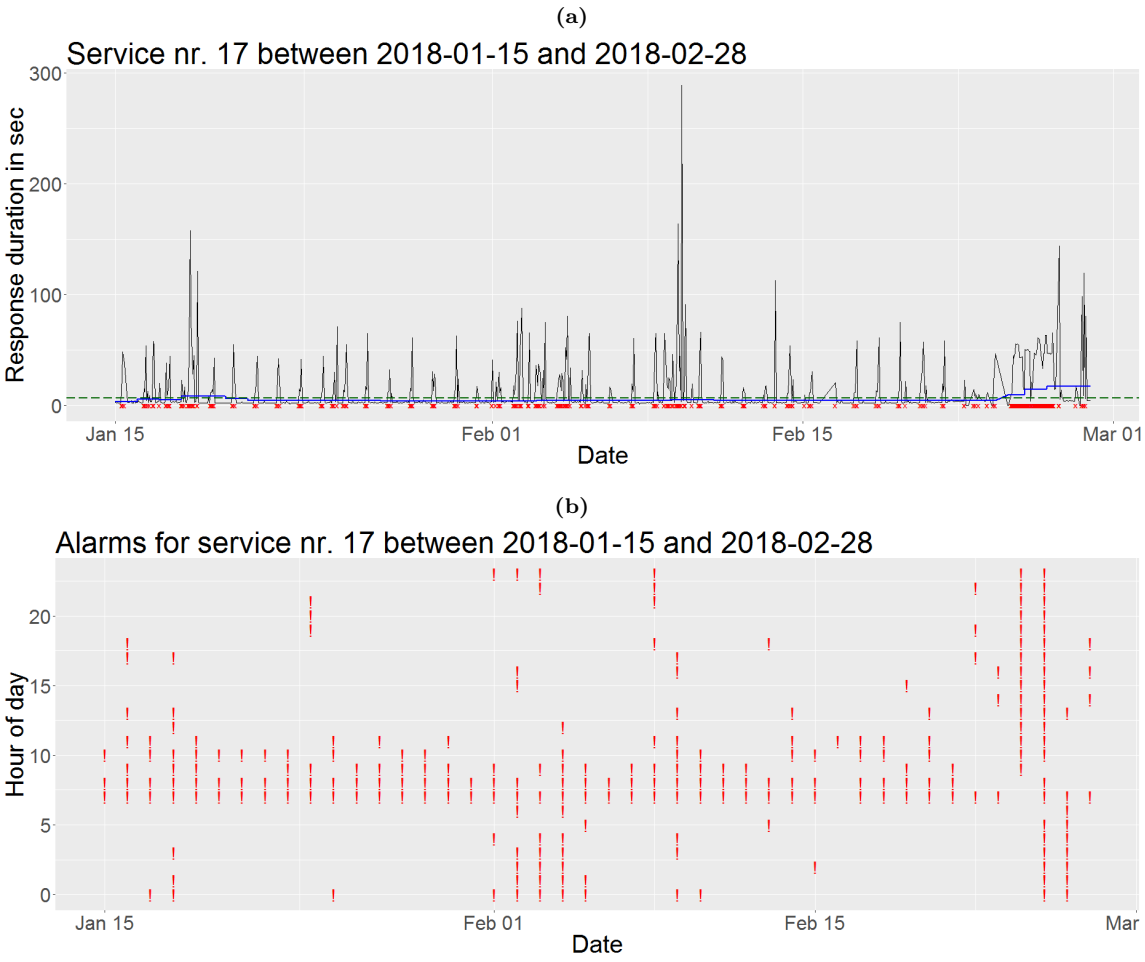


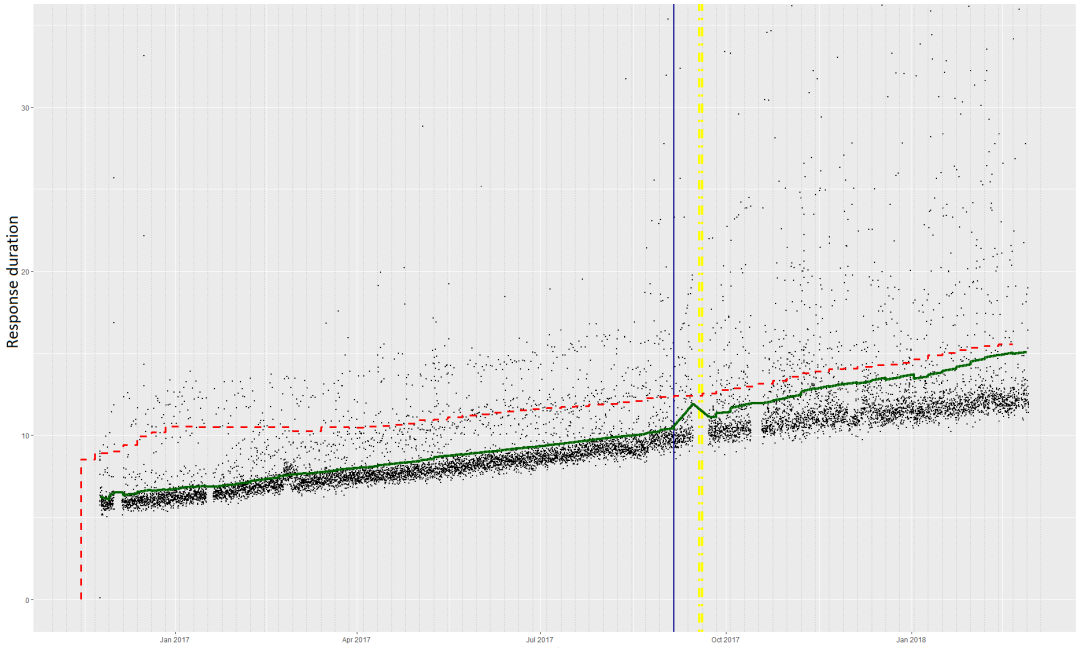
Figure 17: Alerts raised by the too slow local response duration

4.1.3.3 Predictive Alerts in Response Duration

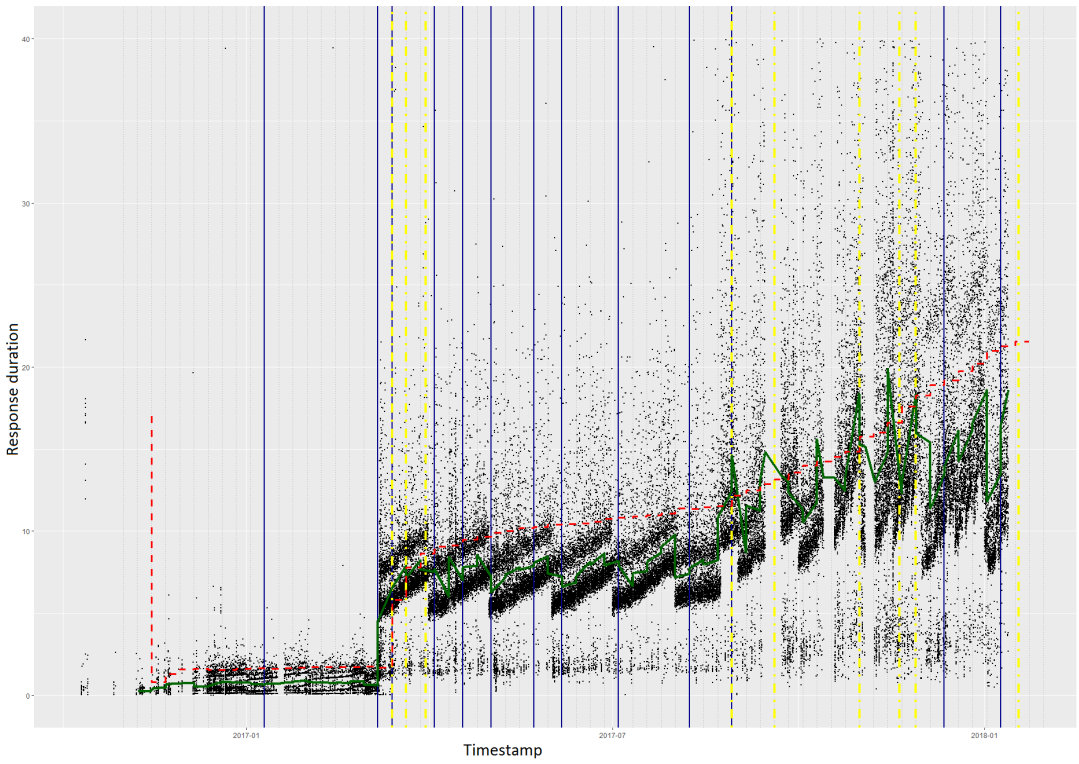
A level of complexity in forecasting is added by the found structural breaks (covered in Section 4.1.3.1), so the linear models used to capture trends had to be updated on a weekly basis too, while only considering data since the last break (because the points before it do not belong to the same trend, and would only deteriorate the forecasting accuracy).

For each of the weeks in validation, a prediction of the linear trend was compared to the quantile based threshold. In case the trend would *hit* the threshold in the coming time window, an alert was raised *in advance* (on the weekly cut-off points – where it was calculated), allowing for a preventive action to be taken.

