Institute of Software Technology
Reliable Software Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Context-Aware Load Testing in Continuous Software Engineering

Alper Hidiroglu

Course of Study:	Softwaretechnik
Examiner:	Dr.-Ing. André van Hoorn
Supervisor:	Henning Schulz, M.Sc. Prof.-Ing. Petr Tůma, Dr.
Commenced:	July 16, 2018
Completed:	January 16, 2019
CR-Classification:	D.4.8

Abstract

To ensure adequate performance of a system, performance regressions have to be detected early in the software development process. Before a new software version is released, load tests should be applied on the system. In the context of continuous software engineering, it is crucial to keep delivery pipelines as short-running as possible in order to release software changes frequently. Since load tests typically take longer than functional tests, it is not possible to test for every possible workload scenario every time before a change is committed. It would be better to focus only on those load scenarios that are relevant for a given context, that consists, for example, of marketing campaigns, sports events, weather, etc.

The goal of this work is to automatically generate load tests that test for the relevant load scenarios in the future. Thereby, we aim at reducing the resource usage and the test execution time. We develop a context description language to express contexts that can occur in the future. Our approach takes as input a context description and recorded request logs from the production system. It then uses the WESSBAS approach to calculate historical workload data from the recorded request logs. Based on the historical workload data and the passed context description, our approach then forecasts the future workload. The forecasted workload is processed and relevant load scenarios are identified that will occur in the future. Our approach then uses the WESSBAS approach again to automatically generate load tests that test for the identified load scenarios.

We evaluated our approach with a real-world data set, that contains recorded requests from the Student Information System (SIS) of the Charles University in Prague. The evaluation shows that contexts help to reduce the testing effort and to focus only on the relevant workload scenarios. However, the evaluation also shows that our approach has limitations regarding the accuracy of the forecasted workload. Load tests, that are generated from inaccurately forecasted workload, do not test for the relevant load scenarios.

Kurzfassung

Zur Sicherstellung geforderter Performance eines session-basierten Systems müssen Performanz-Regressionen im Softwareentwicklungsprozess frühzeitig erkannt werden. Vor der Veröffentlichung einer Softwareversion sollten Lasttests auf das System angewendet werden. Im Rahmen von Continuous Software Engineering ist es wichtig, sicherzustellen, dass die Delivery Pipelines möglichst kurzlaufend gehalten werden, um Softwareänderungen häufig veröffentlichen zu können. Da Lasttests typischerweise mehr Zeit in Anspruch nehmen als funktionale Tests, ist es nicht möglich vor jedem bevorstehenden Commit einer Änderung jedes mögliche Lastszenario zu testen. Es wäre besser sich nur auf die Lastszenarien zu fokussieren, die relevant für einen gegebenen Kontext sind. Kontexte sind beispielsweise Marketing-Kampagnen, Sportereignisse, Wetter etc.

Das Ziel dieser Arbeit ist das automatische Generieren von Lasttests, die für relevante Lastszenarien in der Zukunft testen. Dabei ist es unser Ziel, die Testausführungszeit und die Ressourcennutzung zu reduzieren. Wir entwickeln eine Kontextbeschreibungssprache, um Kontexte, die in der Zukunft auftreten könnten, zu beschreiben. Unser Ansatz nimmt als Eingaben eine Kontextbeschreibung und Request Logs entgegen. Request Logs enthalten aufgezeichnete Anfragen an das Produktionssystem. Unser Ansatz benutzt daraufhin den WESSBAS-Ansatz, um aus den Request Logs die historische Last zu berechnen. Basierend auf der historischen Last und einer Kontextbeschreibung wird dann die zukünftliche Last vorhergesagt. Die vorhergesagte Last wird verarbeitet und es werden relevante Lastszenarien erkannt, die in der Zukunft auftreten werden. Unser Ansatz benutzt daraufhin den WESSBAS-Ansatz erneut, um automatisch Lasttests zu generien, welche für die erkannten Lastszenarien testen.

Wir haben unseren Ansatz mit einem Datensatz aus der echten Welt evaluiert. Dieser enthält aufgezeichnete Anfragen an das Student Information System (SIS) der Karls-Universität in Prag. Die Evaluation zeigt, dass Kontexte dabei helfen den Testaufwand zu reduzieren. Weiterhin helfen Kontexte dabei, sich nur auf die relevanten Lastszenarien zu fokussieren. Die Evaluation zeigt aber auch, dass unser Ansatz in seinem jetzigen Zustand keine genauen Lastvorhersagen tätigen kann. Lasttests, die aus ungenau vorhergesagter Last generiert werden, testen nicht für die relevanten Lastszenarien.

Contents

List of Figures

List of Tables

List of Acronyms

CD continuous delivery

CDL context description language

DB database

EBNF Extended Backus-Naur Form

InfluxQL Influx Query Language

JRI Java/R Interface

RQ research question

SIS Student Information System

SUT system under test

TSDB time series database

List of Listings

List of Algorithms

Chapter 1

Introduction

This thesis presents an approach for the generation of context-aware load tests, that only test for the relevant workload. In the scope of this thesis, a context is an environmental property with an influence on the workload in a session-based system. Such contexts are, e.g., events like marketing campaigns, but also time series like a temperature curve. Our approach forecasts the future workload based on a context. From the forecasted workload, we then extract the load tests.

In this chapter, we want to clarify, why such an approach is meaningful. We start with a short motivation in Section 1.1. In Section 1.2, we list the goals of this thesis. A short overview of the approach is presented in Section 1.3. The last part of this chapter is the document structure in Section 1.4.

1.1. Motivation

In continuous software engineering [Bos16], it is crucial to keep the delivery pipeline [HF10] as short-running as possible to enable high delivery frequencies. For this reason, the classic long-running load testing approach [JH15] is hard to be executed. Load tests take more time than functional tests and therefore, it is not possible to execute one or more load tests that test for every possible load scenario whenever a new version of the system or some of its components are released. Big companies like Amazon are releasing hundreds of new versions of their systems every day [Joh13]. It would be better to focus only on the load scenarios relevant for a given context, for example, marketing campaigns, sports events, weather, etc. This can result in reduced test execution time and resource usage.

The NovaTec Consulting GmbH and the University of Stuttgart (Reliable Software Systems Group) launched the collaborative research project ContinuITy [SAH18], that is an

approach for automated load testing and which will be embedded into the continuous delivery process. ContinuITy aims to automatically generate load tests, that are representative for the usage profiles, with the help of continuously collected measurements [CGD18] or recorded request logs from the production system. One important goal of the project is to reduce the test execution time and resource usage by generating only load tests that are relevant for a given context. Until now, ContinuITy does not consider contexts when generating load tests.

This thesis is part of the ContinuITy approach and targets the load test generation process. The main goal is to output load tests that only test for the relevant workload. For this purpose, we will consider contexts. ContinuITy is able to output load tests with the help of recorded request logs from the production system. The outcome of this thesis will be an approach that works as follows. We process request logs and calculate the past workload the system experienced out of these logs. Afterwards, we forecast the future workload based on the past workload and a given context. The forecasted workload is representative for the real workload the system will experience in the future. From the forecasted workload, we then can generate context-aware load tests, that only test for the workload that will occur in the future.

As described above, our approach aims to forecast future workload based on a context, and then to generate load tests from the forecasted workload. Two main characteristics of workload are the workload intensity and user behavior. The intensity describes how many concurrent users access the system, and the user behavior is how they navigate through the application. To the best of our knowledge, no approach exists that forecasts both of these characteristics, and especially not based on a context. Existing works forecast only the workload intensity [HHKA14; RDG11]. In this thesis, we build an approach that forecasts both of these characteristics based on contexts.

1.2. Goals

In this section, we list the goals of this thesis.

Development of a description language for contexts: The first goal is to develop a description language to express contexts. The language can be used to pass a description of future contexts, that will occur in the time range a user wants to load test for, to our approach, which then will be considered for workload forecasts. A user should be able to express different kinds of contexts with the language. For this purpose, we research what kind of contexts exist, and consider them in the design of the language.

Find a suitable workload representation: The second goal is to find a workload representation, which considers the two main characteristics of workload, that are the workload intensity and the user behavior, and which can be used to forecast the future workload. Furthermore, it should be able to extract load tests from the forecasted workload.

Forecast the future workload accurately: One further goal of this thesis is to forecast the future workload a system will experience with high accuracy. Otherwise, when we extract load tests from the forecasted workload, they will not test for the real workload. To increase the forecast accuracy, we also consider expected occurring contexts in the future.

Extract relevant load tests from the forecasted workload: We only want to extract load tests from the forecasted workload that simulate load scenarios that really occur in the future. Also, a user of our approach might want to test only for particular load scenarios. Our approach should return load tests, that simulate the requested load scenarios. The testing effort (resource usage and test execution time) of the execution of such a load test should not exceed the effort that is really required to simulate the load scenario in the future.

Implementation of the approach: This thesis is part of the ContinuITy approach. Hence, another goal is to extend the ContinuITy approach and to embed our approach into it.

Evaluation of the approach: We want to investigate, whether our approach is able to forecast the future workload with high accuracy, and whether the extracted load tests test only for the relevant workload. The goal is to design experiments whose results can precisely answer the research questions (RQs) of the evaluation.

1.3. Overview of the Approach

The workload a system experiences consists of the amount of concurrent users over time (workload intensity) and the behavior of the users when navigating through the application (user behavior). WESSBAS is already capable of extracting Markov chains from a session log, which is a special representation of request logs containing session identifiers. A session log consists of session entries, where one session entry

holds information about one session. A session entry starts with a session identifier followed by the sequence of requests made in that session. Each extracted Markov chain represents a user group with particular user behavior. The calculation of the Markov chains is based on clustering algorithms, where each session entry of a session log is assigned to a cluster. We extend WESSBAS and the used clustering algorithms in order to retain the information on which session entry was considered for which cluster. I.e., after the chains are calculated, we have user groups with different user behavior and the session entries assigned to the user groups. From the session entries, we can calculate the workload intensity of each user group. The outcome of this algorithm are discrete workload intensity values over time. Additionally, we build a context description language (CDL) in order to pass context descriptions to our approach. Such a context description consists of contextual data, that are events and measurements impacting the workload an application experiences. Such events are, for example, marketing campaigns, sports events, etc. Example measurements that have an impact on the workload are, e.g., measured temperature values. From the contextual data, we calculate regressors, that are understood by time series forecasters Telescope and Prophet. They are able to include regressors in their prediction models and can consider them for the forecasting. By passing the workload intensities and regressors to the forecasters, we get as a result forecasted intensities for each user group. By processing them, we update the occurrence probabilities of the Markov chains during the workload generation. The processed intensities and updated occurrence probabilities are passed to WESSBAS, which finally generates the load tests testing for the workload that is relevant for a given context.

1.4. Document Structure

This thesis is structured as follows.

Chapter 2 – Foundations and State of the Art: In this chapter, we explain foundations for this thesis. We introduce important terminology required for this thesis, and introduce state of the art. State of the art comprises, i.a., existing tools that are important for our approach.

Chapter 3 – Related Work Here, we introduce related work to our approach. Introduced are works on test test case selection and prioritization, workload characterization, and workload forecasting.

Chapter 4 – Approach This chapter is the main part of this thesis. It describes in detail our approach. An overview of our approach was presented in Section 1.3.

Chapter 5 – Implementation In this chapter, we present implementation details of our approach. Especially, we explain how our approach is embedded into ContinuITy.

Chapter 6 – Evaluation Here, we evaluate our approach. We investigate the two main parts of our approach, that are the workload forecasting, and the processing of the forecasted workload.

Chapter 7 – Conclusion In this chapter, we summarize the outcomes of this thesis and propose future work.

Chapter A – Appendices Here, we attach further outcomes of this thesis.

Chapter 2

Foundations and State of the Art

The introduction of this thesis clarified why it is meaningful to load test based on contexts. The main goal is to focus only on the relevant workload and reduce the test execution time and resource usage.

In this chapter, we present foundations and state of the art that are important for this thesis. We first introduce important terms used throughout this document in Section 2.1. Our approach will extend the ContinuITy approach, which is described in Section 2.2. In Section 2.3, we present the WESSBAS approach that is able to automatically generate load tests from request logs. Load tests typically are executed to investigate whether a system can handle the future workload it will experience. We will forecast the future workload based on the past experienced workload and a given context with the help of time series forecasters. An introduction to time series forecasting is provided in Section 2.4. We use a time series database (TSDB) to store intermediate results of parts of our approach. An introduction to TSDBs is given in Section 2.5. To express contexts, we build a description language. The syntax of the language is based on YAML, which we shortly explain in Section 2.6.

2.1. Terminology and Definitions

In continuous delivery (CD) [HF10], the goal is to enable high delivery frequencies of software versions. CD is a continuous process during the software life cycle and concentrates on changes on the software. Changes that were successfully tested automatically during the test process go into the software release. The delivery pipeline has to be kept as short-running as possible in order to enable high delivery frequencies. The context of this work is load testing [JH15], which usually takes more time than functional tests and therefore, it is not possible to execute a load test that tests for every possible load

scenario when a new version of the system or some of its components is released. It would be better to focus only on the load that is relevant for a given context. The major goal of this work concentrates exactly on this aspect.

Load testing is an important approach to test the performance of a system. Performance is described as the runtime efficiency (including throughput, response times and availability) of an application and how fast the application operates [TTP06]. A system under test (SUT) in a testing environment is exposed to concurrent synthetic requests via defined interfaces. The properties of these requests are representative for the usage profile [Mus93] of the real operational environment. A usage profile consists for example of the load intensity, the user behavior, and input data [MA02].

In this work, we focus on session-based application systems [KRM06]. A session consists of a sequence of requests performed by a single user. Some of the requests depend on the results of earlier requests in the session. Systems of this class are session-based. With load testing, many concurrent synthetic requests are sent to these systems in order to observe their behaviors under high load.

The workload, a session-based system is experiencing, is characterized by inter-session and intra-session metrics [GSML06]. Inter-session metrics include the number of sessions per user and the number of active sessions over time (also called workload intensity). In contrast, intra-session metrics characterize single sessions. The metrics are session length, number of requests per session, and think times between requests. The sequence of requests a user performs within a session defines his user behavior [GSML06].

An appropriate approach to model the user behavior are Markov chains [LT03]. A Markov chain is a stochastic process. An example Markov chain is shown in Figure 2.1. In this example, the Markov chain transitions with a probability of 0.7 from the Start state to State A, and with a probability of 0.3 to the state B. From both State A and State B the chain then transitions with 1.0 probability to the end state. Markov chains are well suited to specify how users navigate through a web application [LT03]. Workload models exist that use Markov chains to model the user behavior [MAFM99; VHS+18].

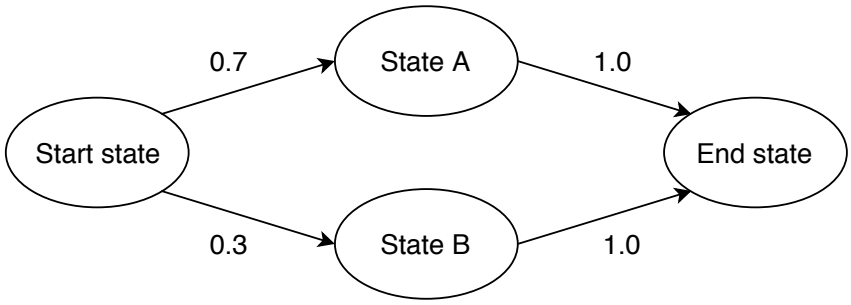

Figure 2.1.: Example Markov chain

A workload model represents the usage profile of an application system and is the result of the workload characterization (also referred to as workload specification) process [MA02]. The workload characterization process analyzes user interaction characteristics (i.e., intra-session and inter-session metrics) with an application and then outputs a workload model from these characteristics [CS93]. Systems, like WESSBAS [VHS+18], exist which automatically generate load tests from workload model instances.

2.2. ContinuITy

ContinuITy is a research project, which deals with automated performance testing in continuous software development. It uses continuously measured data from production in form of traces [JSB97] (recorded by APM tools, like Kieker [HH19] or inspectIT [NTC19]), or in form of request logs (recorded from, e.g., Web servers) to provide efficient load testing. For this purpose, load tests are extracted automatically from these data. The tests are defined through a description language. The goal of ContinuITy is to be embedded into a continuous delivery process and to reduce the usage of resources and test execution time by choosing only relevant load tests to execute [SAH18].

One major goal of the ContinuITy project is to automatically extract relevant load tests for a given context. This will be put into effect by this thesis. Our approach will be embedded into ContinuITy.

ContinuITy utilizes the WESSBAS approach [VHS+18] to automatically generate load tests. It is explained in the following.

2.3. WESSBAS

WESSBAS [VHS+18] is an approach for the automatic extraction and transformation of workload specifications for load testing of session-based systems. The WESSBAS approach comprises three main parts:

- Layered modeling of workload specifications of session-based systems with a domain-specific language (DSL)
- Extracting instances of this DSL automatically from session logs of production systems

- Transforming the DSL instances into workload specifications executable by load generation tools, like Apache JMeter [AJ19]. Load generation tools use the specifications to generate synthetic workload to the SUT by issuing a set of customer requests.