The whole process can be seen on one example in Figure 18, where the methodology was applied on the two methods that show very different behaviour of the response duration trends.



(a) `/consumption-data/comparison/getMaxDate` method



(b) `/consumption-data/comparison/` method

Figure 18: Alerts raised by the forecasting of the response duration for the who methods

Blue lines represent structural breaks detected by the OLS-CUSUM test (as presented) with the level of significance of $\alpha = 0.05$. Red line shows the dynamically

updating threshold for response duration. It grows at a slow pace, allowing for a *reasonable* increase in response duration (which depends on the growing databases, as described in Section 3.3). If the sub-trends (green line) suddenly shows growth that is too steep, it would cross the automatically derived thresholds in the coming time window and an alert would be raised in advance.

In first case, since the OLS-CUSUM test detected a structural break (possibly falsely), and the next trend is calculated from that point onward, there are less points which makes it easier to influence the slope of the linear model, making it steep and raising an preventive alarm at that point. However, one false positive is considered better than missing out of one true positive - it is better to be too sensitive than not being sensitive enough.

In the second case we show that the proposed system raises more alarms, on the right spots.

4.2 Detecting the Periodicities

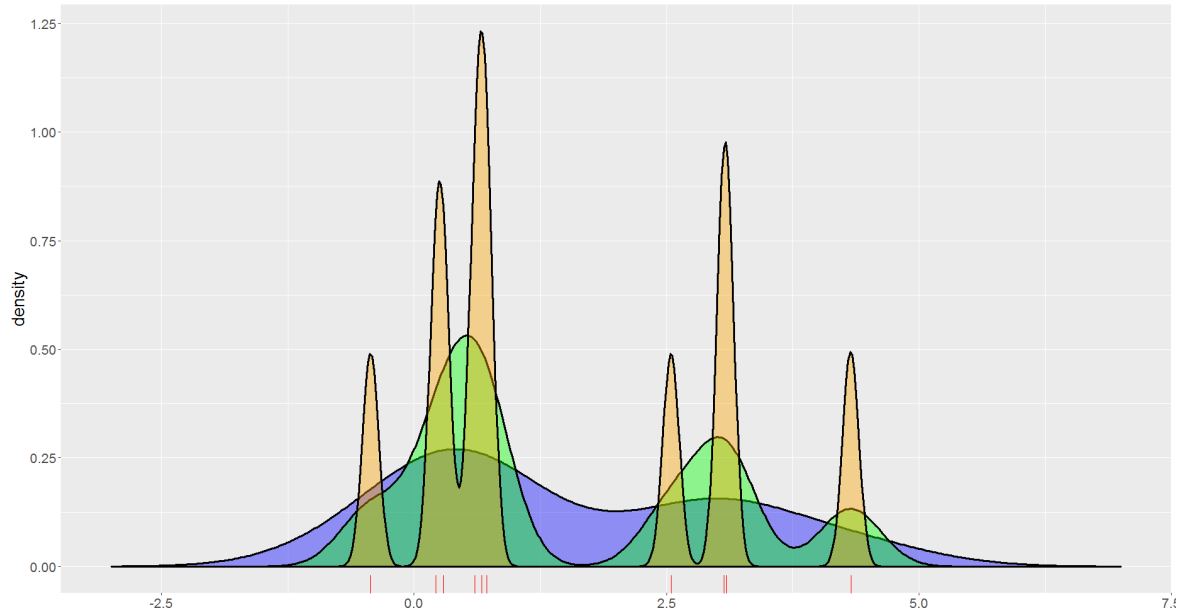
4.2.1 Kernel Density Estimation

Kernel density estimation (**KDE**) is a non-parametric method to estimate the probability density function of a certain random variable. It belongs to *non-parametric statistics*, because instead of assuming the distribution of a random variable, and then estimating the parameters, it estimates the distribution function from the data directly. It can be interpreted as a continuous form of a histogram.

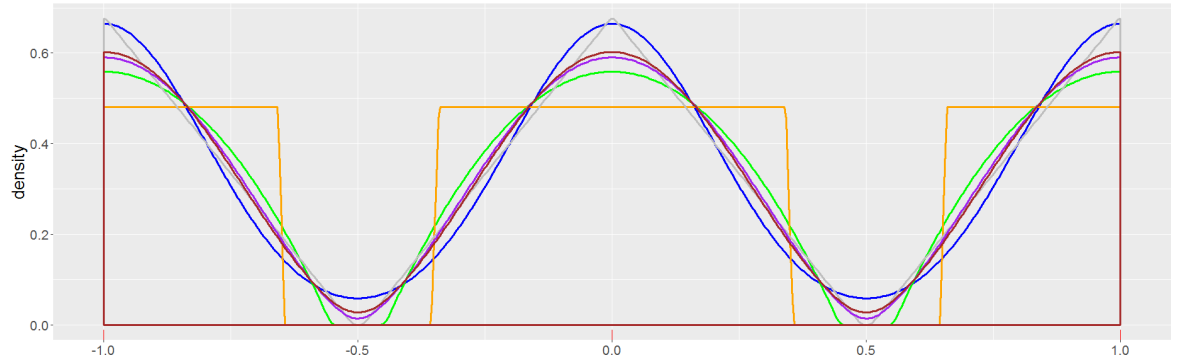
Let $(x_1, x_2, \ldots, x_n)$ be a univariate *i.i.d.* sample taken from a distribution with an unknown density function f. Its kernel density estimator is:

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \tag{8}$$

where K is the *kernel*, a non-negative function that integrates to one, and $h > 0$ is a smoothing parameter called **bandwidth**. There are several options to choose from for both the kernel (some of them are presented in Table 3) and the bandwidth [24], highly depending on our questions of the data. The impact of the choice of bandwidth and kernel to the out-coming function is presented in Figure 19.



(a) Various bandwidths for the Gaussian kernel on a simulated example



(b) Various kernels

Figure 19: Kernel density estimation example

Kernel function	Definition	Support
Uniform	$K(u) = \frac{1}{2}$	$\lvert u\rvert \leq 1$
Triangular	$K(u) = (1 - \lvert u\rvert)$	$\lvert u\rvert \leq 1$
Epanechnikov	$K(u) = \frac{3}{4}(1 - u^2)$	$\lvert u\rvert \leq 1$
Quartic	$K(u) = \frac{15}{16}(1 - u^2)^2$	$\lvert u\rvert \leq 1$
Cosine	$K(u) = \frac{\pi}{4}\cos(\frac{\pi}{2}u)$	$\lvert u\rvert \leq 1$
Gaussian	$K(u) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}u^2}$	$u \in \mathbb{R}$

Table 3: Some kernels

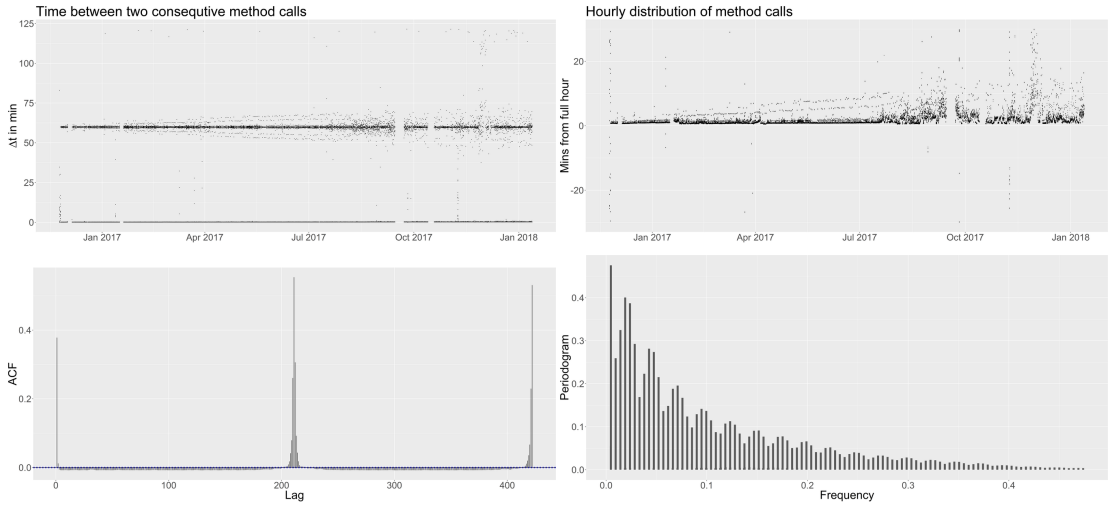
Although they were constructed to estimate the probability distribution of a random variable based on the observed events, I will use them across the analysis to generate continuous signals (curves) from the discrete values. They also represent a good way to go around the issues of random times of the inputs, because of their representation in R. In theory, the output is a continuous function defined on the $\langle -\infty, \infty \rangle$, in R however, they are stored as a set of pairs $(x, f(x))$ with a constant Δx, which we can define, based on our needs. This solves two problems at once – the problem of random x and it quantifies the discrete appearances of a certain categoric event.

4.2.2 Use Case

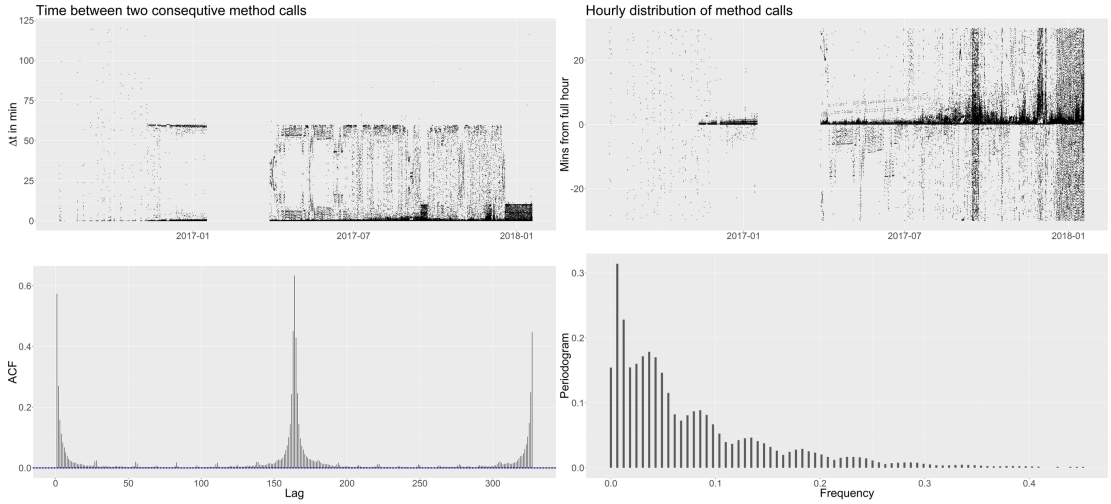
Since the methods are now meaningfully grouped and further clustered in corresponding APIs (see Section 3.1), periodicities in their calls can investigated. We assume, they are mostly (or up to a certain extent) called *on demand*, but there is an indication that there may be some automated processes (e.g. system self-checks), which call the methods on a regular basis (see Figure 10 a) and b)).

I used both Fourier decomposition, as well as autocorrelation function, to test this hypothesis. Fourier decomposition allows an estimation of spectral density of a signal. It is a common method to examine the signal power over frequency, showing if a signal has dominant frequencies. Autocorrelation function however measures the signal's dependency on the previous terms (**lags**). If the signal is an estimation of the probability of event's appearance (and here it is, as covered in Section 4.2.1), the autocorrelation would show peaks at the period(s) as well. Note that both methods require a signal, meaning a continuous numeric feature, rather than a nominal one. Here's where the KDE come in handy.

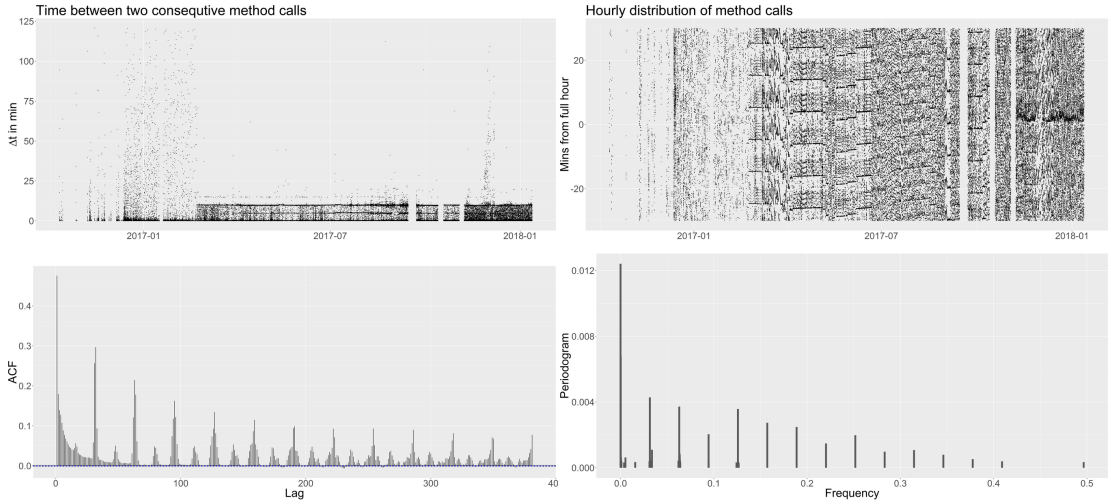
Corresponding plots are visible in Figure 20, and are showing peaks at the expected locations. However, not all methods from the logs show the same behaviour, meaning not all of them have the same periods, or even stable periods - they seem to change (possibly with system's updates or downgrades).



(a) Longer period



(b) Unstable period



(c) Shorter period

Figure 20: Periodicity detection for three selected methods

In Figure 20 periods are evident from the following:

- time difference Δt between two successive method calls is (at most) equal to the period (top left sub-plot),
- functions are being called on every full hour, or at 10 min intervals (top right sub-plot),
- autocorrelation function of the signal generated from the density function shows the most significant lags on the expected places (bottom left sub-plot)
- periodogram shows higher power over the expected frequency (bottom right sub-plot)

Over time, we can see that even the periodicity may vary, i.e. it's not always exactly an hour between the two method calls. Maybe the system is smart enough to compensate for the current system overload by not running the periodic checks when programmed to, but *in a more convenient* time point.

In some cases, periodic checks were completely skipped. In case a system overload is the cause of this, we want to measure these deviations as an indicator that everything is running within predefined parameters.

We will employ kernel density functions as described in Section 4.2.1 to estimate the distribution of Δt as defined above, and we will measure the abrupt changes in that distribution as a sign of disturbances in the system. How this works can be seen in Figure 22.

4.3 Periodicity Violation Based Alerts

Some methods show periodic behaviour, as announced in Section 3.3. Any regularity found in the data that can be measured, can be converted into a rule. A violation of that regularity indicates that something unexpected occurred. We need to be able to detect the existence of a period that changes, as well as quantify when it does not change *per se* but becomes irregular.

4.3.1 Anomaly Detection

One of the simplest ideas in anomaly detection is to analyze the window based sub-sequences of a given sequence in order to derive a set of *allowed* ones [9, 10].

For this we need to define the window length w, suggested to be equal to the (or a multiple of) the period length, if any, and a hop length h, suggested to be less than, or equal to the period. It is obvious that the larger values of w, h would save on the computations, but if we pick them too large, we risk missing out the anomaly.

Let's make an example. Given a series *abcabcabc*, and a window length $w = 3$, the only possible sub-sequences are *abc*, *bca*, and *cab*, regardless of the hop size. However, if we consider a series *abc**c**abcabc*, the ability of the detection of the anomaly depends on the chosen hop size h. In Figure 21 we can see how this process works, and the found anomalies (sub-segments not found in the expected set of sub-segments). It is also visible how choosing the wrong hop size h would prevent the anomaly to be detected.

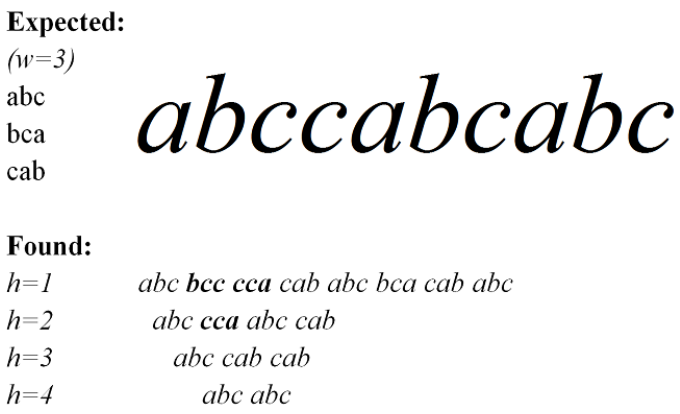


Figure 21: Window based anomaly detection

4.3.2 Modelling of the Deviation of Periodicity

We use Kernel Density Estimator (KDE) to automate the detection of periodicity. Under the assumption that periodic methods would be called more often with the Δt that (at most) equals to the period (in times of user inactivity), the distribution function of Δt would have peaks at the period(s) (see Figure 22).