WESSBAS DSL instances are workload models. The workload model specifies the following attributes:

- An application model that specifies allowed sequences of requests (sequences of service invocations)
- A set of behavior models each representing a Markov chain. States of the Markov chain are services. Edges are transition probabilities. The behavior models also include a think time distribution. This means there is a wait time between subsequent requests chosen accordingly to the think time distribution.
- A behavior mix that assigns each behavior model a probability for its occurrence during workload generation
- A workload intensity that specifies the amount of concurrent users (sessions) during workload generation execution

A session log can be obtained from request logs that contain session identifiers and timestamps for each request. The session information can be extracted from the SUT. WESSBAS takes as an input a session log and outputs a WESSBAS DSL instance. The DSL instance can then be transformed into a JMeter load test. For this thesis, we use WESSBAS for the calculation of the past workload and for the load test generation, since it is already included in ContinuITy. For this purpose, we make use of its components, that are:

- Behavior model extractor: Calculates the behavior models and the Behavior Mix from a session log.
- Workload model generator: Builds the application model and generates a workload model instance. For this purpose, it takes as input the behavior models, the Behavior Mix, and a workload intensity, that is calculated from the session log. It is the average amount of concurrent sessions contained in the log.
- Test plan generator: Converts the workload model instance into a load test.

The behavior model extractor will be used to calculate the past workload. The other components will be used to automatically generate the load tests.

2.4. Time Series Forecasting

An important part of this thesis is to predict the future workload a system will experience. Forecasters already exist that are capable of forecasting time series data. Our approach will build upon existing solutions. The most important requirement we have on the forecaster is the option to include contextual data in forecasts. Two open-source forecasters being able to consider contextual data are Telescope and Prophet, presented in Section 2.4.1 and Section 2.4.2, respectively. Contextual data has to be passed in form of regressors to the tools in order to be understood by the tools. Later, when we build our approach, we will convert contextual data into such regressors. We compare both tools in Section 2.4.3.

2.4.1. Telescope

Telescope [ZBH+17] is a hybrid forecaster that makes use of several forecasting methods. The used methods are XGBoost [CG16], ARIMA, and artificial neural network (ANN) [HK+07]. The main goal of the Telescope approach is to achieve a more robust forecasting result, i.e., to reduce the variance in the forecasting result, by combining the benefits of the different forecasting methods. As shown in Figure 2.2, the forecasting is done in several steps. First, the frequency of the time series is estimated. Outliers and anomalies are then removed from the time series with the help of the estimated frequency. The revised time series is then decomposed into the components season, trend and remainder. The splitting is normally based on additive decomposition (additive model). Multiplicative decomposition is only applied if there is an increase of the seasonal pattern as the trend increases, and vice versa. In the next step, season and trend are separately forecasted. The forecasting of the season is simply done by continuing the seasonality, whereas the trend is forecasted by applying the ARIMA forecasting method. The revised time series is also used for learning and creating categorical information. The time series is cutted into single periods. k-means clustering algorithm [Har75] is then used for the clustering of these periods into two classes (represented by centroids). ANN is applied on both classes to forecast the clusters. The last step of the Telescope approach is the remainder forecast and composition of forecasting results. For this purpose, the machine learning algorithm XGBoost is applied [ZBH+17]. It learns the dependencies between the forecasts by passing the trend, season, and cluster forecasts as covariates to it and, finally, combines the forecasts of the components [Züf17]. Regressors can be passed to Telescope, that are then considered in the forecasting model (additive or multiplicative) as another component.

Telescope is an open-source tool [Uni18].

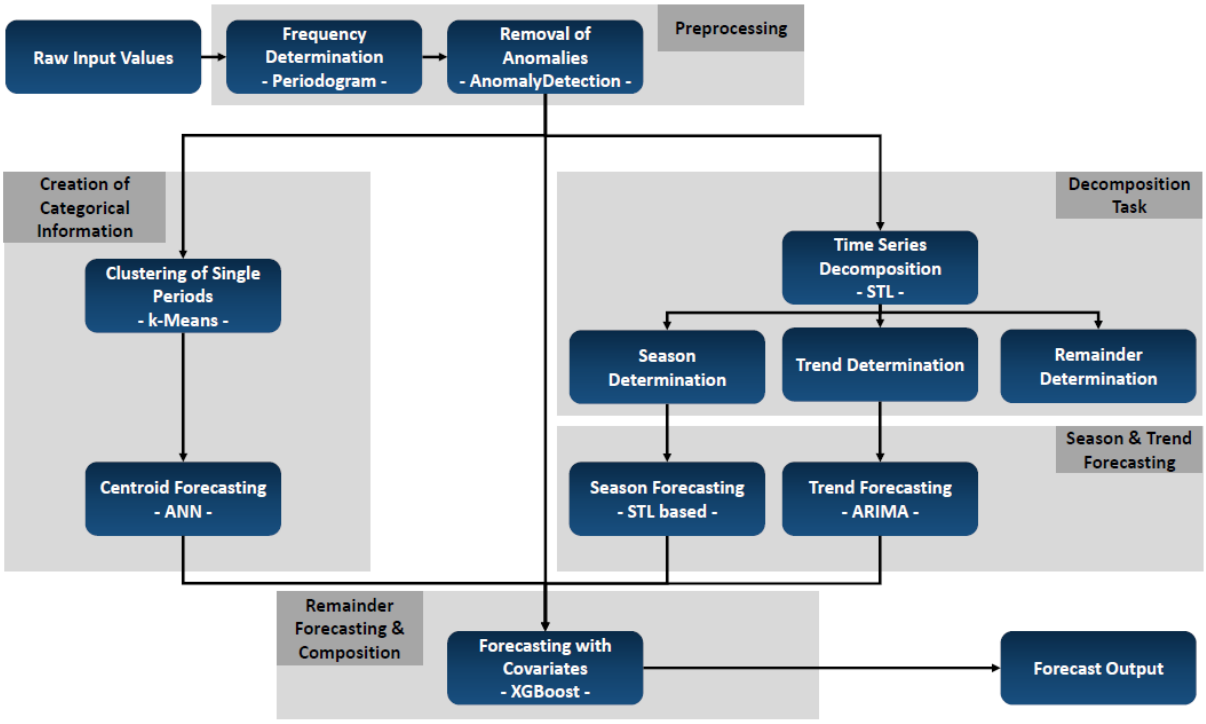

Figure 2.2.: Telescope steps [ZBH+17]

2.4.2. Prophet

Prophet [TL18] is a time series forecasting tool developed by the Facebook research group. Prophet does not aim to completely automate the forecasting, but to include configurable parameters that can be easily adjusted by non-experts in order to cover reliable forecasting results for a wide variety of business use-cases [fac17]. However, compared to other forecasting methods, it still produces accurate forecasts with default settings. Prophet's additive forecasting model handles common features of business time series and consists of three main components: trend, seasonality and holidays. Equation (2.1) shows the additive model as equation:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \tag{2.1}$$

where

- $g(t)$ is the trend function modeling non-periodic changes in the time series values (curve trend)
- $s(t)$ is the seasonality function modeling periodic changes in the time series values

- $h(t)$ is the holidays function representing effects of holidays which can occur at irregular point of times
- ϵ_t is the error term describing idiosyncratic changes which are not accommodated by the model [TL18]

For the trend component, Prophet also provides uncertainty intervals. For the seasonality function, Prophet calculates a yearly seasonal component using Fourier series, and a weekly seasonal component using dummy variables. Holidays are passed as a list to the approach by users. Additionally, users are able to add extra linear regressors to the time series. An extra regressor can also be another time series [fac19].

Prophet is available in R and Python and is open-source. Installation details are provided in the Prophet quick start documentation [fac19].

2.4.3. Comparison of Telescope and Prophet

In Table 2.1, Telescope and Prophet are compared against each other. Both tools are available in R [The19b]. Additionally, Prophet is available in Python [Pyt19]. The forecasting models are similar. The difference is in the third main component of the models. In Telescope, it is noise, whereas in Prophet it is holidays. Both tools can consider regressors as another component of the model. In contrast to Telescope, Prophet considers uncertainty intervals in its models, and forecasted values can be negative.

Telescope	**Prophet**
Available in R	Available in R and Python
Model consists of three main components: 1. trend 2. seasonality 3. noise (remainder)	Model consists of three main components: 1. trend 2. seasonality 3. holidays
Regressors can be included in forecasts	Regressors can be included in forecasts
Does not consider uncertainty intervals	Considers uncertainty intervals
Forecasted values cannot be negative	Forecasted values can be negative

Table 2.1.: Comparison between Telescope and Prophet

2.5. Time Series Database

We use a time series database (TSDB) to store forecasted workload data and contextual data. Our approach then can access the data on demand. A TSDB is a database (DB) built specifically to operate on time series data [Mit10]. Such data are time-ordered measurements or events, which are timestamped [Inf19d]. A TSDB provides typical DB operations like create, update, read and delete, and also mathematical operations to analyze and to aggregate time series data [Inf19c]. As TSDB we use InfluxDB [Inf19b].

InfluxDB is designed to store large amounts of timestamped data. Its query language is similar to SQL [DD98]. A DB table in InfluxDB is called `measurement`. A row in the `measurement` is called a data point. It consists of a timestamp and a single collection of `fields`. A `field` is a key-value pair, where the `field key` is the name of a column in the `measurement`, and the `field value` is a measured value. `field values` with the same `field key` are stored into the same column. Since a data point can consist of more than one field, it can hold more than one measured value associated with the same timestamp. Timestamps are stored in a column named `time`, which is the primary key, i.e., in one measurement two data points cannot have the same timestamp. To retrieve all data points from a `measurement`, we can run the following query:

SELECT * FROM <name of measurement>

where <name of measurement> is the name of the measurement we want to retrieve the data from. To insert a data point with one `field` into a `measurement`, we can run the following query:

INSERT <name of measurement> <field key>=<field value> <timestamp>

where <name of measurement> is the name of the measurement we want to insert the data point into, <field key> is the name of the column the actual value will be stored into, <field value> is the actual value, and <timestamp> is the associated timestamp with the data point. When no measurement named with <name of measurement> exists, this query will first create a new measurement, and then insert the data point to it. For further information on InfluxDB and its query language see the InfluxDB documentation [Inf19c].

2.6. YAML

In order to include contexts in workload forecasts, a user has to pass a context description to our approach. The syntax of such a context description is based on YAML [19b]. YAML stands for "YAML Ain't Markup Language" and is a human-readable data serialization language. It is a superset of JSON [19a]. Here, we introduce constructs of the language we will make use of for our context description language (CDL). They can be seen in the YAML document shown in Listing 2.1. `#` initiates a single-line comment. A dictionary lists `key: value` mappings and is equivalent to an object in object-oriented languages like Java [Ora19]. Dictionaries can be nested. A list is a collection of ordered values. A dictionary can be nested into a list, and vice versa. Document structure is represented through indentation [Bla19; Bre19].

Listing 2.1 YAML document

```
# root object of the document is a dictionary
# a key-value pair
key: value
# a nested dictionary
nested_dictionary:
  key: value
  key_2: value_2
# a list of elements
list:
- element1
- element2
- ...
# a dictionary as a list element
another_list:
- key: value
  key_2: value_2
```

Chapter 3

Related Work

Related work can be found in the field of test case selection and prioritization, workload characterization, and workload forecasting. Test case selection and prioritization is related to our work, since our main goal is to generate few relevant load tests by means of a context description. Related works from this area are described in Section 3.1. Based on historical workload data, we will forecast the future workload. For this purpose, historical workload data has to be represented conveniently. Hence, we deal also with workload characterization and possible representations of the workload. Related works on workload characterization are described in Section 3.2. The last related field discussed in this section is workload forecasting. In this work, we forecast the future workload based on historical workload data and contextual data, like measurements and events, and extract the load tests from the forecasted workload. Related works from this area are listed in Section 3.3.

3.1. Test Case Selection and Prioritization

The following works aim to reduce the amount of tests applied to an SUT. The first two works focus on the selection of relevant functional tests, that are used to find bugs in a new software version. The third work uses regression test selection in order to select relevant performance unit tests for the detection of performance changes between software versions.

Spieker et al. [SGMM17] present a method called RETECS for the automatic selection and prioritization of test cases in continuous integration, where test cases should be performed every time when committing new code. For this purpose, the method uses reinforcement learning in order to select the best suited test cases for effectively finding bugs in the new code. With the help of a reward function, the method learns to prioritize

error-prone test cases higher than successful test cases. In contrast to this work, we do not aim to find bugs in code with functional tests. Instead, we want to execute load tests on the SUT. One more difference is that our approach does not use reinforcement learning and information from the development (code changes) for the selection of relevant load tests, but historical workload data and contextual data, like measurements and events, i.e., we use information from the production environment for the load test selection.

Srivastava and Thiagarajan [ST02] present their test prioritization system Echelon in order to select appropriate functional tests for the detection of bugs in the SUT. It prioritizes a given set of tests by means of the changes that where made to the program. Echelon compares two versions of the code in order to find the differences between them. After that, it uses a heuristic to prioritize and select the tests that cover the changes in the new version. The similarity to our approach is the goal to apply only relevant tests to the SUT. The differences to our approach are again the focus on functional tests instead of load tests and that the authors select the tests based on code changes. We use information from the production environment (measurements/events) to select load tests.

Reichelt and Kühne [RK18] present a method called Performance Analysis of Software Systems (PeASS) for detecting performance changes in software repositories. It builds up a knowledge base of changes that have an impact on performance through the analysis of the version history of a repository. The analysis is done by using the system's unit tests. PeASS uses a method that identifies performance changes between these unit tests by measurement and statistical analysis. In order to reduce the measurement time, PeASS uses regression test selection in order to select the unit tests with potentially changed performance for every software version. The difference to our approach is the selection of performance unit tests instead of load tests. Furthermore, in contrast to this approach, we will predict the future workload a system will experience.

3.2. Workload Characterization

As mentioned in Chapter 2, the workload of session-based systems is characterized by intra-session and inter-session metrics. Beside other metrics, especially the user behavior (sequence of requests a user performs) is characterized by the intra-session metrics. On the other hand, the inter-session metrics especially comprise the workload intensity (number of active sessions over time). The outcome of the specification of these metrics is the workload model. In our approach, we forecast the future workload based on historical workload data. Somehow, we have to represent the workload from the past. For this purpose, we have to consider variations of the user behavior and

the workload intensity over time. In the following, we present works that deal with workload modeling and workload representation. We first will look into raw workload data representations, which include all required workload information. In the second part, workload models are presented, that are extracted from raw workload data and which represent a high-level view of the workload.

3.2.1. Raw Workload Data Representations

When monitoring session-based systems, monitoring facilities can produce different log formats. To be independent from specific monitoring solutions, the works from Menascé et al. [MAFM99], Vögele et al. [VHS+18], and Krishnamurthy, Rolia, and Majumdar [KRM06] first extract a preprocessed consistent log format from raw HTTP server (request) logs, that contain all required session information in order to later generate a workload model. Besides, it is easier to further process these logs than the raw request logs (i.e., for clustering algorithms used for the generation of the workload model), since they list all user sessions (one session containing all performed requests in this session) consecutively. Menascé et al. [MAFM99] and Vögele et al. [VHS+18] call the preprocessed logs session logs, whereas Krishnamurthy, Rolia, and Majumdar [KRM06] call them a trace of sessionlets. The raw request logs and the preprocessed logs were considered as possible workload representations for this thesis, as they include all required workload information (comprising performed requests to services with session identifiers and timestamps of the requests).

3.2.2. Workload Models

Kistowski, Herbst, and Kounev [KHK14] introduce the LIMBO toolkit for the instantiation of load intensity models. LIMBO is based on two metamodels that allow to describe dynamic and variable load intensity profiles and workload scenarios over time. The metamodels do also consider seasonal patterns, trends, bursts, and noise parts. Their evaluation shows that both metamodels are able to capture real-world load profiles with acceptable accuracy. The approach does not consider the user behavior, which is required in our representation of the workload.

To model the user behavior, Markov chains are one possible representation. Li and Tian [LT03] show that the web usage can be accurately modeled through them. In order to test whether the memoryless property (transition from one state to another only depends on the current state) of Markov chains conforms to actual web link usage frequencies, their approach applies a small set of tests on their university's web site, that can be easily extracted from web server logs by gathering the web link usage frequencies

from them. Their evaluation compares history-dependent and history-independent state transition probabilities. Their results show that the memoryless property of Markov chains conforms to the actual usage frequencies and, therefore, they are an accurate model for the web usage.

Menascé et al. [MAFM99] introduce Customer Behavior Model Graphs (CBMGs) as workload models, which describe the user behavior of customers of session-based systems and navigational patterns. They are based on Markov chains and can be extracted from HTTP server logs with the K-means clustering algorithm. The algorithm identifies similar types of user groups in the logs. In addition to Markov chains, CBMGs include user think times between state transitions.

Vögele et al. [VHS+18] introduce the WESSBAS approach for the automatic extraction and transformation of workload specifications for load testing of session-based systems. Their defined workload model specifies the attributes application model, behavior model, behavior mix and workload intensity. The attributes are described in more detail in Section 2.3. A user of WESSBAS has to set the workload intensity attribute manually. Instances of the defined workload model are extracted from session logs.

Krishnamurthy, Rolia, and Majumdar [KRM06] define a workload model that specifies attributes with statistical characterizations. The attributes are, for example, session interarrival time distribution, think time distribution, and session length distribution. The user behavior is modeled through sequences of requests users perform. An instance of the workload model can be created through the extraction of a trace of sessionlets (sequences of real-world user requests) from raw request logs and then passing the trace to the workload model generation engine.

The workload model from Shams, Krishnamurthy, and Far [SKF06] is based on Extended Finite State Machines (EFSMs). They describe valid sequences of user requests in a session. Transitions from one state to another are labeled with Guards and Actions (GaAs), which are predefined state variables. This is the difference to Markov chains, where state transitions are labeled with probabilities. Further attributes that can be specified in the workload model are think times, session length distributions, and a workload mix.

Further works defining workload models are:

- Draheim et al. [DGH+06] and Lutteroth and Weber [LW08] present workload models that are based on stochastic form-oriented analysis models. They can be used to model realistic user behavior in order to achieve valid load testing results.
- Abbors et al. [AATP12] present a model-based performance testing tool that uses probabilistic models which are based on probabilistic timed automata. The models describe how different groups of users interact with the system and include

statistical information like the distribution between different user actions and think times. Using the models, the tool generates synthetic workload which is applied to the system.

- Zhou, Zhou, and Li [ZZL14] present a generic context-based sequential action model together with a workload parameter specification language to model the workload a system experiences. The action model is used to describe users with similar access patterns. The authors implemented a framework based on their proposed model to generate synthetic workloads.

The above described works do not consider both, workload intensity and user behavior variation over time. LIMBO [KHK14] can be used for modeling the workload intensity variation over time, but does not consider the user behavior. Works in the area of user behavior modeling do not consider time variations of the user behavior. However, both of the required information (workload intensity and user behavior variation over time) are included in raw request logs (and also in the preprocessed logs extracted from raw request logs). The problem is, that forecasts and calculations (required for the later processing of the forecasted workload) cannot be performed well on such data. It is easier to perform calculations on, e.g., the workload intensity. In this thesis, we represent the user behavior through Markov chains. Each Markov chain describes a different user group accessing the application. For each of the user groups, we calculate its past workload intensity. Based on the past workload intensities, we forecast the future workload intensity of each user group and update the occurrence probability of each Markov chain during workload generation. Summarized, our workload representation consists of Markov chains, and the workload intensities of different user groups.

3.3. Workload Forecasting

In the following, two works are presented that deal with workload forecasting. The first work forecasts the workload intensity by analyzing the development of the workload over time. The second work uses model predictive techniques to forecast the future workload in order to do a better capacity planning for cloud applications.

Herbst et al. [HHKA14] consider several workload forecasting techniques based on time-series data for resource management and capacity planning. They classify them regarding their computational overheads and investigate their benefits and drawbacks. Furthermore, they introduce an approach for the selection of appropriate forecasting techniques for a given context. The approach is based on a decision tree. The goal of the approach is to provide reliable forecast results at runtime. Future workload is forecasted by analyzing the development of the workload over time. However, only the

workload intensity is forecasted. The user behavior is not considered. For our approach, beside the workload intensity, we will also forecast the future user behavior. Another difference to our approach is that Herbst et al. [HHKA14] do not extract load tests from the forecasted workload.

Roy, Dubey, and Gokhale [RDG11] present a resource allocation algorithm based on model predictive techniques to forecast the future workload a large-scale component-based enterprise application will experience. The algorithm is used for resource autoscaling in the cloud. The goals of the approach are to allocate and deallocate resources in a way that satisfies the Quality of Service (QoS) an application expects from the cloud provider and to keep operational costs low. The authors focus only on forecasting the future workload intensity, but not the future user behavior. Our approach comprises both and extracts load tests from the forecasted workload. The load tests test for future workload a session-based system will experience. The execution of the load tests will show whether the system can handle the future workload, or if more resources are needed. Thus, our approach can also be used for capacity planning.

Chapter 4

Approach

In this thesis, the main goal is to focus only on the future workload that is relevant for a given context, and to generate load tests which test for the relevant workload. Thereby, we aim to keep the test execution time and the resource usage as small as possible, but still apply the relevant workload on the system. One assumption of our approach is that users want to load-test for future contexts. By passing a context description containing contextual information to our approach, a user will get load tests, that only test the relevant workload.

We already shortly outlined our approach in Section 1.3. A more detailed overview of our approach can be found in Section 4.1. It shows the whole pipeline of the load test generation, that starts by passing request logs to the approach. Individual parts of the pipeline are shortly described. We summarize assumptions we make for our approach in Section 4.2. In the next sections, we break down the pipeline into its parts, consisting of the generation of a session log, the context description language (CDL), the representation and forecasting of workload, and the processing of the forecasted workload, and describe them in detail. We use WESSBAS for our approach to calculate historical workload data. The input to WESSBAS is a session log, that is a special representation of request logs. In Section 4.3, we explain how we generate a session log from request logs. A context description passed by the user has to follow rules of a CDL, that is described in detail in Section 4.4. We use time series forecasters to forecast the future workload. To be processible by such forecasters, workload has to be represented in a way that can be understood by the forecasters. In Section 4.5, we perform a process to output such a workload representation and describe the forecasting approach. Finally, in Section 4.6, we process the forecasted workload and output relevant load tests.

4.1. Forecast Process

In this section, we provide a detailed overview of our approach. We develop a process to forecast the future workload based on historical workload and a context description. The output of the process are tailored load tests for the user. An overview of the forecast process is shown in Figure 4.1. The dashed lines border parts of the process, which we shortly describe in the following. The parts are then described in more detail in the next sections.

Generation of Session Log: We forecast the future workload based on historical workload data and a context description. We use WESSBAS to calculate the historical workload from a session log, that is a representation of request logs WESSBAS can process. We build a processor to convert request logs extracted from an system under test (SUT) into a session log.

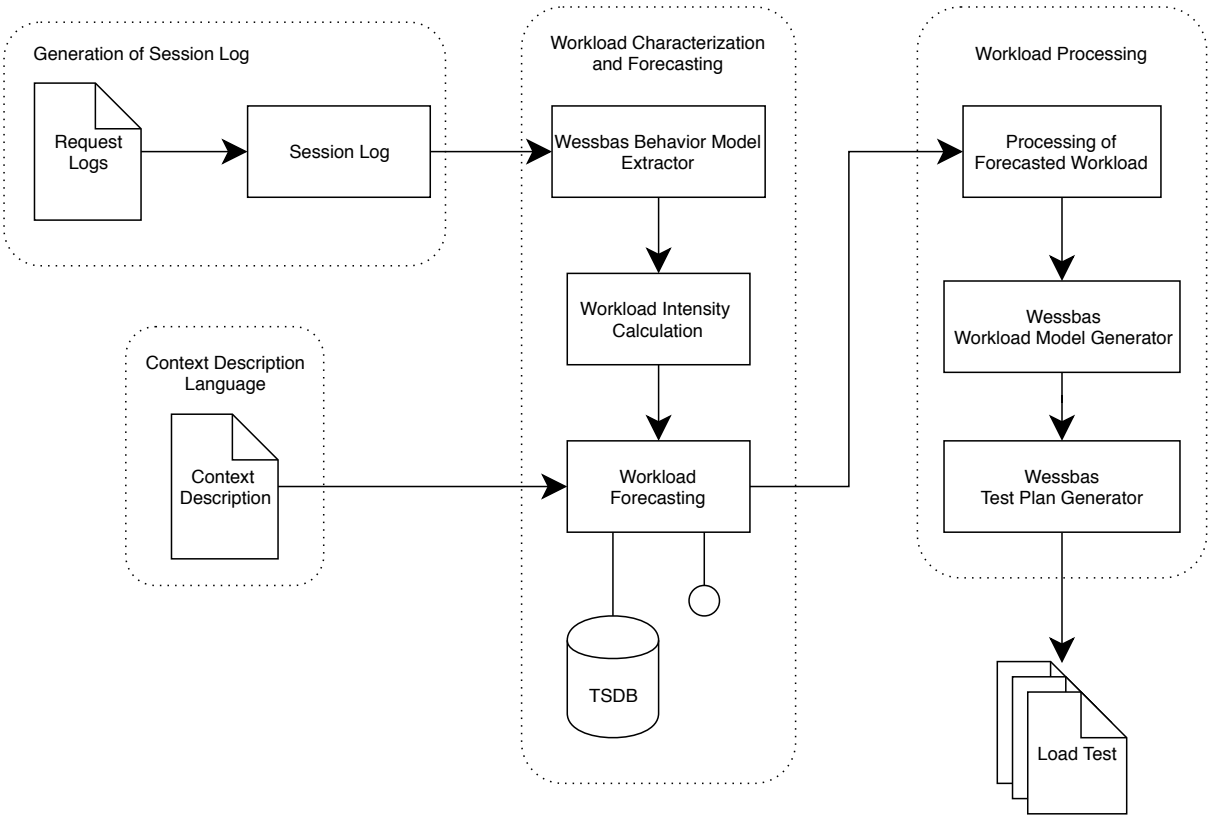

Figure 4.1.: Forecast Process

Workload Characterization: We use time series forecasters to forecast the future workload. For this purpose, we have to represent historical workload in a way that can be understood by the forecasters. As stated in Section 2.1, important characteristics of workload comprise the workload intensity and the user behavior. When forecasting the future workload, we refer to these two characteristics. We calculate the historical workload from a session log. Our assumption is that the log contains all required historical workload information. We use the WESSBAS behavior model extractor to extract behavior models (Markov chains with think times) from a session log. The resulting behavior models represent all different user groups manifested in the log, i.e., with this procedure we calculate the user behavior. However, we cannot input Markov chains to the time series forecasters. Instead, we calculate for each Markov chain (user group) its past workload intensity, which can be expressed through a time series of values. We can do so, since we have session information for each of the chains. The time series is saved into the time series database (TSDB) InfluxDB so that for particular logs of requests the workload intensity has not to be calculated again, when the approach is triggered.

Context Description Language: We can include contexts in workload forecasts. In the scope of this thesis, a context is either an event or a time series (e.g., temperature curve), which has an influence on the workload a session-based system experiences. Our approach has no knowledge about contexts and, hence, all contextual data have to be provided by the user. For this purpose, our approach provides an endpoint, where the user is able to stream contextual data into the InfluxDB. He can stream past as well as future context values. Such values are, e.g., temperature values from the past and future, or past and future occurrences of events. He can stream into a measurement either numerical data or Strings. The user decides by himself, in which measurement the data will retain (by choosing a name, that does not already exist in the InfluxDB, he will create a new measurement). Once contextual data is streamed into the InfluxDB, our approach can access it and consider it in forecasts. The user decides which contexts to include in a particular forecast. For this purpose, the user has to pass a context description to our approach. A context description has to follow rules of a CDL, and consists of one or more context parameters. Each parameter at least tells our approach from which measurement to include contextual data. Additionally, a user is able to pass future context values with each parameter, e.g., that next week's Friday is Black Friday. Future context values passed in the context description have higher priority than future context values already contained in the measurement, and, hence, will override the information.

Workload Forecasting: Before forecasting the workload by passing the past workload intensities to time series forecasters, we process a user's context description input. As time series forecasters we use Telescope [Uni18] and Prophet [fac17], which are able to consider regressors in their prediction models. They expect that for each value in the time series one corresponding value in the regressor exists. In addition, values for the forecast have to exist in the regressor. We will calculate them from the passed context description. Included context parameters tell our approach from which measurements to include contextual data for the forecasting. Depending on the data contained in a measurement, we calculate regressors from a context description as follows:

1. Numerical data: If the context parameter tells our approach to include data from a measurement containing numerical data (e.g., temperature values), we calculate one regressor from the data. It can be seen as an additional time series to the workload intensity. Numerical values are not changed, we insert existing values directly into the regressor. If there are gaps, we insert a 0. Future values passed in a context parameter update the already calculated regressor.
2. Strings (events): If the context parameter tells our approach to include data from a measurement containing events (strings), we calculate for each different string one regressor. One regressor can be seen as a binary indicator whether an event did take place or not. We use 0 for no occurrence (gaps), and 1 for occurrence. Future values passed in a context parameter are transformed to a 1 and update the already calculated regressor.

With the regressors, the forecasters are able to do a more sophisticated forecasting, because influences of events can be learned from the past and considered in the forecasting. For numerical values, they learn the influence of values on the workload, e.g., a higher value leads to a higher workload. The workload intensity of each user group, represented through a time series, and regressors are passed to either Telescope or Prophet, which then predicts the future workload intensity for that user group. The outcome of this process is a time series for each user group, where one particular time series is the forecasted workload intensity of one particular user group.

Workload Processing: In the last step, the forecasted workload intensities have to be processed in order to output load tests. A user of our approach decides for which workload scenarios he wants to test. Such a load scenario could be high workload, or a sharp increase in workload. We cover different load scenarios a user could request for. We aggregate the forecasted workload intensities into one aggregated time series, that is the sum of the values of the forecasted intensities at times t_1, ..., t_n, where t_1 is the timestamp of the first values, and t_n the timestamp of the last values of all forecasted intensities. From the aggregated time series, we extract the load tests. For example,

when the user requested a high workload scenario, we extract the maximum value from the aggregated time series, and set it as the workload intensity of the load test we will return to the user. To update the user behavior, we calculate the occurrence probability of each behavior model during the workload generation. In the example above, we extracted the maximum value from the aggregated time series. This value is the sum of values from the forecasted intensities at a particular time t. To update the occurrence probability of a particular behavior model, we divide the value of the intensity, that was forecasted for the user group that the behavior model describes, by the maximum value. The result is the occurrence probability of the behavior model during the workload generation. When the load test simulates a particular user, a behavior model is then chosen based on the probabilities, and the user navigates through the application as specified by the chosen behavior model. To generate a load test, we pass the extracted workload intensity for the load test and the calculated occurrence probabilities of the behavior models to the WESSBAS workload model generator. This component generates a workload model instance out of the workload intensity, the previously calculated behavior models, and the occurrence probabilities. The workload model instance is then passed to the WESSBAS test plan generator, which converts the workload model instance into a load test. The load test is returned to the user.

In this section, we made a few assumptions, that are summarized in Section 4.2.

4.2. Assumptions

This section provides a list of assumptions we make for our approach. We also list the limitations that are involved with each of the assumptions. They are made to limit the scope of this thesis. The list is shown below.

- **Assumption:** Users want to test for future contexts.
  Limitation: Users are not able to directly test for past contexts, they would have to make a workaround.
- **Assumption:** All required workload and session information is manifested in the request log.
  Limitation: Our approach is restricted to session-based systems.
- **Assumption:** A future context that should be included in the forecast was already observed in the past.
  Limitation: Users cannot test for contexts that have never been observed.
- **Assumption:** Past and future contextual information is provided by the user.
  Limitation: Our approach by itself has no knowledge about contexts.

- **Assumption:** Our approach can access past contextual data.
  Limitation: The past data has to be available to our approach before it is triggered.

4.3. Generation of Session Log

We use WESSBAS for the extraction of behavior models from request logs. As stated in Section 2.3, the input to WESSBAS is a session log, that is a representation of request logs WESSBAS can process. Request logs can be extracted from the session-based SUT and we assume them to contain recorded session information. For instance, such request logs are provided by common web servers in form of HTTP request logs [MAFM99]. In a session log, request entries are grouped by their session identifier into session entries. A request entry contains information of exactly one recorded request. Each session entry lists the sequence of subsequent requests (in form of request entries) in a session [VHS+18]. Figure 4.2 shows an example session entry with two request entries. A session entry starts with a unique session identifier, followed by the sequence of request entries. Request entries are separated with a semicolon. Each request entry has the following information:

1. Requested service
2. Start time of the request
3. End time of the request
4. Request URL
5. Port
6. Host IP
7. Protocol
8. Method
9. Request parameter with values
10. Encoding

For our approach, we build a session log generator that is able to generate a session log from request logs in CSV format. The first row of the CSV represents the column names. They correspond to the information that is contained in a session entry. Each of the following rows represents a request. In Figure 4.2, an example session entry is shown. The CSV from which this entry could have been extracted is shown in Figure 4.3. Some of the columns have been left empty. They could have been omitted completely.

cecef3d42f6cdd008d36e78e20403d14;"studium":1536717132000000000:1536717132000000000:
/studium/login.php:8080:127.0.0.1:HTTP/1.1:GET:<no-query-string>:UTF-8;
"studium":1536717135000000000:1536717135000000000:/studium/verif.php:8080:
127.0.0.1:HTTP/1.1:POST:<no-query-string>:UTF-8;

Figure 4.2.: Example session entry

id	service	start	end	url	port	host ip	protocol	method	parameter	encoding
cecef3d42f6cdd008d36e78e20403d14	studium	1536717132000000000		/studium/login.php				GET		
cecef3d42f6cdd008d36e78e20403d14	studium	1536717135000000000		/studium/verif.php				POST		

Figure 4.3.: Part of example request log in CSV format

It is to show that our approach does not require every information from a request log to build the session log. The minimum information our approach requires are the start time of a request and the request URL or the requested service. The other information can be derived/ mocked as follows:

- Session identifier: To obtain sessions, we use a timeout value of 30 minutes between two subsequent requests [FGL15] in order to calculate the session identifiers.
- Requested service: When the requested service is not available, then the request URL has to be available. We can extract the requested service from the request URL. How this can be achieved is explained for a real-world request log in Section 6.2 in the evaluation chapter.
- End time of the request: We set the request end time to request start time.
- Request URL: When the request URL is not available, then the requested service has to be available. We then set the request url to the service.
- Port: We set “8080” as the port.
- Host IP: We set “127.0.0.1” as the host IP.
- Protocol: We set “HTTP/1.1” as the protocol.
- Method: We set “GET” as the method.
- Parameter: We set “<no-query-string>” when no parameters exist, as in this case this is the String expected by WESSBAS.
- Encoding: We set “UTF-8” as the encoding.

With the session identifiers, our approach groups the requests, so that requests with the same session identifier are in one group. For each group, our approach calculates one session entry. Available request information is directly taken from the CSV. The remaining information is obtained as described above for each request. The approach then writes line by line the calculated session entries into the session log.