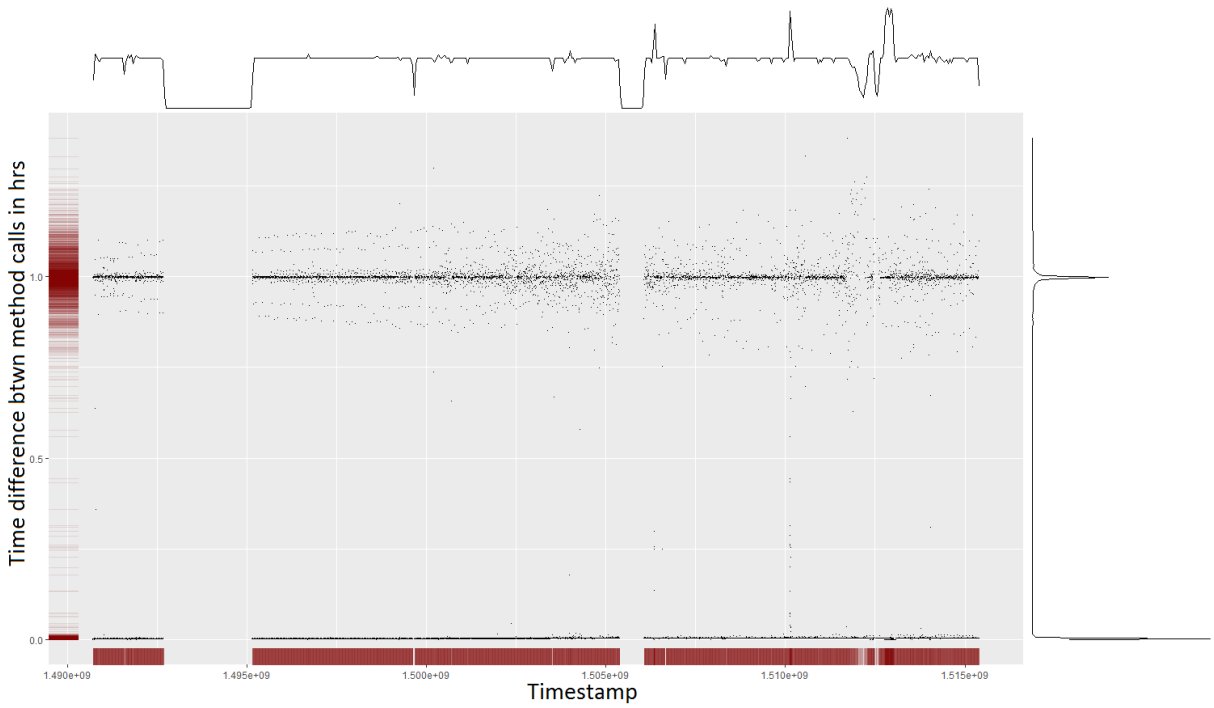


Figure 22: Automated periodicity detection

We use KDE to estimate the probability distribution of requests over the hour, and thereby generating our usable version of the *abcabcabc* sequence, enabling a modification of the proposed window based anomaly detection to be applied here. This distribution changes along with the changes in periodicity of the called methods. A difference of the distribution of requests per hour was calculated for each new week, vs. the last 5 weeks. A measure of difference of distributions was a numeric version of the L^2 distance measure:

$$d(f,g) = \sqrt{\int_{-\infty}^{\infty} (f(x) - g(x))^2 dx} \tag{9}$$

A numeric version because of the way how R stores kernel density functions, as a set of pairs $(x_i, y_i),\ i \in I$, I is a set of indices. We can use the Darboux – Stieltjes [12, 13] inspired approximation of the integral. Let $f : [a,b] \to \mathbb{R}$ be a bounded function, and let $P = (x_0, x_1, \ldots, x_n)$ be a partition of a segment $[a,b]$. Let further:

$$\begin{aligned} M_i &= \sup_{x \in [x_{i-1}, x_i]} f(x) \\ m_i &= \inf_{x \in [x_{i-1}, x_i]} f(x) \end{aligned} \tag{10}$$

The upper Darboux sum of f with respect to P is:

$$U_{f,P} = \sum_{i=1}^{n} (x_i - x_{i-1}) M_i \tag{11}$$

and the lower Darboux sum of f with respect to P is:

$$L_{f,P} = \sum_{i=1}^{n} (x_i - x_{i-1}) m_i \tag{12}$$

The upper Darboux integral of f is

$$U_f = \inf\{U_{f,P} : P \text{ is a partition of } [a,b]\} \tag{13}$$

The lower Darboux integral of f is

$$L_f = \sup\{L_{f,P} : P \text{ is a partition of } [a,b]\} \tag{14}$$

If $\forall \varepsilon > 0$ there exists a partition P_ε of $[a,b]$ such that $U_{f,P_\varepsilon} - L_{f,P_\varepsilon} < \varepsilon$, then we call the common value the **Darboux integral**. We also say that f is (Darboux-) integrable [3] and set:

$$\int_a^b f(x)dx = U_f = L_f \tag{15}$$

Since we do not know the behaviour of the function on the segments $[x_{i-1}, x_i]$, or more exactly, we only know the values of f on bordering points $(x_i)_{i \in I}$, instead of supremum and infimum of the f on the segments, we use the linear approximation, meaning the averages of f on the segments. Given all this, equation (9) in my case becomes:

$$d_{\text{num}}(f,g) = \sqrt{\sum_{\substack{x \in P \\ i \geq 1}} \left(\frac{f(x_{i-1}) + f(x_i)}{2} - \frac{g(x_{i-1}) + g(x_i)}{2} \right)^2 \Delta x} \tag{16}$$

where Δx is constant.

We can control the relative error by choosing higher number of steps n in KDE, thus reducing the Δx. The function's behaviour we cannot control (partially we can – by choosing a broader *bandwidth* parameter), but since our functions $f, g \in \mathbb{C}^1$ are continuous (as defined in (17) – they are a finite sum of continuous and bounded

functions, as defined in Section 4.2.1, equation (8)), they cannot go too wild on the narrow segments $[x_{i-1}, x_i]$:

$$\begin{aligned} &\text{for } f : D \to \mathbb{R} \\ &\forall \varepsilon > 0,\ \exists \delta > 0,\ \forall x \in D : \\ &|x - x_0| < \delta \Rightarrow |f(x) - f(x_0)| < \varepsilon \end{aligned} \tag{17}$$

The error of the numeric integration is then limited by an upper bound:

$$\epsilon \leq \frac{\Delta x}{2} \sum_i \left(f(x_{i-1}) + f(x_i) \right) \tag{18}$$

Joining the approximations from equations (17) and (18) we get the relative error:

$$\epsilon_{rel} \leq \frac{\|\varepsilon_{\text{cont.}}\|_\infty}{|f(x)|} \leq 5 \cdot 10^{-5} \tag{19}$$

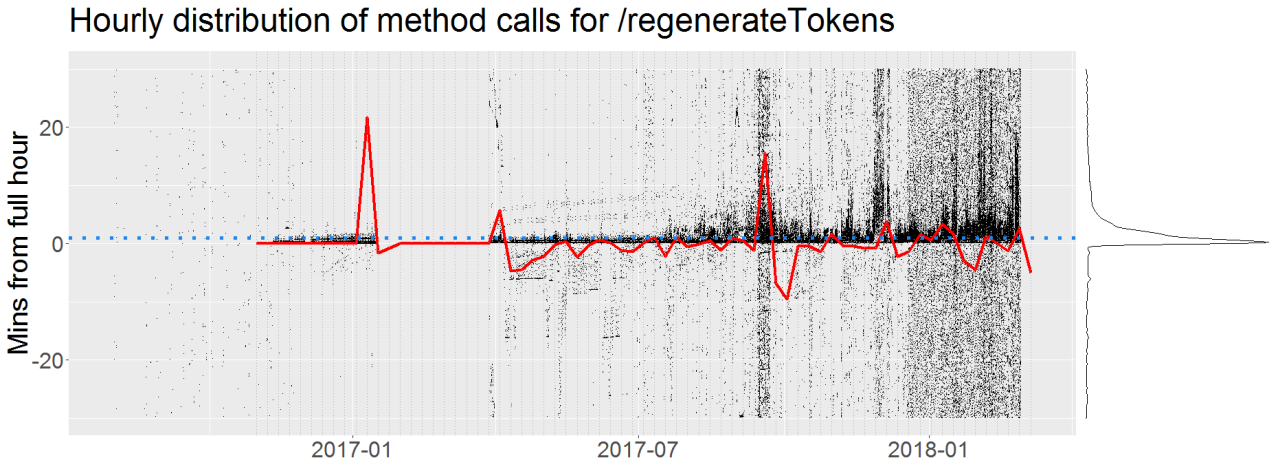
4.3.3 Use Case

Figure 23 shows the hourly distribution of requests of three selected REST services (subfigures a, b, and c, respectively). These services differ in terms of their periodicity. The first one shows hourly and 10-minutely periods. The second one shows exclusively 10-minute periods. The third shows exclusively hourly periods. Each dot represents a method call. The y-axis represents the time the call was recorded (in terms of minutes within the hour). On the right panel we depict the KDE of the hourly distribution of calls for the whole history. Red line represents the **change** of the distribution of the newest calls compared to history (differential). Peaks in the red line imply changes in periodicity (or irregularities). The system detects these peaks by comparing them with a threshold, represented by the blue line.

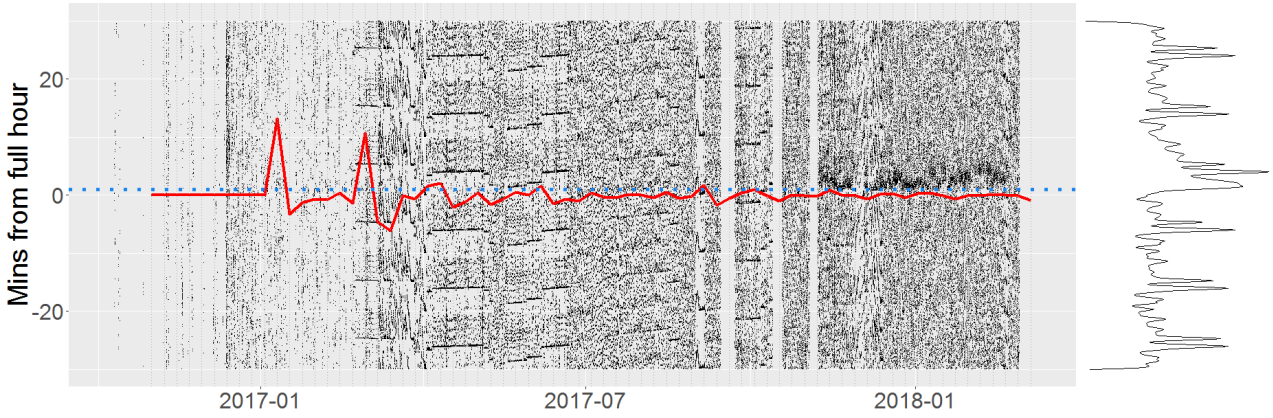
We verified that the proposed measure reacts properly to the changes of periodicity. Obviously, application of such a methodology is only meaningful in case of periodic service calls.

Note that irregularities that can be found with this method are complementary to the one detected with structural break tests. We verified that structural break tests are not statistically powerful enough to detect irregularities in periodicity.

(a)



(b)



(c)

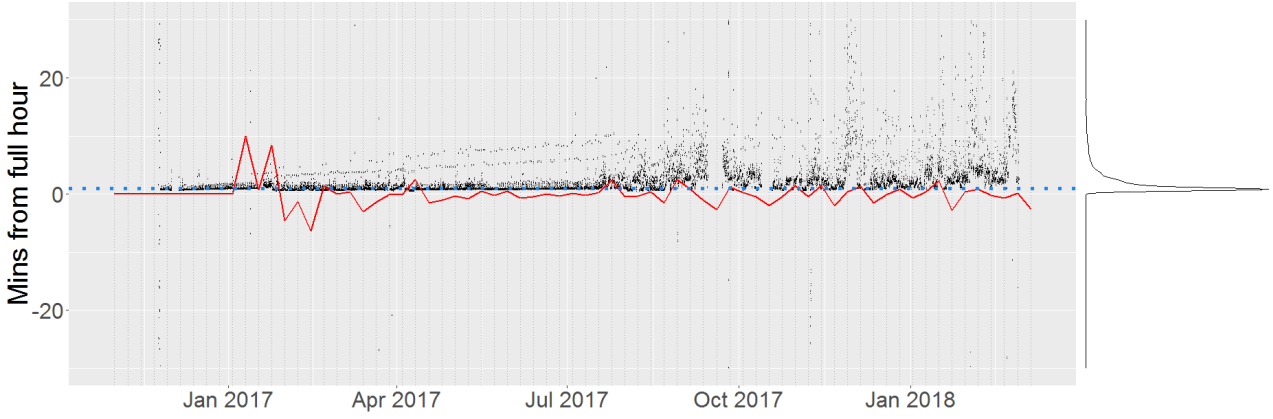


Figure 23: Alerts raised by the changes in periodicity for exemplary methods

5 Implementation of the Rules in SUPERSEDE

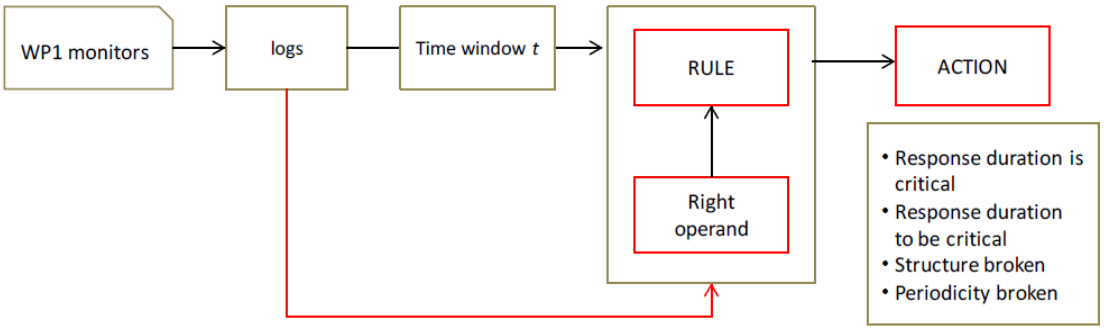


Figure 24: Rule adaptation and evaluation pipeline

Figure 24 shows the steps of the rule adaptation and evaluation work-flow. A monitor collects data from a system and creates event-based log files (see Section 1.2.1, and Figure 2). These files are processed and split into historical and current data. The time window t can be chosen depending on the use case. Generally speaking, right operands (*thresholds*) are extracted from historical data and capture the overall state of the system, while left operands (*evaluating metrics*) are extracted from the most recent t time units. This describes the dynamic rule adaptation concept.

More concretely, the threshold for response time is computed every 24 hrs. Two ECA rules are evaluated every hour based on this computed threshold, i.e., one for the detection and one for the prediction of the exceeding of the threshold. The other two rules (structural break alarm and periodicity violation) are inherently dynamic, as they detect changes in the statistical properties of the signal by measuring its deviations from history. Structural break test continuously tests if there has been a breach in the linear trend of the new data compared to the old data, i.e., testing if the new data belongs to the same (linear) trend of the old data. Proposed periodicity validation checks are a probability-based method, calculating the distributions of the hourly requests, thus enabling the detection of periods (times of the hour where the requests are denser) and irregularities.

5.1 R Scripts

The first argument of each script (input data path) is mandatory. The rest is optional (scripts assume the arguments to have some default paths). A summary of the scripts is described in Table 4.

Script	Inputs	Outputs	Syntax	Remark
getThresholds.r	Data_ALL; methodClustering;	thresholds	getThresholds.r <Data_ALL> <thresholds> <methodClustering>	to be run every 24 hours, with all of the available data by that time point
hourlyRespDurEvaluate.r	Data_1hr; thresholds; methodClustering;	RD_alarms	hourlyRespDurEvaluate.r <Data_1hr> <thresholds> <RD_alarms> <methodClustering>	to be run every hour, on the data of the previous hour
structuralBreakTests.r	Data_ALL; lastStrucChange; methodClustering	lastStrucChange; SBT_alarms	structuralBreakTests.r <Data_ALL> <lastStrucChange.csv> <SBT_alarms.csv> <methodClustering>	to be run every 24 hours, with all of the available data by that time point. lastStrucChange is being updated with every run
periodicityBreak.r	Data_ALL; methodClustering	Per_alarms	periodicityBreak.r <Data_ALL> <Per_alarms> <methodClustering>	to be run every 24 hours, with the data of the last 5 weeks (or all data)

Table 4: Scripts

The **getThreshold.r** script extracts thresholds (right operands) from the monitoring data for each service, every 24 hours, as defined in Section 4.1.3.2. It requires all of the logs by that point and outputs the thresholds per API/Service.

The **hourlyRespDurEvaluate.r** script extracts left operands from the most recent data (to be evaluated with the output of the previous script). It requires the logs of the past hour, and outputs the APIs that broke their thresholds.

The **structuralBreakTests.r** script takes all of the data, and every 24h performs the structural break test described in Section 4.1.3.1. It outputs the list of the APIs that have had their structure broken and updates the joint table with all of the last structural breaks.

The **periodicityBreak.r** script takes all of the data and tests every 24h if there has been a change in the previously detected periodicity, as described in Section 4.3.2.

The code is listed in the appendix.

6 Conclusions

We propose a statistical approach of automating the rule extraction for operational log data. It is possible that the same approach can be applied on the data with similar properties.

By using the kernel density estimation, we detect if a certain method (or more generally, class in a nominal variable) is periodic. The periodicity in live and busy systems may be violated depending on the overall system load, and this property can be measured to estimate the *health* state of the system.

Furthermore, each *query log* that tracks user requests and system responses, automatically tracks response duration. Response duration shows trends as well, which may break (due to a system upgrade or migration). Statistical break tests have shown to be a reliable statistical tool to detect breaks in trends, thus enabling more reliable forecasting and raising alerts in the future.

They are developed for breaks in linear trends, and under some non-trivial constraints. There is a lot of space for their generalization to further cases, but due to the complexity of their asymptotic properties, it has not been done yet.