4.4. Context Description Language

In the scope of this thesis, a context is either an event or a time series (e.g., temperature curve), which has an influence on the workload a session-based system experiences. For example, sale events of web shops, like a Boxing Day event, typically lead to an increased workload as a lot of users access the website. It is not unusual that web pages then take extremely long to load or that the web application even crashes. This could be prevented by load testing the application before the event occurs and then to prepare for the upcoming scenario. To decrease the test execution time, it suffices to test for the increased workload that occurs during the event. In this thesis, we build an approach to generate load tests which test for the workload that is relevant for one or more given contexts. We assume that all contextual data is provided by the user and that our approach can access it from measurements from an InfluxDB. To test for the relevant workload, the user passes a context description to our approach, which consists of one or more context parameters. Each context parameter tells our approach from which measurement in the InfluxDB contextual data should be included in the workload forecast. We then build regressors out of the data, which can be considered by time series forecasters in their prediction models, i.e., forecasts then consider contexts. By processing the forecast results, we then output context-aware load tests. We summarize and define the terms regarding context we will use throughout this thesis:

- **Context:** An event or time series (e.g., temperature curve) that has an influence on the workload a session-based system experiences.
- **Contextual data:** Past and future occurrences of the event, or past and future values of the time series, i.e., one or more context values.
- **Context value:** Either an event occurrence or one value of the time series.
- **Context description:** Used to describe one or more contexts. Consists of one or more context parameters.
- **Context parameter:** Specifies at least from which measurement contextual data should be included in the forecast. Furthermore, a user can pass future context values with the parameter.

In this section, we design a description language for context descriptions. They have to follow the rules of the language. How a context description is then processed by our approach is explained in more detail in the next section, Section 4.5.

Contexts impacting the workload can be very different for different session-based systems. Hence, before we design the CDL, we first collect occurred real-world contexts that had an influence on the workload of the affected application. They are listed in Section 4.4.1. Based on the collected real-world contexts, we explain design decisions of the CDL in Section 4.4.2. The metamodel is presented in Section 4.4.3, where we describe syntax and semantics of provided context descriptions.

As already mentioned in Section 4.2, to limit the scope of this thesis, we assume in the following that all contextual information is provided by the user and that contexts occurring in the future already have been observed in the past.

4.4.1. Observed Contexts in Real World

Table 4.1 and Table 4.2 list observed real world context that had an influence on the affected application. When designing the CDL, we want to be able to map all of the collected contexts with our language. For this purpose, we categorize the found contexts by their context type, which we could identify from the collected real world contexts. The types are:

- **Recurring event**: An event, that was already observed in the past. It has an impact on the workload, when it occurs. It lasts for a particular time span and reoccurs at a particular point of time.
- **Unpredictable event**: An event, from which it is not known when it will occur. This type can be splitted into two subtypes:
  1. **Observed unpredictable event**: An event, that was already observed in the past. It is not known, if and when the event will reoccur. It has an impact on the workload, when it occurs. It lasts for a particular time span.
  2. **Once-in-a-lifetime event**: An event, that was never observed in the past. It is not known, when it will occur, how long it will occur, and what the impact on the workload will be. It occurs (with high probability) once in a lifetime and can have a great impact on the experienced workload.
- **Measurements**: A time series of measured numerical values that can change over time. Changes in the values mean a change on the experienced workload.

Date	Context	Context Type	Impact	Source
-	Weekend/ holiday effects (all websites)	Recurring event	Web traffic increases/ decreases during weekends and holidays	[BRA16]
-	Temperature effects	Measurements	Web traffic for web shops depends on the weather/ temperature	[Wea14]
26/12/2012	Boxing Day	Recurring event	High traffic due to special offers	[Gee12]
03/03/2014	Twitter tweet during Oscars with many film stars	Once-in-a-lifetime event	Twitter was disrupted for 20 minutes	[new14]
01/10/2016	Release of new TV series on a rainy Saturday	Recurring event/ Continuous data	Netflix went down for 2.5 hours due to high traffic	[USA16]
24/11/2017	Black Friday and new sneaker releases	Recurring event(s)	High traffic due to Black Friday offers and additionally new limited releases	[Int17]
2018	Outage of message endpoint	Observed unpredictable event	Buffered messages from devices are send all at once when the endpoint recovers, leads to a peak load	internal
28/02/2018	Denial of service attack	Observed unpredictable event	High traffic through simulated users, GitHub went down for 10 minutes	[WIR18]

Table 4.1.: Observed contexts in real world (1)

Date	Context	Context Type	Impact	Source
29/05/2018	Question in the “Wer wird Millionär?” German quiz show [RTL19] about existence of German words in the German Duden	Once-in-a-lifetime event	duden.de [Bib19] was not reachable	internal
04/06/2018	Project imports from GitLab to GitHub	Once-in-a-lifetime event	High traffic due to increase of project imports when GitHub was acquired by Microsoft	[t3n18]
06/07/2018	Publication of a summertime survey	Recurring event	Server went down due to high traffic	[EUS18]
16/07/2018	Amazon Prime Day	Recurring event	High traffic due to special offers	[BGR18]

Table 4.2.: Observed contexts in real world (2)

4.4.2. Design Decisions

As mentioned in the introduction to this section, contexts impacting workload are not the same for all kind of applications. Therefore, it is not meaningful to predefine a pool of contexts and restrict the user to this pool. For a specific application, a very specific context could impact the experienced workload, that does not apply for other applications. Therefore, we decide to let the users define their own contexts. However, in Section 4.4.1, we identified three categories of contexts which we assume to cover all specific contexts. This means, when users define their own context, we assume it to be one of the identified context types in Section 4.4.1.

To limit the scope of this thesis, we made a few assumptions in Section 4.2. Two of the assumptions we will apply here are:

1. A context that will or is assumed to occur in the future already has been observed in the past
2. Contextual data is provided by the user

We exclude contexts, that never have been observed in the past, since we have no information about them and we do not know what the impact on the workload will be when they occur. From a provided context description, our approach will build regressors that can be passed to time forecasters, where a regressor includes past as well future values of a context. When there are no past values, they cannot consider the regressor in their prediction model. We first investigate for each context type, if there is a conflict with the first assumption:

- **Recurring event**: Since such an event was already observed in the past, we assume that past values are available.
- **Unpredictable event**: An unpredictable event is one of two subtypes:
    1. **Observed unpredictable event**: It cannot be foreseen, when and if this event will occur. However, this event was already observed in the past, and we assume that our approach has access to past values. Hence, we can utilize these types of contexts for what-if analyses. For that, a user has to specify the assumed future contexts.
    2. **Once-in-a-lifetime event**: Since this event never occurred in the past, our approach cannot access past values. Hence, the user would have to provide necessary information in the context description in order to simulate an event according to his wishes. We will not consider once-in-a-lifetime events in our CDL, since there is a conflict with our assumption. However, we suggest for future work to extend the CDL to cover once-in-a-lifetime events.
- **Measurements**: We assume that our approach can access numerical values from the past.

Shortly summarized, our approach will cover each context type, but not the once-in-a-lifetime event subtype of unpredictable events. We can handle recurring and observed unpredictable event in the same way, as in both cases we assume that our approach has access to past values of such events. Hence, we merge them in the following together and refer to them as the **event** context type.

We expect that contextual data from the past and future is available to our approach. The information has to be provided by the user. Our approach provides an endpoint that can be used to stream contextual data into a database (DB). As DB we use the TSDB InfluxDB, where a DB table is called measurement. The user is able to stream past as well as future values. The user decides in which measurement(s) the data should be stored. He then has to tell our approach, from which measurements contextual data should be included for the forecast. Past values of contexts must be stored into the DB. For future values of contexts, we decide that they can be stored into the DB, and that

the user should also be able to pass them directly in the context description. This has the following advantages:

- When future values are known, they still can be streamed into the DB.
- When future values are not known, or the user wants to test what would be if the future values were different, he still can pass them in the context description.

Users will be able to describe more than one context in one context description. The user can describe contexts of different context types, e.g., temperature (measurement context type) and a special event like Boxing Day, as well as contexts with the same context type, e.g., two special events, in one context description. For each context, our approach will build exactly one regressor. All regressors are then included to the forecasts.

With the decisions we made here, we are now able to build the metamodel of the CDL. It is described in the following.

4.4.3. Metamodel

The metamodel of our CDL should be as simple as possible, but at the same time a user should be able to describe contexts of event and measurement context type. Identified requirements on the CDL are:

1. The user defines contexts by himself, we do not predefine any context
2. We cover events and measurements
3. In one context description, the user can describe several contexts

We want to build upon existing appropriate metamodels. Contextual data has to be provided by the user. Our approach has an endpoint, where users can stream contextual data into the InfluxDB. He is able to stream past as well as future values into the DB. Since our approach uses the Influx Query Language (InfluxQL) to read the contextual data, we build our metamodel directly upon the InfluxQL metamodel [Inf19a].

We first start with an example context description, that follows the syntax of our metamodel. It is shown in Listing 4.1. The user passes three context parameters to the approach, which are the three list entries of the `context` list. The value of `measurement` tells our approach the name of the measurement containing past and future values of contexts to include in the forecast. The first parameter, “number of offers”, consists only of the measurement name. For “temperature”, the user did pass future temperature numbers and the dates where the numbers will occur. The same applies for “marketing campaigns”. The only difference is that the user did not pass future numbers here, but Strings representing future events. Future events in this example are `Sale 30%` and

Listing 4.1 Example context description

```
context:
- measurement: number of offers
- measurement: temperature
  future:
  - value: 30.0
    time:
    - 2019/07/22 12:00:00 to 2019/07/22 15:00:00
  - value: 35.0
    time:
    - 2019/07/23 14:00:00
- measurement: marketing campaigns
  future:
  - value: Sale 30%
    time:
    - 2018/07/22 10:00:00 to 2018/07/22 20:00:00
  - value: Sale 70%
    time:
    - 2018/07/23 10:00:00 to 2018/07/23 20:00:00
```

`Sale 70%`. In order to understand individual parts of the example context description better, we have to anticipate some DB design decisions, that are described in detail in Section 4.5. When users stream past and future values of contexts into the InfluxDB, they have to pass a measurement name to our approach in order to locate the measurement where the values should be saved into. A name, that is not yet connected with a measurement, will lead to a new measurement. However, we recommend the user to store belonging numerical values (e.g., temperature values) and events from one category (such categories can be, e.g., marketing campaigns, holidays, sports events, etc.) respectively into one measurement. For example, “Sale 30%” and “Sale 70%” should be located in the “Marketing Campaigns” measurement. Another example would be that “Labor Day” and “Christmas” should be stored together in the “Holidays” measurement. A user can also store single events into an own measurement, e.g., he only stores occurrences of “Boxing Day” into the “Boxing Day” measurement. When the user wants to include values from a measurement to the workload forecasting, he has to pass the name of the measurement. In a measurement, past as well as future values of contexts can be stored. Future values not necessarily have to be included in a measurement, they can also directly be passed in the context description with a context parameter. Passed future values with the context parameter have higher priority than future values in the measurement. They overwrite the existing data. In the example description above, future context values were passed with the “temperature” and the “marketing campaigns” parameter. With this DB design, we already covered the first requirement on the CDL, since the user is able to define contexts by himself by just providing the name

of measurements he wants to be considered for the forecasting. Additionally, he is able to describe future context values directly in the context description.

We only need numerical values for the measurements context type, and Strings for the event context type. In the context description example above, the “number of offers” and “temperature” parameters lead to measurements where numerical values are stored. The “marketing campaigns” parameter leads to a measurement where Strings are stored. From the contextual data we build regressors and pass them to time series forecasters. Numerical values are directly inserted into one regressor. Strings (events) are transformed into binary indicators (0 - no occurrence, 1 - occurrence). For each different event in a measurement, one regressor is build, since the forecaster should learn different impacts of different events. Timestamps are always covered by InfluxDB, and, hence, we have in addition one field “value”, that is either numerical or a String. We have fulfilled the second requirement, since we cover events expressed through Strings, and measurements expressed through numerical values.

The third requirement is covered by allowing for a series of context parameters in one context description, as shown in the example context description above. Besides, more than one context can be included to the forecast with only one context parameter, since a context parameter can lead to a measurement with more than one event. We now prepared everything to specify the syntax and semantics (metamodel) of our CDL. In Section 4.4.3, we first introduce the syntax of a context description. It is based on YAML [19b], since we use it as the serialization format for context descriptions. In Section 4.4.3, we transform the context description to constructs and semantics of the InfluxQL metamodel.

Syntax

Our syntax is based on the YAML syntax, since we use it as data serialization format. However, we limit the user to a subset of YAML constructs. We introduced YAML and required constructs in Section 2.6. The CDL syntax shown in Listing 4.2 is defined using the Extended Backus-Naur Form (EBNF) notation [Sta96; W3C17] and shows allowed Strings and YAML constructs. For better readability, we illustrate the EBNF using railroad diagrams [Rad18]. We explain the syntax with the help of the diagrams. The round boxes represent terminal symbols, and the squared boxes nonterminal symbols. Spaces, which are represented through empty boxes in the diagrams, are used for the right indentation in the YAML and to distinguish a key from its value, and are not mentioned in the following. The same applies for “-” in a yellow box, which indicates a new list element and for “\n”, which expresses new line.

Listing 4.2 Syntax of a context description

```
Context ::= "context:" (NewLine "-" Space Parameter)+
Parameter ::= MeasurementAndFuture | Measurement
MeasurementAndFuture ::= Measurement NewLine TwoSpaces Future
Measurement ::= "measurement:" Space String
Future ::= "future:" (NewLine TwoSpaces "-" Space FutureType)+
FutureType ::= FutureNumber | FutureEvent
FutureNumber ::= "value:" Space Number NewLine FourSpaces Time
FutureEvent ::= "value:" Space String NewLine FourSpaces Time
Time ::= "time:" (NewLine FourSpaces "-" Space FutureDate)+
FutureDate ::= DateInstance (Space Operator Space DateInstance)?
DateInstance ::= /* Valid instance of yyyy/MM/dd HH:mm:ss */
Operator ::= "to"
Number ::= /* A double value */
String ::= Char+
Char ::= /* Unicode character (except newline) */
NewLine ::= "\n"
Space ::= " "
TwoSpaces ::= Space Space
FourSpaces ::= Space Space Space Space
```

Figure 4.4 shows the root entity of the context description, which is a list. The key of the list is “context”. The user then has to pass a list of context parameters (one context parameter is expressed through `Parameter`) to our approach (at least one).

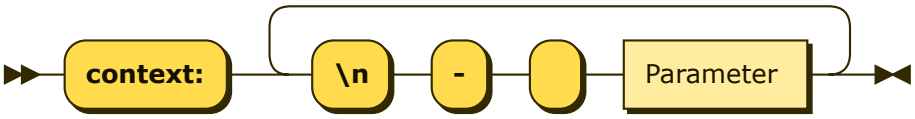

Figure 4.4.: Railroad diagram: Context

As shown in Figure 4.5, a context parameter is either a `Measurement`, or a `MeasurementAndFuture`.

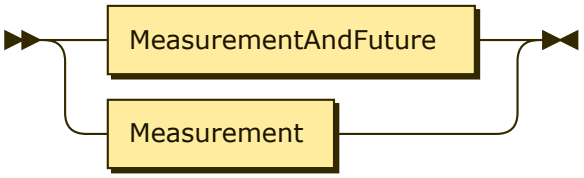

Figure 4.5.: Railroad diagram: Parameter

Figure 4.6 shows the `MeasurementAndFuture`. It represents an object consisting of `Measurement` and `Future`, i.e., the user passes to the approach a measurement name,

and also future context values. `Future` represents a second attribute of the object and has to start in a new line.

Figure 4.6.: Railroad diagram: MeasurementAndFuture

`Measurement`, as shown in Figure 4.7, consists of the String "measurement:" followed by a String value provided by the user. The String represents the measurement name of the measurement containing context values which should be considered for the forecasting.

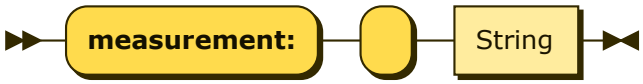

Figure 4.7.: Railroad diagram: Measurement

Beside a measurement name, the user can also pass future values as a list in a context parameter. This is shown in Figure 4.8, where the key of the list is "future", and instances of `FutureType` are listed in the next lines, where each `FutureType` consists of one future context value and belonging dates to that value.

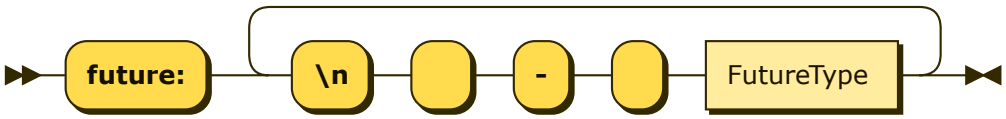

Figure 4.8.: Railroad diagram: Future

As shown in Figure 4.9, `FutureType` is either `FutureNumber` or `FutureEvent`.

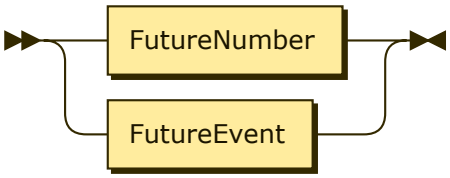

Figure 4.9.: Railroad diagram: FutureType

`FutureNumber`, as shown in Figure 4.10, starts with key "value", followed by a numerical value passed by the user (`Number`). It is a future context value passed by the user. Our

approach also needs the future dates, where the values will occur. Dates are passed in the next line and are referred to as `Time`, which represents a list of date elements.

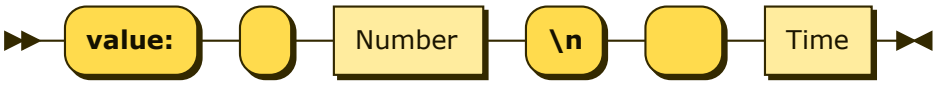

Figure 4.10.: Railroad diagram: FutureNumber

`FutureEvent` is similar as `FutureNumber`, as can be seen in Figure 4.11. The only difference is that the user passes a String instead of a numerical value to our approach. The passed String represents an event that will occur in the future.