Appendices

A getThresholds.r

```
args = commandArgs(trailingOnly=TRUE)
t.start =Sys.time()
## test if there is at least one argument: if not, return an
    error
if (length(args)==0) {
  stop("At least one argument must be supplied (input file)",
      call.=FALSE)
}
if (length(args)==1) {
  # default output file
  args[2] = "thrsh_out.csv"
}
if (length(args)==2) {
  # default input file
  args[3] = "MethodClustering.csv"}

########################################
##initial setup

# Function to check whether package is installed
install.req.if.not <- function(mypkg){
  if (!is.element(mypkg, installed.packages()[,1])){
    install.packages(mypkg)}else{library(mypkg,character.only =
        TRUE)}}

#misc
options(repos = "http://cran.rediris.es/")
options(digits.secs = 3)
Sys.setenv(LANGUAGE="en")
Sys.setlocale("LC_ALL", "English")
```

```
lower_bound=7
upper_bound=60

#required packages
packages=c(
  "data.table",
  "R.filesets",
  "RecordLinkage")

#check if required packages are installed, install if not, else-
    require
sapply(packages,FUN=install.req.if.not)

########################################
## program...

#args=("C:\\Users\\Z003TF1W\\Google Drive\\Siemens\\Code\\SVN\\
    part-00000")

coln=c("LogType","Timestamp.old","Class","User","Role","SessionID
    ",
        "EventType","Direction","MethodNameOrig","ContentType","
            Content","TransID")

keep=c(1,2,6,8,9)
my.data <- fread(args[1], col.names = coln[keep],
                 select=keep,
                 header=FALSE, fill=TRUE, sep="|",blank.lines.skip
                     =T, dec=",",quote='')

my.data=my.data[LogType=="TRACE",.(Timestamp.old,MethodNameOrig,
    Direction,SessionID)]
my.data=my.data[Direction!=""]

my.data[,c("MethodNameOnly","MethodParams"):=data.table(t(sapply(
    lapply(
```

```
  as.character(my.data$MethodNameOrig),function(x){
    y=strsplit(x,"?",fixed=T)[[1]]
    if(length(y)==0){return(c("",""))}
    if(length(y)==1){return(c(y[1],""))}
    else{return(y[1:2])}}), "["))))]

my.data[,MethodParams:=NULL]

#delete the prefixes
to.del=c("http://localhost:6080","http://localhost","http://demo-
    ecosys:6080/ecosyscore/apiconsume/call",
        "http://10.50.1.100:6080","http://10.50.1.100","call","
            http://127.0.0.1:6080")

for (i in 1:length(to.del)){
  ind.here=grep(to.del[i],my.data$MethodNameOnly,value=F,fixed=T)
  if (length(ind.here)==0) next
  my.data[ind.here,MethodNameOnly:=gsub(to.del[i],"",
      MethodNameOnly,fixed=T)]}

my.data[,Timestamp:=as.POSIXct(strptime(Timestamp.old, "%Y-%m-%d
    %H:%M:%OS"))]
#my.data[is.na(Timestamp),Timestamp.old]

meth.n.cl=fread(args[3],sep=";")
#meth.n.cl=fread("C:\\Users\\Z003TF1W\\Google Drive\\Siemens\\
    Code\\SVN\\MethodClustering.csv",sep=";")

indic=match(my.data$MethodNameOnly,meth.n.cl$MethodNameOnly)

my.data[,MethodNameOnlyGrouped := meth.n.cl[indic,"ClusterName"]]
my.data[,API := meth.n.cl[indic,"API"]]

#in case of new unseen method names
########################################
if(nrow(my.data[is.na(MethodNameOnlyGrouped)])>0){
```

```
Newclust=apply(my.data[is.na(MethodNameOnlyGrouped),.(unique(
    MethodNameOnly))],1,function(x){levenshteinSim(x,as.character(
    meth.n.cl$MethodNameOnly))})
ncl.maxind=apply(Newclust,2,which.max)
ncl.max=numeric(length(ncl.maxind))
for(i in 1:length(ncl.maxind)){
  ncl.max[i]=Newclust[ncl.maxind[i],i]}

toch=is.na(my.data[,MethodNameOnlyGrouped])
indic=match(my.data[is.na(MethodNameOnlyGrouped),MethodNameOnly],
    my.data[is.na(MethodNameOnlyGrouped),unique(MethodNameOnly)])

my.data[toch,MethodNameOnlyGrouped := meth.n.cl[ncl.maxind,"
    ClusterName"][indic]]
my.data[toch, API:= meth.n.cl[ncl.maxind,"API"][indic]]}
dur=my.data[,.(Time=min(Timestamp),RESPONSE=max(Timestamp),
    REQUEST=min(Timestamp),Method=head(MethodNameOnlyGrouped,1),
    API=head(API,1),.N),by="SessionID"]

dur[,Duration:=RESPONSE-REQUEST]
dur=dur[Duration>0
        & Duration<5000
        & Duration!=Inf,]
df_out=dur[,quantile(Duration,0.75,na.rm=T),by="Method"]
colnames(df_out)=c("GroupedMethodName","DurationThresh")
df_out[DurationThresh==Inf,DurationThresh:=NA]
df_out[DurationThresh<lower_bound,DurationThresh:=lower_bound]
df_out[DurationThresh>upper_bound,DurationThresh:=upper_bound]

write.csv(df_out, file=args[2], row.names=FALSE)
paste("Execution duration:",round(Sys.time()-t.start,2),'seconds.
    ')
```

B hourlyRespDurEvaluate.r

```
args = commandArgs(trailingOnly=TRUE)
t.start =Sys.time()
## test if there is at least one argument: if not, return an
    error

#args=("C:\\Users\\Z003TF1W\\Google Drive\\Siemens\\Code\\SVN\\
    part-eval")

if (length(args)==0) {
  stop("At least one argument must be supplied (input file)",
      call.=FALSE)
}
if (length(args)==1) {
  # default input file
  args[2] = "thrsh_out.csv"
}
if (length(args)==2) {
    # default output file
    args[3] = "alarm.csv"
}
if (length(args)==3) {
      # default input file
      args[4] = "MethodClustering.csv"}

#########################################
##initial setup

# Function to check whether package is installed
install.req.if.not <- function(mypkg){
  if (!is.element(mypkg, installed.packages()[,1])){
    install.packages(mypkg)}else{library(mypkg,character.only =
        TRUE)}}

#misc
```

```
options(repos = "http://cran.rediris.es/")
options(digits.secs = 3)
Sys.setenv(LANGUAGE="en")
Sys.setlocale("LC_ALL", "English")

#required packages
packages=c(
  "data.table",
  "R.filesets",
  "RecordLinkage")

#check if required packages are installed, install if not, else-
    require
sapply(packages,FUN=install.req.if.not)

########################################
## program...
#dat = read.table(args[1], header=TRUE)

#fread instead
coln=c("LogType","Timestamp.old","Class","User","Role","SessionID
    ",
        "EventType","Direction","MethodNameOrig","ContentType","
            Content","TransID")

keep=c(1,2,6,8,9)
my.data <- fread(args[1], col.names = coln[keep],
                 select=keep,
                 header=FALSE, fill=TRUE, sep="|",blank.lines.skip
                     =T, dec=",",quote='')

my.data=my.data[LogType=="TRACE",.(Timestamp.old,MethodNameOrig,
    Direction,SessionID)]
my.data=my.data[Direction!=""]
my.data[,Timestamp:=as.POSIXct(strptime(Timestamp.old,   "%Y-%m-%d
    %H:%M:%OS"))]
```

```
my.data[is.na(Timestamp),Timestamp:=as.POSIXct(strptime(Timestamp
    .old,    "%Y-%m-%dT%H:%M:%OS%z"))]
my.data[,c("MethodNameOnly","MethodParams"):=data.table(t(sapply(
    lapply(
  as.character(my.data$MethodNameOrig),function(x){
    y=strsplit(x,"?",fixed=T)[[1]]
    if(length(y)==0){return(c("",""))}
    if(length(y)==1){return(c(y[1],""))}
    else{return(y[1:2])}}),  "[")))]

my.data[,MethodParams:=NULL]
#delete the prefixes
to.del=c("http://localhost:6080","http://localhost","http://demo_
    ecosys:6080/ecosyscore/apiconsume/call",
          "http://10.50.1.100:6080","http://10.50.1.100","call","
              http://127.0.0.1:6080")

for (i in 1:length(to.del)){
  ind.here=grep(to.del[i],my.data$MethodNameOnly,value=F,fixed=T)
  if (length(ind.here)==0) next
  my.data[ind.here,MethodNameOnly:=gsub(to.del[i],"",
      MethodNameOnly,fixed=T)]}

my.data[,Timestamp:=as.POSIXct(strptime(Timestamp.old,    "%Y-%m-%d
    %H:%M:%OS"))]
meth.n.cl=fread(args[4],sep=";")

indic=match(my.data$MethodNameOnly,meth.n.cl$MethodNameOnly)

my.data[,MethodNameOnlyGrouped := meth.n.cl[indic,"ClusterName"]]
my.data[,API := meth.n.cl[indic,"API"]]

#in case of new unseen method names
if(nrow(my.data[is.na(MethodNameOnlyGrouped)])>0){
  Newclust=apply(my.data[is.na(MethodNameOnlyGrouped),.(unique(
      MethodNameOnly))],1,function(x){levenshteinSim(x,as.
```

```
        character(meth.n.cl$MethodNameOnly))})
    ncl.maxind=apply(Newclust,2,which.max)
    ncl.max=numeric(length(ncl.maxind))
    for(i in 1:length(ncl.maxind)){
      ncl.max[i]=Newclust[ncl.maxind[i],i]}
    toch=is.na(my.data[,MethodNameOnlyGrouped])
    indic=match(my.data[is.na(MethodNameOnlyGrouped),MethodNameOnly
        ],my.data[is.na(MethodNameOnlyGrouped),unique(MethodNameOnly
        )])
    my.data[toch,MethodNameOnlyGrouped := meth.n.cl[ncl.maxind,"
        ClusterName"][indic]]
    my.data[toch, API:= meth.n.cl[ncl.maxind,"API"][indic]]}
dur=my.data[,.(Time=min(Timestamp),RESPONSE=max(Timestamp),
    REQUEST=min(Timestamp),Method=head(MethodNameOnlyGrouped,1),
    API=head(API,1),.N),by="SessionID"]

dur[,Duration:=RESPONSE-REQUEST]
dur=dur[Duration>0
          & Duration<5000
          & Duration!=Inf,]

df_out=dur[,mean(Duration),by="Method"]
colnames(df_out)=c("GroupedMethodName","ResponseDuration")

thrs=fread(args[2],sep=",")
setkey(thrs,GroupedMethodName)
setkey(df_out,GroupedMethodName)

write.csv(df_out[thrs][ResponseDuration>DurationThresh], file=
    args[3], row.names=FALSE)
paste("Execution duration:",round(Sys.time()-t.start,2),'seconds.
    ')
```

C structuralBreakTests.r

```
args = commandArgs(trailingOnly=TRUE)
t.start =Sys.time()
## test if there is at least one argument: if not, return an
    error
if (length(args)==0) {
  stop("At least one argument must be supplied (input file)",
      call.=FALSE)
}
if (length(args)==1) {
  # default output file
  args[2] = "SBT_alarms.csv"
}

getOption(datatable.fread.dec.locale)

########################################
##initial setup

# Function to check whether package is installed
install.req.if.not <- function(mypkg){
  if (!is.element(mypkg, installed.packages()[,1])){
    install.packages(mypkg)}else{library(mypkg,character.only =
        TRUE)}}

#misc
options(repos = "http://cran.rediris.es/")
options(digits.secs = 3)
Sys.setenv(LANGUAGE="en")
Sys.setlocale("LC_ALL", "English")

#required packages
packages=c(
  "data.table",
  "R.filesets",
```

```
  "RecordLinkage",
  "strucchange"
  )

#check if required packages are installed, install if not, else-
    require
sapply(packages,FUN=install.req.if.not)

########################################
## program...
#dat = read.table(args[1], header=TRUE)

#fread instead

#args=("ecosys_consumption.log.2018-01-10")

my.data <- fread(args[1], header=FALSE,sep="|",blank.lines.skip
    =T)
colnames(my.data)=c("LogType","Timestamp.old","Class","User","
    Role","SessionID","EventType","Direction","MethodNameOrig","
    ContentType","Content","TransID")

my.data=my.data[LogType=="TRACE",.(Timestamp.old,MethodNameOrig,
    Direction,SessionID)]
my.data=my.data[Direction!=""]

my.data[,c("MethodNameOnly","MethodParams"):=data.table(t(sapply(
    lapply(
  as.character(my.data$MethodNameOrig),function(x){
    y=strsplit(x,"?",fixed=T)[[1]]
    if(length(y)==0){return(c("",""))}
    if(length(y)==1){return(c(y[1],""))}
    else{return(y[1:2])}}), "["))))]

my.data[,MethodParams:=NULL]
```

```
#delete the prefixes
to.del=c("http://localhost:6080","http://localhost","http://demo_
    ecosys:6080/ecosyscore/apiconsume/call",
          "http://10.50.1.100:6080","http://10.50.1.100","call","
              http://127.0.0.1:6080")

for (i in 1:length(to.del)){
  ind.here=grep(to.del[i],my.data$MethodNameOnly,value=F,fixed=T)
  if (length(ind.here)==0) next
  my.data[ind.here,MethodNameOnly:=gsub(to.del[i],"",
      MethodNameOnly,fixed=T)]}

my.data[,Timestamp:=as.POSIXct(strptime(Timestamp.old,   "%Y-%m-%
    dT%H:%M:%OS%z"))]
meth.n.cl=fread("MethodClustering.csv",sep=";")

indic=match(my.data$MethodNameOnly,meth.n.cl$MethodNameOnly)

my.data[,MethodNameOnlyGrouped := meth.n.cl[indic,"ClusterName"]]
my.data[,API := meth.n.cl[indic,"API"]]

#in case of new unseen method names
#######################################
if(nrow(my.data[is.na(MethodNameOnlyGrouped)])>0){
  Newclust=apply(my.data[is.na(MethodNameOnlyGrouped),.(unique(
      MethodNameOnly))],1,function(x){levenshteinSim(x,as.
      character(meth.n.cl$MethodNameOnly))})
  ncl.maxind=apply(Newclust,2,which.max)

  ncl.max=numeric(length(ncl.maxind))
  for(i in 1:length(ncl.maxind)){
    ncl.max[i]=Newclust[ncl.maxind[i],i]}

  toch=is.na(my.data[,MethodNameOnlyGrouped])
  indic=match(my.data[is.na(MethodNameOnlyGrouped),MethodNameOnly
      ],my.data[is.na(MethodNameOnlyGrouped),unique(MethodNameOnly
```

```
    ) ] )

  my.data[toch,MethodNameOnlyGrouped := meth.n.cl[ncl.maxind,"
      ClusterName"][indic]]
  my.data[toch, API:= meth.n.cl[ncl.maxind,"API"][indic]]}

dur=my.data[,.(Time=min(Timestamp),RESPONSE=max(Timestamp),
    REQUEST=min(Timestamp),Method=head(MethodNameOnlyGrouped,1),
    API=head(API,1),.N),by="SessionID"]

dur[,Duration:=RESPONSE-REQUEST]
dur=dur[Duration>0
        & Duration<5000
        & Duration!=Inf,]

uniqm=dur[,.N,by=Method][,Method] ############################
lastchange=fread("lastStrucChg.csv")
lastchange[,V1:=as.POSIXct(V1)]
lastchcomp=lastchange
test.type=c("Rec-CUSUM","OLS-CUSUM","Rec-MOSUM","OLS-MOSUM")

for (l in 1:length(uniqm)){
  for (j in 1:length(test.type)){
    dd=dur[Method==uniqm[l],.(as.numeric(Duration),Time)]

    sbt=dd
    colnames(sbt)=c("Session_Duration","Timestamp")

    lastchg1=lastchange[Method==uniqm[l],V1]

    tmpdat=sbt[Timestamp > lastchg1]
    if(nrow(tmpdat)<20){break}

    p=sctest(with(tmpdat,efp(Session_Duration~as.numeric(
        Timestamp),dynamic=T,type=test.type[j],h=0.15)))$p.value
    if(is.na(p)){p=1}
```

```
    if(p<0.05){
      lastchange[Method==uniqm[l],V1:=tmpdat[,max(Timestamp)]]
      break}
    }}

setkey(lastchange,Method)
setkey(lastchcomp,Method)

paste(lastchcomp[lastchange][V1!=i.V1,.(Method,i.V1)])

#lastchange=dur[,min(Time),by=Method]
write.csv(lastchange, file="lastStrucChg.csv", row.names=FALSE)
write.csv(lastchcomp[lastchange][V1!=i.V1,.(Method,i.V1)], file=
    args[2], row.names=FALSE)
paste("Execution duration:",round(Sys.time()-t.start,2),'seconds.
    ')
```