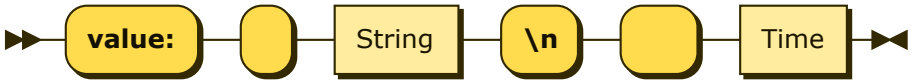

Figure 4.11.: Railroad diagram: FutureEvent

For each future value, our approach needs at least one future date where the value will occur. Figure 4.12 shows a visual representation of `Time` and how dates have to be passed in the YAML. The user provides a list with key "time", and lists in the next lines instances of `FutureDate`. He has to provide at least one `FutureDate`.

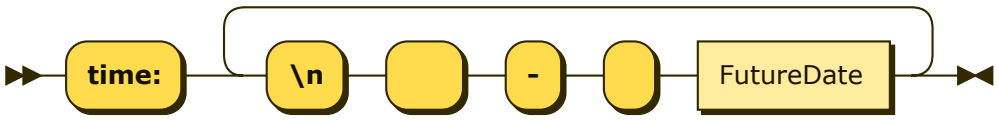

Figure 4.12.: Railroad diagram: Time

A `FutureDate` can be passed in two ways to the approach. The user either can list a single `DateInstance`, or even a range of dates. A range of dates starts with a `DateInstance` and is followed by the String "to", followed again by a `DateInstance`.

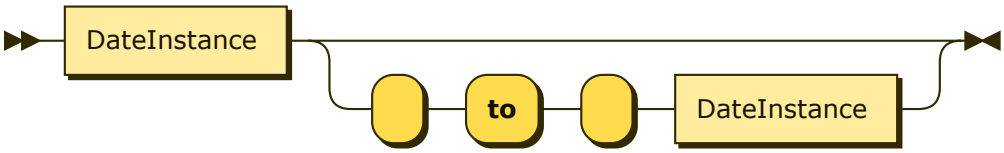

Figure 4.13.: Railroad diagram: FutureDate

We now explain how a context description that follows the above syntax is interpreted.

Semantics

As already explained in Section 4.4.3, we build our metamodel upon the InfluxQL metamodel. A context description can be interpreted as select and insert statements. For each listed context parameter in the context description, a user either provides only a measurement name, or a measurement name combined with future values. To retrieve the contextual data from a measurement, we can simply run a select statement against the InfluxDB:

SELECT * FROM <name of measurement>

where <name of measurement> is the corresponding value to the key `measurement` of a passed context parameter. Additionally, when the user passed future values beside the measurement name with the context parameter, we handle them like an insert statement:

INSERT <name of measurement> value=<passed value> <timestamp>

where <passed value> is a corresponding future value passed with the context parameter and <timestamp> is derived from a passed date where the value will occur. Figure 4.14 visualizes with the help of an example how the statements are extracted from a passed context parameter. In the example, <name of measurement> is “temperature”, <passed value> is 35.0, and <timestamp> is 1563883200000, which is a milliseconds timestamp calculated from the date “2019/07/23 14:00:00” (passed as String). In comparison to the select statement, we do not run the insert statement in practice against the InfluxDB, i.e., future values passed in the context description are not written into the DB. However, the semantics is similar, since we insert the passed future values to the already extracted data set obtained through the select statement. The timestamps are calculated from the passed dates for a future value. We perform this statement for each future value and for each of its corresponding dates. If future values with equal timestamps already existed in the measurement, they will be updated by the new values. Otherwise, the values are simply inserted to the existing data set.

More information on processing contextual data is provided in the next chapter, Section 4.5.

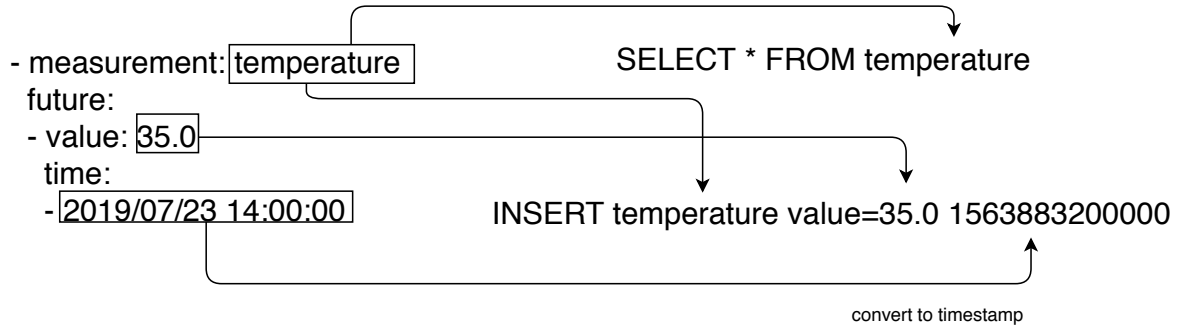

Figure 4.14.: Extraction of select and insert statement from a passed context parameter

4.5. Workload Characterization and Forecasting

In this section, we present the first part of our main approach, that consists of forecasting the future workload based on a context description. We first have to represent past experienced workload in a way, so that it can be used as input to time series forecasters, which will predict the future workload. As stated in Section 2.1, the workload a session-based system is experiencing is characterized by intra-session and inter-session metrics, where intra-session metrics especially characterize the user behavior and inter-session metrics the amount of active sessions over time (workload intensity). Further metrics are provided in Section 2.1. For our approach, we require variations of the workload over time in order to do meaningful forecasts on the workload data. We aim to represent past workload to include user behavior and workload intensity variation over time, and apply forecasting on these two characteristics.

In Section 3.2, we already discussed several approaches to represent workload. Summarized, possible found representations are:

- Raw request logs
- Preprocessed logs, like a session log extracted from raw request logs
- Workload intensity models
- User behavior models
- Workload models modeling both, user behavior and workload intensity

Extracted request logs from an SUT and preprocessed logs contain all required workload information, including variations of the workload over time. However, we cannot perform calculations well on these representations and we cannot input a log to a time series forecaster. Approaches like WESSBAS [VHS+18] exist, that calculate workload models from such logs. However, they either do not model variations of workload

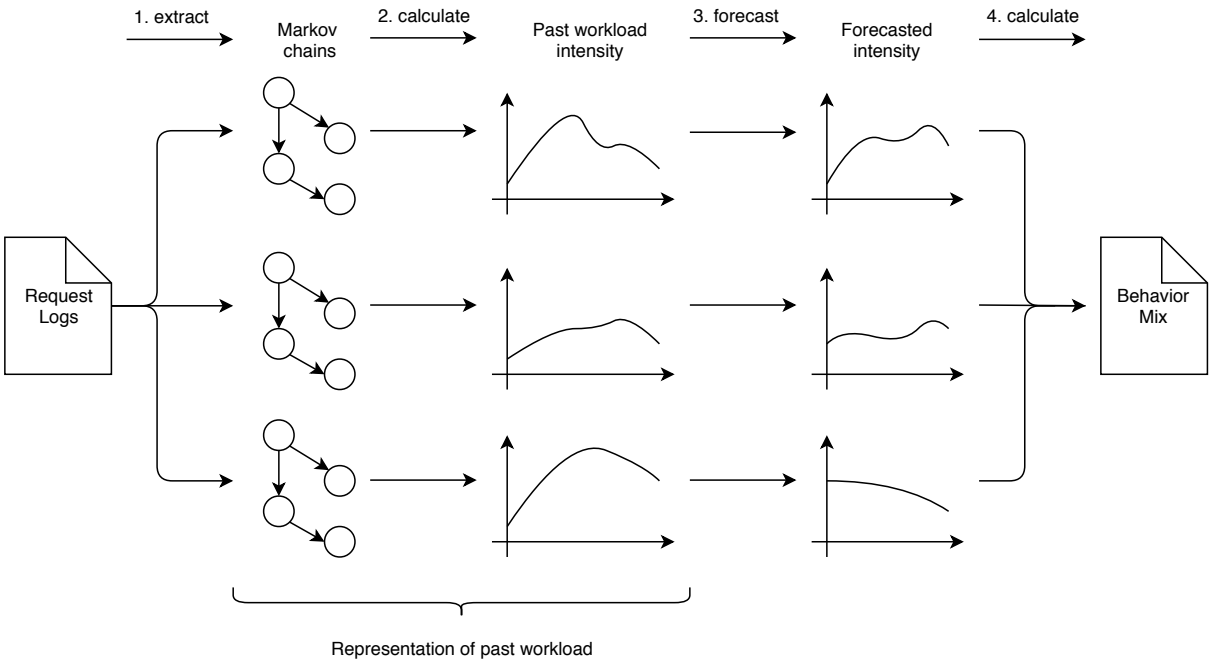

Figure 4.15.: Forecast of user behavior and workload intensity

over time at all, or their models are missing either user behavior or workload intensity variation over time. In Section 4.5.1, we characterize workload and output a workload representation which covers both and which can be used for forecasts. After we represented past workload in an appropriate way, we can perform forecasts. A user of our approach not necessarily has to pass a context description to our approach to forecast the future workload. The workload forecasting without context is described in Section 4.5.2. Details on the context-based forecasting are provided in Section 4.5.3.

4.5.1. Workload Characterization

In Section 3.2, we provided related work on modeling workload and discussed possible workload representations. An appropriate approach to model user behavior are Markov chains, where each Markov chain represents a particular type of user group. However, our approach will build on existing time series forecasters, and we cannot input Markov chains to them. For example, Prophet and Telescope require as input a time series of numerical values to perform forecasts. Workload data, that can be considered for this case, is the workload intensity, as it can be described through a time series of numerical values describing the amount of active sessions over time. In order to forecast both, the future user behavior and future workload intensity, we can perform the steps shown in Figure 4.15. The steps are:

1. Identify all different type of users (user groups) accessing the system (e.g., through request logs extracted from the system) and model the user groups through Markov chains
2. Calculate a time series representing the past workload intensity for each Markov chain, i.e., the past amount of active sessions over time for a particular user group
3. For each Markov chain, pass the time series to a forecaster and let it predict the future workload intensity for each chain
4. Calculate the occurrence probability of each Markov chain during workload generation based on the forecasted intensity and save these into the Behavior Mix

Using this approach, the future workload intensity is predicted directly using time series forecasters. We forecast the future intensity for each Markov chain separately. By processing and aggregating the resulting time series of predicted values, we will be able to calculate the occurrence probability for each Markov chain during workload generation. The probabilities are saved into the Behavior Mix, which describes the future user behavior. More information on the workload processing is provided in Section 4.6. Summarized, for our approach, we will represent past workload through Markov chains and the workload intensity as a time series for each of the chains.

As described in the following, we can use the WESSBAS approach to calculate Markov chains from a session log. The past workload intensity then can be calculated from the session entries that were considered for each chain.

Leveraging WESSBAS

The outcome of our forecast process will be one or more tailored load tests for the user. WESSBAS is already capable of extracting a load test from workload model instances. We decide to build our approach upon the WESSBAS workload model (see Section 2.3). These are the advantages when we leverage WESSBAS:

- WESSBAS' **behavior model extractor** component extracts behavior models (Markov chains enriched with think times) from a session log. We can use this component to calculate the Markov chains.
- WESSBAS specifies a **behavior mix**, which assigns probabilities to each of the behavior models describing their occurrence probability during workload generation. We can manipulate the Behavior Mix after the workload forecasting to assign updated probabilities to the chains.

- WESSBAS' **workload model** specifies intra- and inter-session metrics (including user behavior, workload intensity and behavior mix). We can use this workload model to cover important workload characteristics required to accurately model the workload a session-based system experiences.
- WESSBAS' **workload model generator** extracts instances of the workload model based on the previous calculated behavior models. We can use it after the workload forecasting to generate workload model instances.
- WESSBAS' **test plan generator** is already capable of extracting load tests from workload model instances. We can leverage it to generate load tests.

However, we could identify the following challenges when using WESSBAS individually:

- WESSBAS takes the average or maximum amount of active sessions observed from the session log, and sets is as the intensity that should occur during workload generation. WESSBAS does not forecast the future intensity.
- WESSBAS does not consider future contexts, which could impact the distribution of users to user groups

Hence, the workload generated through the load tests could be not representative. With our approach, we will address the challenges by forecasting future workload based on context descriptions, as described in detail in Section 4.5.3.

Calculation of Markov Chains

We use WESSBAS' behavior model extractor to calculate behavior models from a session log. In Section 4.3, we discussed how to extract a session log from request logs and, therefore, can assume here that they are already available. WESSBAS' behavior model extractor utilizes the X-means clustering algorithm [PM+00] to extract the Behavior Mix and corresponding behavior models from a session log. For this purpose, session entries in the session log are first transformed into vectors on which clustering can be applied. Vectors representing session entries with similarities in performed requests are assigned to one cluster. Since one session entry represents the sequence of requests one user made, this process identifies users with similar navigational patterns. Such users are grouped into one user group, which is then characterized by a behavior model. The more session entries were considered for a resulting behavior model, the higher the occurrence probability is in the mix. More details on the calculation of the Behavior Mix and behavior models are described in detail in the WESSBAS paper [VHS+18].

We extend the behavior model extractor for our purpose, because WESSBAS does not save a list of the session entries that were considered for a particular behavior model. We require this information in order to calculate the past workload intensity for each chain. WESSBAS converts session entries into vectors for the clustering. After clustering, the information of the assignments of the initial vectors to the resulting clusters is available. Since an initial vector corresponds to one session entry, we can infer assignments of session entries to the behavior models and, hence, save for each behavior model a list of belonging session entries.

Workload Intensity Calculation

In the previous part, belonging session entries were assigned to each Markov chain (behavior model). One session entry covers session information that is required by WESSBAS in order to build a workload model instance. The session information is presented in Section 4.3. For our approach, we can make use of the contained session information in order to calculate the past workload intensity for each user group. We will represent the workload intensity as a time series of intensity values, where one data point in the time series (intensity value and timestamp) describes the amount of active sessions at a particular point of time. Such a time series can be the input to a time series forecaster. To calculate the intensity values, we divide the time range in which all sessions occurred into sub-ranges. From the sessions that lie in a particular sub-range, we then calculate the intensity value for that sub-range.

Before we present the workload intensity calculation algorithm, we introduce some design decisions that will impact the algorithm. These are:

- All sub-ranges have the same length. This guarantees equidistance between calculated intensity values, since the timestamp of one value is set to the start timestamp of the sub-range for which it was calculated.
- Sub-ranges have to start at a full time unit, that is either full second, minute, hour or day. Thereby, we will be able to directly assign regressor values to each calculated intensity value, because timestamps of regressor values describe the same full time unit. More information on regressors is provided in Section 4.5.3.

The user tells our approach about the time unit for which intensities should be calculated. In order to do so, a user has to pass his choice in the same YAML file that is used to pass a context description to our approach. For this purpose, beside the `context` list (list of context parameters), a user adds in the YAML file a `forecast-options` dictionary holding key-value pairs (representing an object with attributes), where one of the key-value pairs has as key time unit, and the value either is second, minute, hour or day. Forecast options represent required information from the user in order to perform the forecasting.

Further options comprise, e.g., how long the forecast should last. An example YAML file that comprises both, a context description and forecast options, is provided later in the sections about forecasting. There we define the remaining key-value pairs. Summarized, the passed time unit will be used

1. to infer at which full time unit sub-ranges have to start (e.g., to full hour) and
2. to infer the length of a sub-range (e.g., if the passed time unit is hour, then the length of the sub-range is also one hour).

For each Markov chain, we have a list of session entries. One session entry contains session information of exactly one recorded session. Important information are the session identifier, start times of the requests, and end times of the requests. From the start and end times, we can easily extract the start and end time of a session, by picking the lowest start time and the highest end time of all requests recorded for that session. The session identifier, start time, and end time of a session are the session information required by our algorithm. The information is contained in the session entries, which we input to our algorithm. We call them sessions in the following.

We can subclassify the workload intensity calculation algorithm into two algorithms. The first algorithm calculates the sub-ranges and assigns to each sub-range the sessions that lie in that sub-range. It then passes each sub-range and belonging sessions to the second algorithm, which calculates the intensity value for that sub-range. The result is passed back to the first algorithm, which then saves all values with belonging timestamps into a map, that represents the intensity time series. We start by explaining the first algorithm, named $WorkloadIntensityCalculation$, with the help of the pseudocode that is shown in Algorithm 4.1. The inputs are sessions and the sub-range length, that is either second, minute, hour, or day. The format of the length are nanoseconds. We first order the sessions by their start times in ascending order. In line 4, we set $t_{startTimeFirstSession}$ to the start time of the first session, before in line 6 it is rounded down to the next full unit of time, that can be inferred from the sub-range length. Analogous, $t_{highestEndTimeSessions}$ is first set to the highest end time of all sessions, and then rounded up in line 7 to the next full unit of time. The overall time range for which intensity values will be calculated is then the difference between $t_{highestEnd}$ and t_{start} (line 9). In line 10, The amount of sub-ranges that have to be considered is calculated by dividing the overall time range by the sub-range length. Since we rounded $t_{startTimeFirstSession}$ down to the next full time unit (t_{start}), we ensure in line 13 until 17 in the for loop, which calculates the sub-ranges, that all sub-ranges start at a full time unit by starting from t_{start}, and then consecutively adding the sub-range length to it. The result is a list of sub-ranges. From line 22 until 30, to each sub-range considered sessions are assigned. These basically occur within the sub-range. In the algorithm a session is considered for a sub-range, when

- the start time of the session is contained in the sub-range or

Algorithm 4.1 Workload intensity calculation algorithm for a user group

```
function WorkloadIntensityCalculation(sessions, subRangeLength)
    sort sessions by their start times in ascending order

    t_startTimeFirstSession = sessions.get(0).getStartTime()
    t_highestEndTimeSessions = getHighestEndTime(sessions)
    t_start = roundDownToNextFullTimeUnit(t_startTimeFirstSession)
    t_highestEnd = roundUpToNextFullTimeUnit(t_highestEndTimeSessions)

    overallTimeRange = t_highestEnd - t_start
    amountOfSubRanges = overallTimeRange / subRangeLength
    subRangesList = ∅

    for i = 0 to amountOfSubranges.size() - 1 do
        subRange = range(t_start, t_start + subRangeLength)
        subRangesList.add(subRange)
        t_start = t_start + subRangeLength
    end for

    intensity = ∅
    for all subRange ∈ subRangesList do
        sessionsInSubRange = ∅
        for all session ∈ sessions do
            t_startOfSession = session.getStartTime()
            t_endOfSession = session.getEndTime()
            rangeOfSession = range(t_startOfSession, t_endOfSession)
            if subRange ∈ rangeOfSession
                || t_startOfSession ∈ subRange
                || t_endOfSession ∈ subRange then
                sessionsInSubRange.add(session)
            end if
        end for
        valueOfSubRange =
            ValueCalculation(subRange, sessionsInSubRange)
        intensity.put(subRange.min, valueOfSubRange)
    end for
    return intensity
end function
```

- the end time of the session is contained in the sub-range or
- the session ranges completely over the sub-range, i.e., the start time of the session is before the start time of the sub-range and the end time of the session is after the end time of the sub-range.

In order to check if the session ranges completely over the sub-range, we build a range out of the session in line 25, where the start is $t_{startOfSession}$ and the end is $t_{endOfSession}$. If the range of the session contains the sub-range, then the session is considered for the sub-range. One session can be considered for more than one sub-range, since it can occur in more than one. For each sub-range, a list of considered sessions is calculated. The intensity value for that sub-range is then calculated by calling in line 33 $ValueCalculation(subRange, sessionsInSubRange)$, passing the sub-range and the considered sessions for that sub-range to the $ValueCalculation$ algorithm. In line 34, the resulting intensity value $valueOfSubRange$ is put into the $intensities$ map, where the key is the start timestamp of the sub-range, and the value is the intensity value.

The algorithm to calculate an intensity value, named $ValueCalculation$, is shown in Algorithm 4.2. It is called by the $WorkloadIntensityCalculation$ algorithm in line 33.

Algorithm 4.2 Calculation of the workload intensity value for a particular sub-range

```
    function ValueCalculation(subRange, sessionsInSubRange)
        sumOfTime = 0
        t_startOfSubRange = subRange.min
        t_endOfSubRange = subRange.max
 5:
        for all session ∈ sessionsInSubRange do
            t_startOfSession = session.getStartTime()
            t_endOfSession = session.getEndTime()
            if t_startOfSession < t_startOfSubRange then
10:             t_startOfSession = t_startOfSubRange
            end if
            if t_endOfSession > t_endOfSubRange then
                t_endOfSession = t_endOfSubRange
            end if
15:         lengthOfSession = t_endOfSession - t_startOfSession
            sumOfTime+ = lengthOfSession
        end for

        return sumOfTime / (t_endOfSubRange − t_startOfSubRange)
20: end function
```

The inputs to the algorithm are a sub-range for which the intensity value should be calculated and the sessions that occur in the sub-range. The algorithm sums up the session lengths of all sessions that lie in the passed sub-range, and then divides the session length by the sub-range length to obtain the overall amount of active session for the sub-range. As a particular session could have started earlier and/or ended later than the sub-range, we maybe have to shorten its session length to ensure that only the session length that is relevant for the sub-range is considered. For this purpose, as we iterate from line 6 until 17 over all sessions, we check in line 9 for each session whether $t_{startOfSession}$ is smaller than the minimum of the sub-range ($t_{startOfSubRange}$). If this is the case, we set $t_{startOfSession}$ to $t_{startOfSubRange}$. Analogous, when $t_{endOfSession}$ is higher than the maximum of the sub-range, it is set to $t_{endOfSubRange}$. In line 15, the considered session length is the difference between $t_{endOfSession}$ and $t_{startOfSession}$. In line 16, the variable $sumOfTime$ is increased by the session length. The algorithm returns the overall amount of active sessions for the passed sub-range by dividing $sumOfTime$ by the sub-range length.

The workload intensity calculation algorithm is executed for each Markov chain (user group). Hence, we have a time series of values for each of them. At this point, it could be that not all time series start and end at the same time (depends on the smallest start time and highest end time of all session entries considered for a user group). The considered session log contains recorded requests from users in a particular time range. At the beginning of this time range, some user groups could have been inactive, while others were active. Analogous, at the end not all user groups must have been active. This information is currently missing in the time series, i.e., we have to add zero intensity values to the time series if necessary. For this purpose, we first search for the smallest and highest timestamp of all time series. For each time series, we proceed as follows:

- We compare the timestamp of the first value with the smallest timestamp. If the timestamp of the first value is higher, we subtract the sub-range length from the timestamp, and insert a zero value with the resulting timestamp to the time series. We repeat this until the resulting timestamp equals the smallest timestamp of all time series.
- If the timestamp of the last value in the time series is smaller than the highest timestamp, we add the sub-range length to the timestamp, and insert a zero value with the resulting timestamp to the time series. We repeat this until the resulting timestamp equals the highest timestamp of all time series.

We not only ensured that all time series start and end at the same time, but also that all time series have the same amount of values.

One time series is saved into one measurement in the TSDB, i.e., for each user group one measurement exists. Thereby, intensity values are calculated once for a particular

request log. The next time the user triggers our approach for the same request log, our approach will first check if already measurements holding the intensity values exist. If so, our approach will skip the intensity calculation part. In the following, we explain how the past intensity is forecasted without context.

4.5.2. Forecasting without Context

A user of our approach not necessarily has to provide a context in order to forecast the future workload. It is also possible to trigger our approach without providing a context. In this case, the user only has to pass request logs to our approach (and some required forecast information), but no context description. Internally, we then do not pass any regressors to time series forecasters. We make use of time series forecasters Telescope [ZBH+17] and Prophet [TL18]. Before we can forecast intensities, we need some information from the user. The information that is required is:

- Which time series forecaster should be used
- Until which date the forecast should last

A user has to pass the information in a YAML file to our approach. It is the YAML file where he also could pass a context description to our approach. For the calculation of the past workload intensities, a user has to pass the time unit for which intensity values will be calculated in a `forecast-options` dictionary. The user adds the further required information into this dictionary. Listing 4.3 shows an example `forecast-options` dictionary that could be passed by an user. In this example, the user wants the forecast to last until February 15th, 2019. The chosen forecaster is Prophet. The time unit for which workload intensity values should be calculated is one hour. By passing the options to our approach, we first calculate the workload intensities and save them into measurements. If this has already been done for a particular request log, we omit this step. Then, the forecast part of our approach is triggered. In contrast to the context description, `forecast-options` have to be always passed by the user in order to do the forecasting.

In the previous part, we already calculated the past workload intensity for each user group, and saved the values in several measurements. The forecasting is done step-wise for each measurement. In the following, we explain how the intensity is forecasted for one of these measurements. The forecast for the other measurements is done analogously.

We first fetch all data points from the measurement. One data point consists of a timestamp and a value. The forecasting is either done by Telescope or Prophet. As

Listing 4.3 Example forecast options

```
forecast-options:
  forecast-date: 2019/02/15 00:00:00
  forecaster: prophet
  time-unit: hour
```

described above, the user sets the tool in the passed YAML file. The intensity values are passed as follows to the tools:

- Telescope expects as input either a vector of values, a matrix with two columns, where the first column contains the timestamps and the second column contains the values, or an R time series object. We pass the intensity values as a vector to the tool, i.e., we omit the timestamps.
- Prophet expects as input a R dataframe object, where the first column contains dates and the second column contains the values. We obtain the dates by converting the timestamps into the expected format.

The second expected input by both tools is the amount of values that should be forecasted. We can calculate the amount with the help of the passed date that describes the point of time until the forecast should last. We convert this date to a timestamp. From the fetched data points we take the timestamp of the last value in the time series. We can calculate the amount of values to be forecasted by increasing the timestamp of the last value by the passed time unit until we reached the timestamp of the passed date. To do the forecasting without context, no more information is required by the tools. Using Prophet, we directly get a time series of predicted intensity values. Using Telescope, we only get a vector of predicted intensity values. As we calculated the amount of values to be forecasted, we also obtained their future timestamps, i.e., for each value in the vector we can set its corresponding timestamp in order to get the time series.

In the next part, we add regressors to each of the past workload intensities, and then trigger the forecast.

4.5.3. Context-Based Forecasting

In the previous part, we already forecasted the past intensity without context. Here, we add contexts to the forecast. In Section 4.4, we introduced our CDL in order to describe contexts that should be considered for the workload forecasting. All contextual information has to be provided by the user. This includes past as well as future values of contexts. Past context values have to be stored into measurements in the InfluxDB. Future values can be included into the forecast in two ways:

1. Future values of a context are stored in the same measurement where its past values are stored. The user then only has to pass the name of the measurement to our approach.
2. The user can pass future values of contexts in the context description.

Past and future values of contexts are used in order to build regressors, which can be considered by the time series forecasters Telescope and Prophet in their forecasts.

In the following, we first describe how a user can provide contextual data to our approach in detail. Once contextual data is available to our approach, it can be included in forecasts. We then explain how regressors are build from the contextual data, and then pass the regressors along with the workload intensity to the time series forecasters.

Providing Contextual Data

Our approach provides an API endpoint that can be used to provide contextual data to our approach. We use JSON [19a] as exchange format. Example (shortened) JSONs containing contextual data are shown in Listing 4.4. The user determines by himself in which measurement context values will be stored. For this purpose, in one JSON, the user passes one or more context values with corresponding unix timestamps, along with the name of the measurement into which these data should be stored, to our approach. In the examples, for the `temperature` measurement the user passes only one value, and for the `marketing campaigns` measurement two values with corresponding unix timestamps. The resolution of the timestamps is milliseconds. A timestamp can describe a date in the past as well as a future date, i.e., the user is able to pass past and future values to our approach. Furthermore, it has to describe a full time unit, that is either full second, minute, hour or day. All timestamps passed in one JSON have to describe the same full time unit. This design decision later ensures equidistance between regressor values. Furthermore, regressor values then can be easily assigned to intensity values. Gaps between passed values in the JSON are allowed, i.e. a user has not to ensure that for every full time unit in a time series a context value has to exist. The data type of the values can either be numerical or String. It is not possible to mix data types in one JSON. This corresponds to our CDL, where a user can pass events (Strings) and measurements (numerical values) in a context description to our approach. One context parameter passed in a context description at minimum has to contain the name of a measurement, where our approach can find either events or measurements stored in that measurement. If a JSON input is valid (i.e., conforms to the above described design decisions), our approach will process it. This is done in the following two steps:

1. Check if the measurement in the JSON already exists. If it does not exist, create it. Otherwise access the already existing one.

Listing 4.4 Example JSONs containing contextual data

```
{"measurement":"temperature", "values":[{"timestamp": 1548975600000, "value": 35.0}]}

{"measurement":"marketing campaigns", "values":[{"timestamp": 1542927600000, "value":
    "Black Friday"}, {"timestamp": 1543186800000, "value": "Cyber Monday"}]}
```

2. Store the passed data into the measurement.

A measurement only has two columns `time` and `value`, where timestamps and values of contexts are stored into. It is the user's decision into which measurements contextual data is distributed. However, as already described in Section 4.4.3, we recommend the user to store belonging numerical values (e.g., temperature values) and events from one category respectively into one measurement. Such categories can be, e.g., marketing campaigns, holidays, etc. Then, when the user selects a measurement in the context description, all of its contained values are considered for the workload forecasting by building one or more regressors out of them. In the next paragraph, we build the regressors.

Building Regressors

A regressor can be an additional time series to the workload intensity or a binary indicator and can be considered by time series forecasters Telescope and Prophet for forecasts. It consists of values with timestamps. We build regressors from contextual data contained in measurements. By passing context parameters in a context description to our approach, the user decides about the measurements that should be considered by our approach. A measurement either contains values with String data type, or numerical data type. Strings are not understood by Telescope and Prophet. They cannot include String values in their prediction models. Their models are either based on additive models or multiplicative models, and components of the model are expressed through numerical data. Hence, Strings have to be transformed into numerical data. Since Strings express events, and events either occur or not occur, we decide to use a binary indicator for events. The binary indicator is defined as follows:

- **0** - no event occurrence
- **1** - event occurrence

For each different String in a measurement one regressor will be build, since the forecaster should learn different impacts of different events. From numerical data contained in a measurement only one regressor will be build. In contrast to events, numerical data can be inserted directly to a regressor, resulting in an additional time series to the workload intensity.

In order to add a regressor to the workload intensity for the forecast, Telescope and Prophet expect a corresponding regressor value for each intensity value. This applies to the calculated past intensity values as well as to the intensity values that will be forecasted. Timestamps of the future intensity values are known (see Section 4.5.2). A regressor value is build from one context value. The timestamp of the regressor value is set to the timestamp of the context value. We assign regressor values based on timestamps to intensity values. A regressor value is assigned to an intensity value when their timestamps match. Hence, we require a uniform time unit representation of the timestamps of the regressor and intensity values. As already explained in the previous part, timestamps of context values in measurements describe a full time unit. The time unit has to be the same as the time unit for which intensity values were calculated, otherwise regressor values will not be assigned correctly. The user is responsible that the time units conform. For example, when the timestamps of intensity values describe full hours, timestamps of the provided context values should also describe full hours.

From contextual data contained in a particular measurement, a regressor is build as follows:

- When the measurement contains numerical data, one regressor is built. Values and their timestamps are directly transferred to the regressor. When for a particular intensity value with timestamp t no numerical value with timestamp t in the measurement is found, we insert a 0 with timestamp t to the regressor.
- When the measurement contains Strings, one regressor from each different String is build. Same Strings are considered for one particular regressor. They are converted into a 1 and then are inserted with their timestamps into the regressor. Afterwards, when for a particular intensity value with timestamp t no regressor value with timestamp t exists in the regressor, we insert a 0 with timestamp t to the regressor.

The start time of the time range of the context values inside a measurement can be smaller than the start time of the past workload intensities. In this case, only the contextual data starting from the start time of the past workload intensities is considered for the forecast. Analogous, the end time of the time range of the context values inside a measurement can be greater than the provided forecast date by the user. In this case, only the contextual data until the passed forecast date is considered.

A regressor contains past as well as future regressor values. When the user did pass future context values with a context parameter, they are converted into regressor values, and override the future regressor values with the same timestamps already contained in the regressor.

In the next part, we perform an example forecast with regressors.

Forecasting with Regressors

Figure 4.16 shows an example forecast with regressors. In the example, we use Prophet as the forecaster. The future workload intensity is forecasted based on a context description and on a past workload intensity. Timestamps describe full days and are represented as dates, e.g., 01/02/2018. Past dates are the dates of 2018, and future dates are the dates of 2019. The passed context description contains two context parameters. They tell our approach that contextual data should be included from the `temperature` and the `marketing campaigns` measurement. For `marketing campaigns`, the user also passed two future values with the context parameter. `Sale 30%` occurs at 01/02/2019, and `Sale 40%` occurs at 04/02/2019. The workload forecasting approach then queries a past workload intensity and the contextual data from the InfluxDB. As it can be seen, the `temperature` measurement contains future temperature values, and the `marketing campaigns` measurement not. However, the user passed future values of marketing campaigns with the context parameter. From the context values, regressors are built. They are assigned to the past workload intensity, which is denoted with `Y`. The dates of `DS` correspond to the date representation accepted by Prophet. For example, 2018-02-01 is the same date as 01/02/2018. For the dates of 2019, `Y` has no values, as these values will be forecasted. However, the regressors contain future values in order to provide the forecaster the future context. The values will impact the future intensity values. The regressors contain a 0 value when no context value is available at the date. For example, the `temperature` regressor has a zero value at 2018-02-03, as there is no context value in the `temperature` measurement at 03/02/2018. The `Sale 30%` regressor contains a zero value at 2019-02-02, as there will be no `Sale 30%` event. It contains a 1 at 2019-02-01, as the user passed in the context parameter that at 01/02/2019 a `Sale 30%` event will take place. The past intensity and assigned regressors are the input to Prophet, which forecasts the future workload intensity. The column `YHAT` contains the forecasted values. In the example, we forecasted the future workload intensity of one particular user group. The process is repeated for the other user groups by querying the other workload intensities from the InfluxDB and assigning them the regressors. Using Telescope as the forecaster, the process is the same, except that we do not pass the timestamps of the values (intensity and regressor values) to it.

The result of the context-based forecasting are a time series of predicted values for each user group. As described in the next section, we can update the Behavior Mix based on the time series. This will describe the future user behavior. Besides, we process and aggregate the time series and output load tests.