D periodicityBreak.r

```
args = commandArgs(trailingOnly=TRUE)
t.start =Sys.time()
## test if there is at least one argument: if not, return an
    error
if (length(args)==0) {
  stop("At least one argument must be supplied (input file)",
      call.=FALSE)
}
if (length(args)==1) {
  # default output file
  args[2] = "SBT_alarms.csv"
}

getOption(datatable.fread.dec.locale)

########################################
##initial setup

# Function to check whether package is installed
install.req.if.not <- function(mypkg){
  if (!is.element(mypkg, installed.packages()[,1])){
    install.packages(mypkg)}else{library(mypkg,character.only =
        TRUE)}}

#misc
options(repos = "http://cran.rediris.es/")
options(digits.secs = 3)
Sys.setenv(LANGUAGE="en")
Sys.setlocale("LC_ALL", "English")

#required packages
packages=c(
  "data.table",
  "R.filesets",
```

```
  "RecordLinkage",
  "strucchange"
  )

#check if required packages are installed, install if not, else-
    require
sapply(packages,FUN=install.req.if.not)

########################################
## program...
#dat = read.table(args[1], header=TRUE)

#fread instead

#args=("ecosys_consumption.log.2018-01-10")

my.data <- fread(args[1], header=FALSE,sep="|",blank.lines.skip
    =T)
colnames(my.data)=c("LogType","Timestamp.old","Class","User","
    Role","SessionID","EventType","Direction","MethodNameOrig","
    ContentType","Content","TransID")

my.data=my.data[LogType=="TRACE",.(Timestamp.old,MethodNameOrig,
    Direction,SessionID)]
my.data=my.data[Direction!=""]

my.data[,c("MethodNameOnly","MethodParams"):=data.table(t(sapply(
    lapply(
  as.character(my.data$MethodNameOrig),function(x){
    y=strsplit(x,"?",fixed=T)[[1]]
    if(length(y)==0){return(c("",""))}
    if(length(y)==1){return(c(y[1],""))}
    else{return(y[1:2])}}), "["))))]

my.data[,MethodParams:=NULL]
```

```
#delete the prefixes
to.del=c("http://localhost:6080","http://localhost","http://demo_
    ecosys:6080/ecosyscore/apiconsume/call",
          "http://10.50.1.100:6080","http://10.50.1.100","call","
              http://127.0.0.1:6080")

for (i in 1:length(to.del)){
  ind.here=grep(to.del[i],my.data$MethodNameOnly,value=F,fixed=T)
  if (length(ind.here)==0) next
  my.data[ind.here,MethodNameOnly:=gsub(to.del[i],"",
      MethodNameOnly,fixed=T)]}

my.data[,Timestamp:=as.POSIXct(strptime(Timestamp.old,   "%Y-%m-%
    dT%H:%M:%OS%z"))]
meth.n.cl=fread("MethodClustering.csv",sep=";")

indic=match(my.data$MethodNameOnly,meth.n.cl$MethodNameOnly)

my.data[,MethodNameOnlyGrouped := meth.n.cl[indic,"ClusterName"]]
my.data[,API := meth.n.cl[indic,"API"]]

#in case of new unseen method names
#######################################
if(nrow(my.data[is.na(MethodNameOnlyGrouped)])>0){
  Newclust=apply(my.data[is.na(MethodNameOnlyGrouped),.(unique(
      MethodNameOnly))],1,function(x){levenshteinSim(x,as.
      character(meth.n.cl$MethodNameOnly))})
  ncl.maxind=apply(Newclust,2,which.max)

  ncl.max=numeric(length(ncl.maxind))
  for(i in 1:length(ncl.maxind)){
    ncl.max[i]=Newclust[ncl.maxind[i],i]}

  toch=is.na(my.data[,MethodNameOnlyGrouped])
  indic=match(my.data[is.na(MethodNameOnlyGrouped),MethodNameOnly
      ],my.data[is.na(MethodNameOnlyGrouped),unique(MethodNameOnly
```

```
    )])

  my.data[toch,MethodNameOnlyGrouped := meth.n.cl[ncl.maxind,"
      ClusterName"][indic]]
  my.data[toch, API:= meth.n.cl[ncl.maxind,"API"][indic]]}

dur=my.data[,.(Time=min(Timestamp),RESPONSE=max(Timestamp),
    REQUEST=min(Timestamp),Method=head(MethodNameOnlyGrouped,1),
    API=head(API,1),.N),by="SessionID"]

dur[,Duration:=RESPONSE-REQUEST]
dur=dur[Duration>0
        & Duration<5000
        & Duration!=Inf,]

uniqm=dur[,.N,by=Method][,Method] ############################
lastchange=fread("lastStrucChg.csv")
lastchange[,V1:=as.POSIXct(V1)]
lastchcomp=lastchange
test.type=c("Rec-CUSUM","OLS-CUSUM","Rec-MOSUM","OLS-MOSUM")

for (l in 1:length(uniqm)){
  for (j in 1:length(test.type)){
    dd=dur[Method==uniqm[l],.(as.numeric(Duration),Time)]

    sbt=dd
    colnames(sbt)=c("Session_Duration","Timestamp")

    lastchg1=lastchange[Method==uniqm[l],V1]

    tmpdat=sbt[Timestamp > lastchg1]
    if(nrow(tmpdat)<20){break}

    p=sctest(with(tmpdat,efp(Session_Duration~as.numeric(
        Timestamp),dynamic=T,type=test.type[j],h=0.15)))$p.value
    if(is.na(p)){p=1}
```

```
    if(p<0.05){
      lastchange[Method==uniqm[l],V1:=tmpdat[,max(Timestamp)]]
      break}
    }}

setkey(lastchange,Method)
setkey(lastchcomp,Method)

paste(lastchcomp[lastchange][V1!=i.V1,.(Method,i.V1)])

#lastchange=dur[,min(Time),by=Method]
write.csv(lastchange, file="lastStrucChg.csv", row.names=FALSE)
write.csv(lastchcomp[lastchange][V1!=i.V1,.(Method,i.V1)], file=
    args[2], row.names=FALSE)
paste("Execution duration:",round(Sys.time()-t.start,2),'seconds.
    ')
```

References

[1] What is log (log file)? - definition. URL http://whatis.techtarget.com/definition/log-log-file.

[2] SUPERSEDE H2020 Project: "SUpporting evolution and adaptation of PERsonalized Software by Exploiting contextual Data and End-user feedback", grant agreement no: 644018, 2015. URL www.supersede.eu.

[3] Darboux integral, Feb 2018. URL https://en.wikipedia.org/wiki/Darboux_integral.

[4] Mumtaz Ahmed, Gulfam Haider, and Asad Zaman. Detecting structural change with heteroskedasticity. *Communications in Statistics - Theory and Methods*, 46(21):1044610455, Oct 2016. doi: 10.1080/03610926.2016.1235200.

[5] Donald W. K. Andrews. Tests for parameter instability and structural change with unknown change point. *Econometrica*, 61(4):821, 1993. doi: 10.2307/2951764.

[6] ASCR. Web, Sep 2017. URL http://www.ascr.at/en/.

[7] Joop Aué. Log analysis from A to Z: A literature survey. Master's thesis, 2016.

[8] R. L. Brown, J. Durbin, and J. M. Evans. Techniques for testing the constancy of regression relationships over time. *Journal of the Royal Statistical Society. Series B (Methodological)*, 37(2):149–192, 1975. ISSN 00359246. URL http://www.jstor.org/stable/2984889.

[9] Varun Chandola, Deepthi Cheboli, and Vipin Kumar. Detecting anomalies in a time series database. 2009.

[10] Deepthi Cheboli. Anomaly detection of time series. Master's thesis, 2010.

[11] Gregory C. Chow. Tests of equality between sets of coefficients in two linear regressions. *Econometrica*, 28(3):591, 1960. doi: 10.2307/1910133.

[12] Allan Cortzen. Darboux integral. URL http://mathworld.wolfram.com/DarbouxIntegral.html.

[13] D.J. Foulis and M.A. Munem. *After Calculus–analysis*. Dellen Publishing Company, 1989. ISBN 9780023391309. URL https://books.google.de/books?id=kSMnAQAAIAAJ.

[14] Xiaoyu Fu, Rui Ren, Jianfeng Zhan, Wei Zhou, Zhen Jia, and Gang Lu. Logmaster: Mining event correlations in logs of large-scale cluster systems. *2012 IEEE 31st Symposium on Reliable Distributed Systems*, 2012. doi: 10.1109/srds.2012.40.

[15] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R. Lyu. An evaluation study on log parsing and its use in log mining. *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2016. doi: 10.1109/dsn.2016.66.

[16] J. Del Hoyo, G. Llorente, and C. Rivero. Testing for constant parameters in nonlinear models: A quick procedure with an empirical illustration. *Computational Economics*, 2017. doi: 10.1007/s10614-017-9693-5.

[17] George Kapetanios. Testing for structural breaks in nonlinear dynamic models using artificial neural network approximations. *SSRN Electronic Journal*, Jan 2003. doi: 10.2139/ssrn.358340.

[18] Friedrich Leisch, Kurt Hornik, and Chung-Ming Kuan. Monitoring structural changes with the generalized fluctuation test. *Econometric Theory*, 16(6): 835854, 2000. doi: 10.1017/s0266466600166022.

[19] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, February 1966.

[20] Esfandiar Maasoumi, Asad Zaman, and Mumtaz Ahmed. Tests for structural change, aggregation, and homogeneity. *Economic Modelling*, 27(6):13821391, 2010. doi: 10.1016/j.econmod.2010.07.009.

[21] Adam Oliner, Archana Ganapathi, and Wei Xu. Advances and challenges in log analysis. *Communications of the ACM*, 55(2):55, Jan 2012. doi: 10.1145/2076450.2076466.

[22] E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100, 1954. doi: 10.2307/2333009.

[23] Richard E. Quandt. Tests of the hypothesis that a linear regression system obeys two separate regimes. *Journal of the American Statistical Association*, 55(290):324, 1960. doi: 10.2307/2281745.

[24] Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *Ann. Math. Statist.*, 27(3):832–837, 09 1956. doi: 10.1214/aoms/1177728190. URL `https://doi.org/10.1214/aoms/1177728190`.

[25] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 03*, 2003. doi: 10.1145/956750.956799.

[26] Achim Zeileis, Friedrich Leisch, Kurt Hornik, and Christian Kleiber. strucchange: An R-package for testing for structural change in linear regression models. *Journal of Statistical Software*, 7(2), 2002. doi: 10.18637/jss.v007.i02.

[27] Achim Zeileis, Friedrich Leisch, Kurt Hornik, and Christian Kleiber. Testing, monitoring, and dating structural changes [R package strucchange version 1.5-1], Jun 2015. URL `https://cran.r-project.org/package=strucchange`.

[28] Ziming Zheng, Zhiling Lan, Byung H. Park, and Al Geist. System log preprocessing to improve failure prediction. *2009 IEEE/IFIP International Conference on Dependable Systems and Networks*, 2009. doi: 10.1109/dsn.2009.5270289.