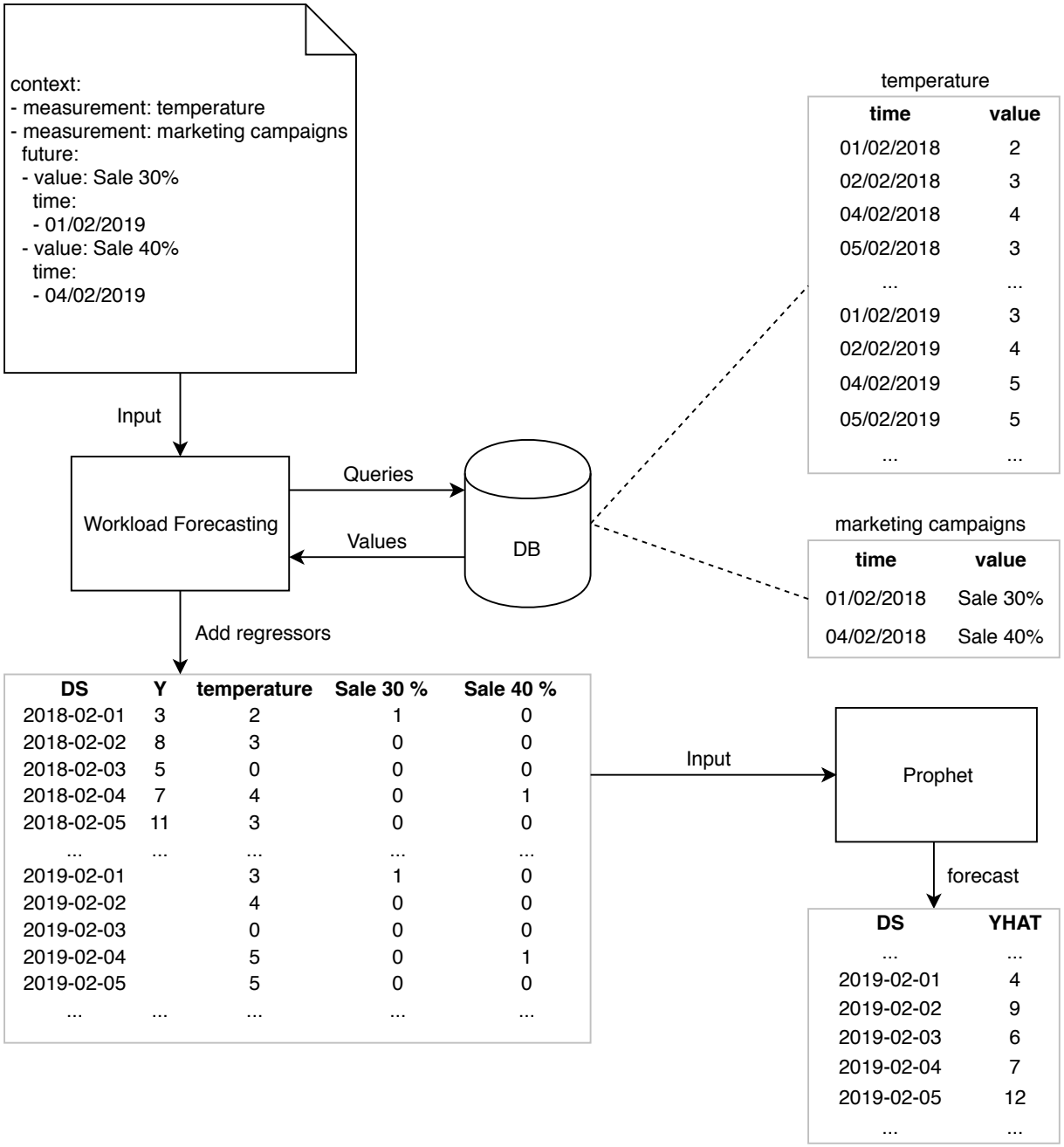

Figure 4.16.: Forecasting with regressors

4.6. Workload Processing

In the previous section, we forecasted the future workload intensity of each different user group identified by WESSBAS from a session log. Each of these user groups is described through a behavior model, which specifies how a particular user belonging to the corresponding user group navigates through the application. In this section, we will process the forecasted intensities and return one or more load tests to the user, where each load test simulates a load scenario. An example load scenario would be high workload, and a load test simulating this scenario would generate a high workload on the SUT. A user of our approach decides, for which load scenarios he wants to test. The load scenarios our approach covers are presented in Section 4.6.1. Our approach then processes the forecasted intensities, and extracts for each requested load scenario one load test from the forecasted workload. How this is done is explained in Section 4.6.2.

4.6.1. Covered Load Scenarios

We derive load scenarios users could want to test for mainly from application scaling scenarios. The scenarios are also relevant for applications that cannot scale. In the following, we call applications that can scale “scalable application” and applications that cannot scale “static application”.

We first start with load scenarios where the application experiences small, medium and high workload. In order to ensure that an application can perform its basic functionalities, we first could apply small workload on the system. The next step would be to test the medium workload on the system. Here, we would test whether the system functions during expected load. High workload will show whether the application crashes under more extreme situations. For scalable applications, these load scenarios will also show if capacities are always utilized, regardless of the current load on the system. Our approach covers all three load scenarios.

The next covered load scenarios comprise sharp increases (spikes) and sharp decreases in workload. A sharp increase in workload tests whether a scalable application can scale up fast enough in order to handle the situation. Scaling up fast can ensure that the application still functions during unexpected high increase of workload, and that the application does not crash. Analogously, a sharp decrease in workload tests whether a scalable application scales down fast. Scaling down fast ensures that resources are released fast once they are not needed anymore. Allocating resources from cloud providers often costs money as long as you use them, and, hence, the faster resources are released, the more money is saved. Sharp increase and decrease in workload can also

be applied on static applications to test if the application can handle these situations, i.e., if the application functions during these situations.

The last load scenario we cover is to test for all possible workloads. For this load scenario, we will output a load test that simulates the entire forecasted workload. This tests whether the application can cope with every workload situation that will occur in the future, but will also result in huge testing effort.

The user is able to output load tests simulating the above described scenarios. For future work, we suggest to add even more load scenarios to our approach. Two possible load scenarios are listed in Section 7.3. In the next part, we describe how we extract load tests from the forecasted workload.

4.6.2. Load Test Extraction

A user of our approach decides for which load scenarios he wants to test. As described in Section 4.5.2, a user has to pass a `forecast-options` dictionary in a YAML file to our approach. It includes information that is required in order to do the workload forecasting. We extend this dictionary through a list element `scenarios`, that is optional. When the user omits it, we will return one load test that tests for the maximum workload by default. The user can list one or more load scenarios in `scenarios`. An example `forecast-options` dictionary including `scenarios` is shown in Listing 4.5. The user wants to test for three scenarios, that are high and medium workloads, and a sharp increase in workload.

A load scenario can be described through one or more intensity values describing the amount of concurrent users, and the behavior of the users. To obtain the intensity values, we proceed as follows. The forecasted workload intensities are a time series of predicted intensity values. We aggregate these time series to exactly one time series. For this purpose, we sum up the intensity values of all time series at times t_1, ..., t_n, where t_1 is the timestamp of the first intensity values, and t_n is the timestamp of the last intensity values, and obtain as result the aggregated time series, which we call in the following the aggregated workload intensity. The values of the aggregated workload intensity are called aggregated intensity values. For each listed load scenario in `scenarios`, we will process the aggregated workload intensity and extract one or more values, which we set as the load intensity of the load test that will be returned to the user. The user behavior was already calculated by WESSBAS in form of the behavior models. Each of the forecasted workload intensities belongs to one user group, and hence belongs to one behavior model. We will not change the calculated behavior models, but will update the behavior mix. The behavior mix contains the occurrence probability of each behavior model during workload generation. To update the behavior mix, we will first extract a

Listing 4.5 Example forecast options

```
forecast-options:
  forecast-date: 2019/02/15 00:00:00
  forecaster: prophet
  time-unit: hour
  scenarios:
  - high
  - medium
  - sharp increase
```

single intensity value from each forecasted workload intensity. Each of these extracted values belongs to exactly one behavior model. The occurrence probability of a particular behavior model results from the division of its belonging intensity value by the sum of all extracted intensity values. For each load scenario a new behavior mix is calculated, and then included in the load test that simulates the scenario. Each load test then includes one or more intensity values, a behavior mix, and the previously calculated behavior models. For each simulated user, a behavior model with the probability specified in the behavior mix is chosen, and the user navigates through the application as specified in the behavior model.

In the following, we explain for each presented load scenario in Section 4.6.1, how we extract the load test that simulates the load scenario from the forecasted intensities. For this purpose, we explain how we extract the workload intensity for a load test and how we update the behavior mix it will include. We use the following structure to explain how load tests are extracted for each load scenario:

- **User input:** The term a user has to pass in the `scenarios` list in order to get the load test that simulates the desired load scenario
- **Intensity:** How the load intensity for the load test is extracted from the aggregated workload intensity
- **Mix:** How the occurrence probabilities in the Behavior Mix are calculated

We start with the “entire workload” scenario.

Entire Workload

- **User input:** all
- **Intensity:** We do not process the aggregated workload intensity. The load intensity of the load test is set to the aggregated workload intensity.

- **Mix:** To update the occurrence probabilities of each behavior model in the behavior mix, we calculate the average intensity value of each forecasted time series. Afterwards, we sum up the average intensity values. To update the occurrence probability of a particular behavior model, we divide its belonging average intensity value by the sum of all average values.

High Workload

- **User input:** high
- **Intensity:** From the aggregated workload intensity, the maximum value is extracted.
- **Mix:** The maximum value extracted from the aggregated workload intensity is the sum of the intensity values from the forecasted intensities at a time t. We extract these intensity values from the forecasted intensities. The behavior mix is updated by dividing for each behavior model its belonging intensity value by the maximum value.

Medium Workload

- **User input:** medium
- **Intensity:** From the aggregated workload intensity, we calculate the average intensity value. For this purpose, all aggregated intensity values are summed up, and the result is divided by the total number of intensity values.
- **Mix:** We calculate the average intensity value from each forecasted time series. Afterwards, we sum up the average intensity values. To update the occurrence probability of a particular behavior model, we divide its belonging average intensity value by the sum of all average values.

Small Workload

- **User input:** small
- **Intensity:** From the aggregated workload intensity, we extract the smallest value, that is not zero. This is to ensure that the resulting load test does simulate workload on the SUT.

- **Mix:** The minimum value extracted from the aggregated workload intensity is the sum of the intensity values from the forecasted intensities at a time t. We extract these intensity values from the forecasted intensities. The behavior mix is updated by dividing for each behavior model its belonging intensity value by the minimum value.

Sharp Increase in Workload

- **User input:** sharp increase
- **Intensity:** We perform the following two steps to find the sharpest increase in the aggregated workload intensity:
  1. Iterate over all aggregated intensity values and select all pairs with two consecutive values, where the second value is at least twice as high as the first value, i.e., the increase is minimum 100 percent.
  2. From the selected pairs select the pair with the highest second value. If two or more pairs include this value, select the one where the percentage increase between first and second value is higher. We call the selected pair the sharpest increase in the intensity.

  The outcome is a load test that simulates consecutively two different intensity values, e.g., in the first hour 300 users, and in the second hour 900 users.
- **Mix:** The sharpest increase includes two values at times t_i and t_{i+1}. To update the occurrence probabilities of the behavior models, we calculate from each forecasted intensity the average value from its values at times t_i and t_{i+1}. We sum all calculated averages up. The occurrence probability of a behavior model is its belonging average value divided by the sum of the average values.

Sharp Decrease in Workload

- **User input:** sharp decrease
- **Intensity:** We perform the following two steps to find the sharpest decrease in the aggregated workload intensity:
  1. Iterate over all aggregated intensity values and select all pairs with two consecutive values, where the first value is at least twice as high as the second value, i.e., the decrease is minimum 50 percent.

2. From the selected pairs select the pair with the highest first value. If two or more pairs include this value, select the one where the percentage decrease between first and second value is higher. We call the selected pair the sharpest decrease in the intensity.

- **Mix:** Analogous to the sharp increase in workload load scenario.

We use WESSBAS as the load test generator to automatically generate the load tests. For each load scenario, we pass the extracted workload intensity and the updated Behavior Mix to the WESSBAS workload model generator. The behavior models were already calculated and are available to WESSBAS. It first generates a workload model instance out of the data. Afterwards, the instance is passed to the WESSBAS test plan generator, which converts the instance to a load test. The resulting load tests are returned to the user. The question that remains is how long the load tests have to be executed. For the load tests that simulate one intensity value (maximum, average, and minimum), the user has to decide how long the load test should run. However, approaches exist, which recommend when to stop the load test [ASSH16]. To test the entire workload, the load test should run as long as the time range of the forecast, e.g., when the forecast was made for the next week, the load test would have to run for one week, consecutively simulating the intensity values. Therefore, the test execution time can be extremely high, but every occurring load scenario is covered. Load tests simulating sharp increase and decrease simulate two consecutive intensity values. When they were calculated for an hourly time interval, the load test could be executed for two hours, in the first hour simulating the first intensity value and then increasing the intensity in the second hour to the second intensity value.

Chapter 5

Implementation

In this chapter, we provide implementation details of our approach. As already explained in the introduction of this thesis (Chapter 1), our approach extends the ContinuITy project. This is done by embedding our approach into it. All parts of our approach are then executed automatically. The extended ContinuITy approach is presented in Section 5.1. In Section 5.2, we explain how the forecast process works in ContinuITy. Finally, in Section 5.3, we summarize the technologies used to implement our approach.

Our code is available on GitHub [CP19].

5.1. ContinuITy Extension

ContinuITy consists of a microservice architecture, which is depicted in Figure 5.1. The workflow is controlled by the Orchestrator. It receives an order at /order/submit and then calls the other services via RabbitMQ [Piv19] exchanges in order to produce required results. For example, the Session Logs microservice produces session logs. When a microservice has to access a created artifact of another microservice, it can call a link passed by the Orchestrator to obtain the artifact. For example, the Workload Model microservice, which generates a workload model from a session log, obtains a produced session log by calling the link it received from the Orchestrator [CP19].

To embed our approach into ContinuITy, we extend the ContinuITy architecture by a new microservice, that contains the workload forecast logic. The extended architecture is shown in Figure 5.2. The new Forecast microservice shares results with the Workload Model microservice, as WESSBAS is included in it. WESSBAS is used to calculate the behavior models. The Forecast microservice receives a link from the Orchestrator to obtain the session log entries for each Behavior Model, and forecasts the future workload intensities, processes them, and updates the Behavior Mix. Afterwards, the Orchestrator

provides the Workload Model service a link to obtain the produced results from the Forecast microservice. From the updated behavior mix, the processed future intensity, and the previously calculated behavior models WESSBAS then generates a load test. A more detailed overview of the forecast process is presented in Section 5.2.

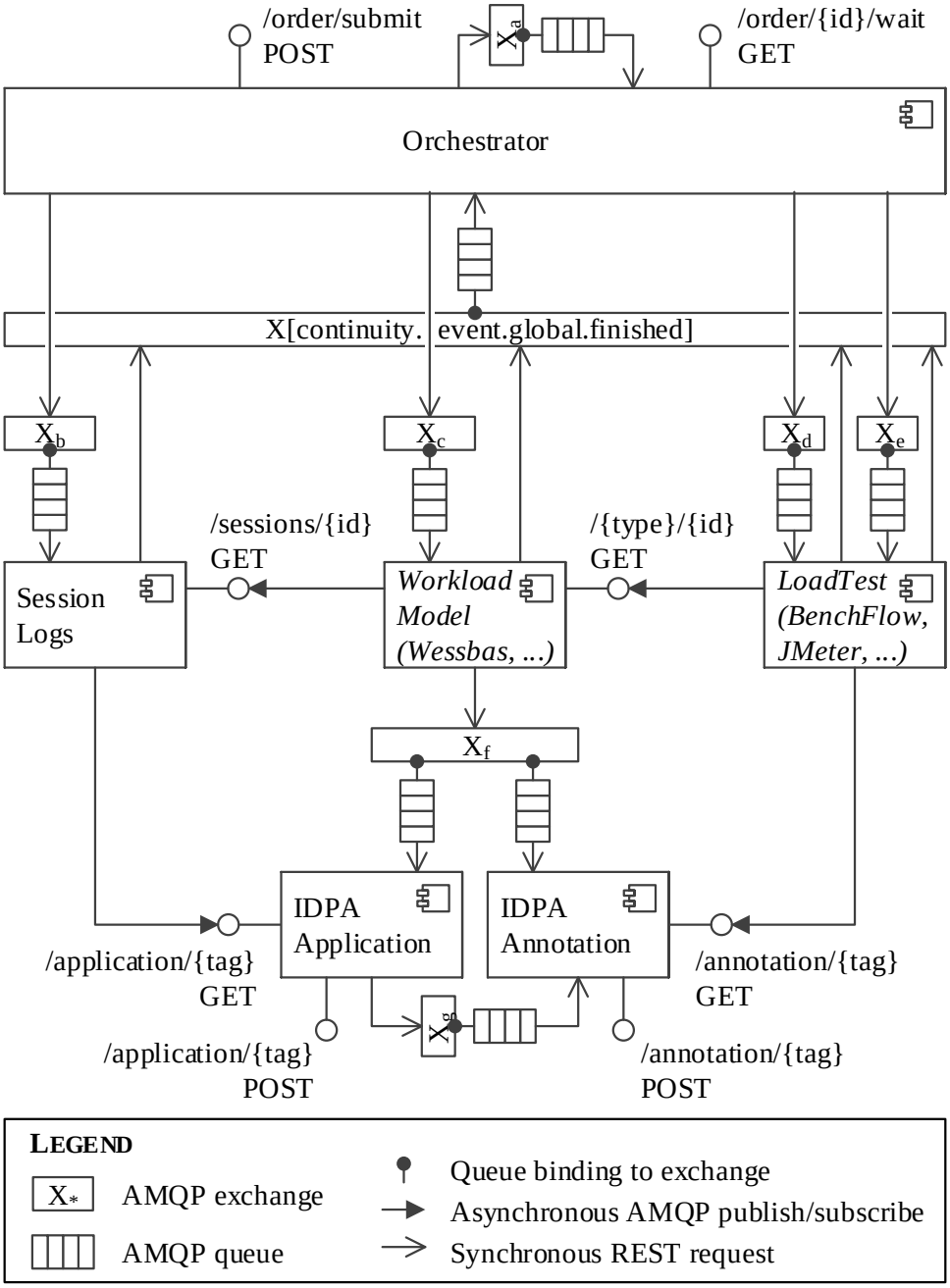

Figure 5.1.: Previous architecture of ContinuITy [CP19]

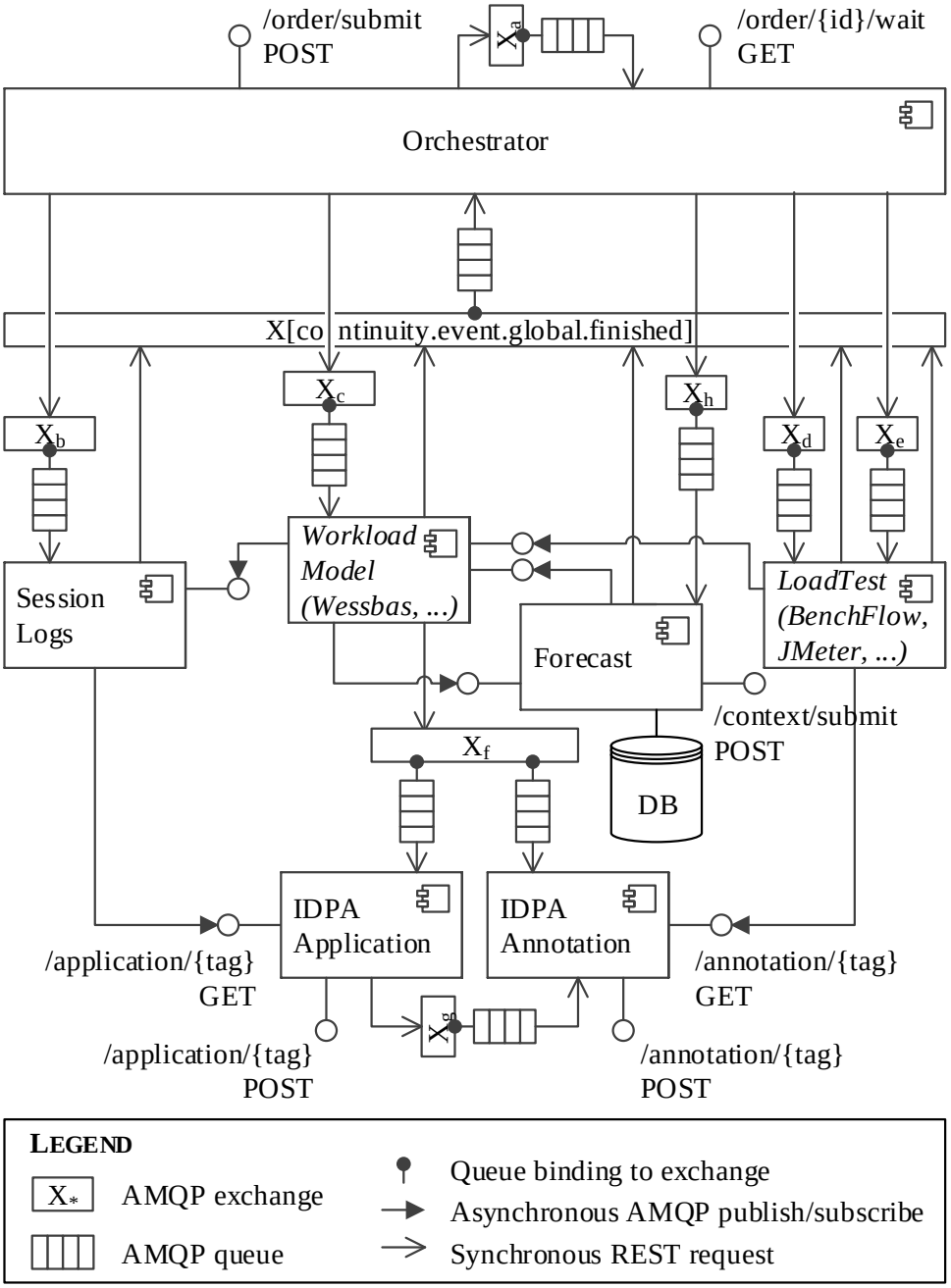

Figure 5.2.: Updated architecture of ContinuITy (based on [CP19])

For our approach, we mainly extended the Session Logs and Workload Model microservice, and have implemented the Forecast microservice. In the following, we summarize for each of these three microservices the innovations.

Extension of the Session Logs Microservice

Before this thesis, the Session Logs microservice could only extract session logs from invocation sequences [NTC19] and OPEN.xtraces [OHH+16], i.e., session logs could not be produced from recorded request logs. We extend the microservice, so that session logs can be generated from request logs in CSV format. For this purpose, we embed our session log generator into the service, that was presented in Section 4.3.

Extension of the Workload Model Microservice

The WESSBAS behavior model extractor component is already capable of generating behavior models from a session log by clustering session entries. As explained in Section 4.5, we extend the clustering of the session entries to retain the information of which session entry was assigned to which cluster. For each behavior model, we then have a list of session entries. We extend the microservice by an endpoint, that the Forecast microservice can use to obtain the lists of session entries. We extend the microservice further in order to access the produced results from the Forecast microservice, that are the forecasted workload intensity and the updated Behavior Mix. The artifacts are used to create a WESSBAS model instance with the WESSBAS workload model generator component. The model instance is then converted into a load test with the WESSBAS test plan generator. More details on the load test generation can be found in Section 4.6.2.

Forecast Microservice

The implementation of this microservice was the main part of our approach. It requests from the Workload Model microservice the session entries for each behavior model. It then calculates the past workload intensity of each user group. How this is done in detail was already explained in Section 4.5. The calculated intensities are saved into an InfluxDB, so that for the same request log intensities do not have to be calculated again. Furthermore, this microservice receives JSONs at /context/submit containing contextual data, i.e., the endpoint can be used to stream contextual data into the InfluxDB. When the user passes a context description with an order to the Orchestrator, this microservice sends queries to the InfluxDB to obtain the contextual data that should be considered for the workload forecast, and calculates regressors from the data. The regressors are assigned to the calculated workload intensities and the data is passed to either Telescope or Prophet, which forecast the future workload intensity of each user group. More details on the workload forecasting can be found in Section 4.5.3. The service processes the future workload intensities and updates the Behavior Mix. How this is done is described

in Section 4.6. The Forecast microservice provides the processed load intensity and updated Behavior Mix through an endpoint, that the Workload Model microservice can use to access these artifacts.

In the next part, we show how the forecast process works.

5.2. ContinuITy Forecast Process

After we embedded our approach into ContinuITy, two different processes for the load test generation exist. Figure 5.3 shows the processes. The process to generate a load test that does not include forecasting already existed before this thesis. The process is triggered by submitting an order to the Orchestrator. The order is based on YAML and contains information required by ContinuITy, like the workload intensity that should be considered for the load test, the duration of the load test, and especially the link where traces containing measured data from production can be found. The Orchestrator passes the link to the Session Logs microservice, which converts the data first into a session log. The session log is then processed by the WESSBAS microservice, which clusters the session entries to calculate the behavior models and the behavior mix. With the workload intensity passed by the user, WESSBAS then first generates a workload model instance, and then converts the workload model instance into a load test.

The forecast process we implemented is similar to the process that does not include forecasting, but contains more steps. Again, the user submits an order to the Orchestrator. He sets a flag in the order to tell ContinuITy that it should generate a load test from forecasted workload. The order must contain the `forecast-options` dictionary, that tells ContinuITy, for example, how long the forecast should last. If the user wants to include contexts in the forecast, the order must also contain a context description. Further required information include the duration of the load test and the link where either traces containing measured data from production can be found or recorded request logs. The workload intensity has not to be passed with the order. The Session Logs microservice converts the traces/request logs into a session log. The WESSBAS microservice takes the session log as input, and calculates the behavior models and the mix, which will be updated later. Then, the Forecast microservice performs the workload forecasting. For this purpose, it takes the lists of session entries as input. The result is the workload intensity for the load test and the new behavior mix frequencies. The WESSBAS microservice executes a second time and takes the workload intensity and the new behavior mix frequencies as input. It updates the behavior mix with the new frequencies, and generates then the workload model instance. The workload model instance is converted into a load test, which is returned to the user.

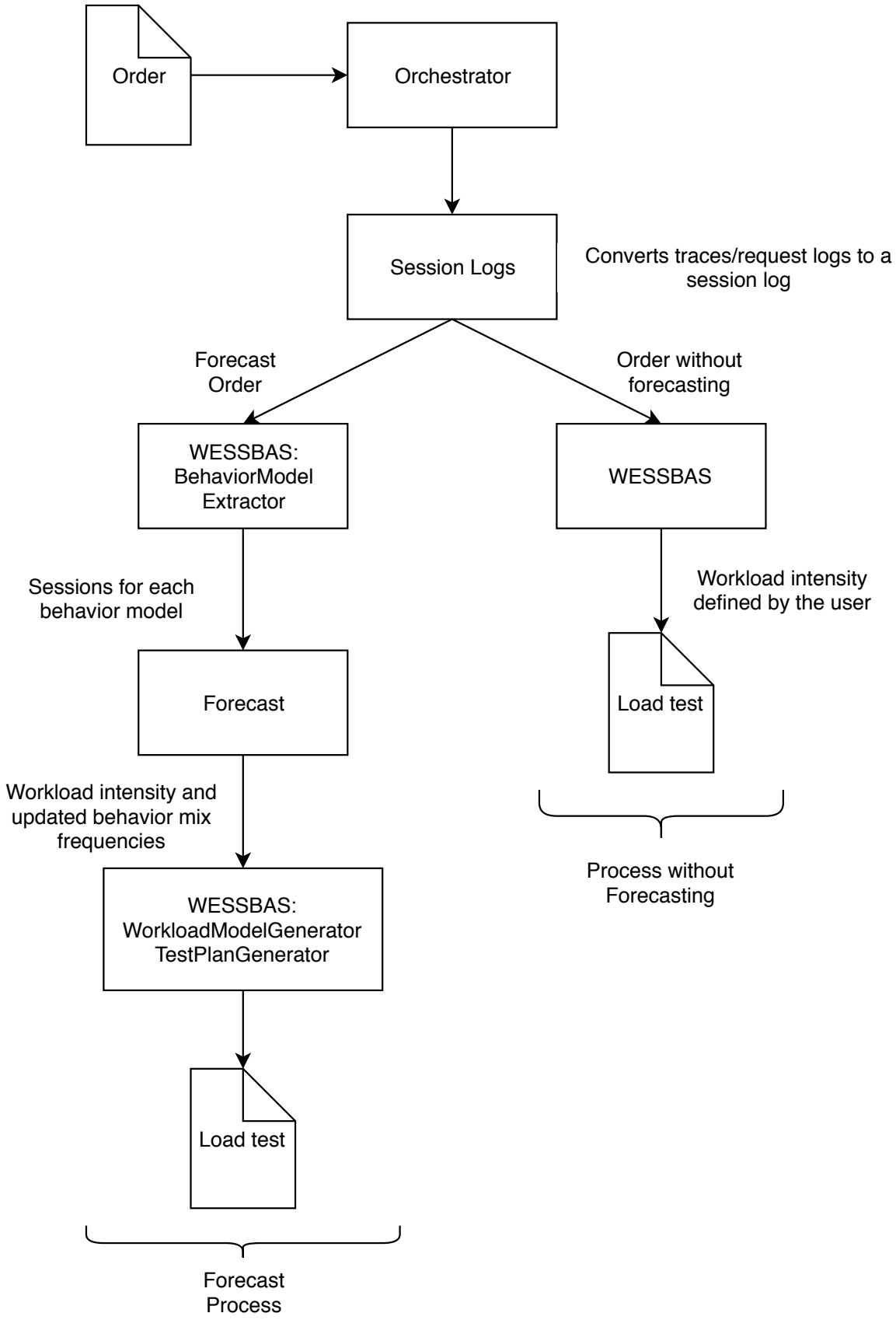

Figure 5.3.: ContinuITy processes

In the current implementation, the user of ContinuITy can only request the high workload, average workload, and the low workload scenario. The reason is that WESSBAS in its current implementation can generate a load test with only a single workload intensity value. We suggest for future work to extend WESSBAS, so that more than one intensity value can be mapped into a load test.

5.3. Used Technologies

We used mainly the Java [Ora19] programming language to implement our approach. Time series forecasters Prophet [fac17] and Telescope [Uni18] are available in R [The19b]. We use them to forecast the future workload intensities of different user groups. To be able to execute R code during the forecast process, we use the Java/R Interface (JRI) [RFo19], which allows us to embed R code in Java code. Using JRI, R code is run as a single thread. For each tool, one main R script exists that performs the time series forecasting. JRI allows us to pass Java variables to the scripts. They take as input the past workload intensities, and output a time series of forecasted values. The forecasted values are passed back to the Java code, which then processes the forecasted data.

In order to be able to process inputs by the user, we use the Jackson API [Fas18], that can be used to map, e.g., JSON and YAML data to Java objects and vice versa. We use the Jackson API to map context descriptions (YAML data) to Java objects, and to map provided contextual data in form of JSON Strings to Java objects.

To process request logs in form of CSV data, we use the Univocity CSV parser [Uni19] to parse the request logs and to generate Java objects out of the data. From each row in the CSV, one Java object is generated. We then do calculations on the Java objects to generate the session log. We use WESSBAS [VHS+18] to calculate behavior models from a session log.

Our algorithms to calculate the past workload intensity from session data (Algorithm 4.1 and Algorithm 4.2) utilize the range implementation of the Apache Commons Lang Java library [The18]. The past workload intensities are saved into a time series database (TSDB). We use InfluxDB [Inf19b] as the TSDB.

Chapter 6

Evaluation

In this chapter, we evaluate our approach. In Section 6.1, we first present our evaluation goals, and derive our research questions (RQs) from them. We perform our experiments with non-synthetic logs extracted from a real-world information system. The logs are described in Section 6.2. Afterwards, in Section 6.3, we present the setup of our experiments in order to answer the RQs. After we have run the experiments, we obtained our evaluation results. The results are presented in Section 6.4. In Section 6.5, we analyze the results and answer the RQs. Finally, we investigate whether our evaluation is valid. Threats to validity comprise internal, external, construct, and conclusion threats, which are discussed in Section 6.6.

6.1. Evaluation Goals

Our approach consists of two main parts. The first part is the workload forecasting based on a context, and the second part the processing of the forecasted workload to extract load tests. Our evaluation will cover both parts. We investigate the following:

- is WESSBAS appropriate for our approach to calculate the historical workload
- are time series forecasters Telescope and Prophet suitable for our approach to predict the future workload intensity for several user groups
- is the forecasted workload accurate and can contexts increase the accuracy
- do load tests returned by our approach cover requested load scenarios that occur in the future, and if only the required testing effort (test execution time and resource usage) is spent when executing the test

- do contexts help to decrease the testing effort and increase the coverage of requested load scenarios that will occur in the future

We can derive the RQs from the evaluation goals. The first RQ is:

RQ1: How accurate is the forecasted workload?

By answering this RQ, we can find out whether our overall approach is meaningful. Our approach strongly depends on existing tools. We use WESSBAS for the extraction of different user groups from request logs and time series forecasters to predict their future load intensities. The workload will only be accurate if WESSBAS and the time series forecasters function as expected. They are appropriate for our approach, when the forecasted workload is accurate. For this purpose, first the user groups have to be identified correctly and second their future load intensities have to be forecasted accurately. Prophet and Telescope are also able to include regressors in their prediction models. We calculate regressors from contexts. With the regressors, we hope to gain even higher accuracy. The second RQ is:

RQ2: What is the impact of contexts on the accuracy of the forecasted workload?

RQ1 and RQ2 target the first main part of our approach. The next RQs target the second main part, the workload processing.

The third RQ is:

RQ3: How efficient is our approach in covering requested load scenarios that will occur in the time range we test for?

This RQ considers the fourth evaluation goal. We want to cover all load scenarios a user could request and that will occur in the future, and at the same time we want to spend only the testing effort that is really required to simulate a particular load scenario. For example, when the maximum intensity that will occur in the future is 10000, then a returned load test that tests for 20000 users covers the maximum intensity that occurs in the future, but the testing effort is too high, as 10000 users more than required are simulated with the load test. Analogous to RQ2, we can test if providing a context can help decrease the testing effort and/or to cover requested load scenarios that occur in the future. The fourth RQ is:

RQ4: What is the impact of contexts on the coverage and on the efficiency?

For each RQ, we design one experiment to answer it. The setup of these experiments is described in Section 6.3.

6.2. Input Data for the Evaluation

Our experiment is executed with non-synthetic logs from a real-world system. The usage of synthetic logs would result in external threats to validity, since they could not contain representative information of occurred workload during production.

For our evaluation, we use real-world logs [Hid19] extracted from the Student Information System (SIS) [Cha19b] of the Charles University in Prague [Cha19a]. Users of the system not only include normal students, but also, e.g., Ph.D. students and professors. It is possible to look up exam dates, register for courses, search for persons at the university, add teaching material, etc. The logs were recorded by an Apache web server [The19a] over a six month time period. All log entries include the IP address of the machine from which the requests came from. However, not all log entries contain session identifiers. Session identifiers are only available for those records where the user was logged in. Users that are not logged in are able to access only some of the SIS pages. However, in order to build a session log out of the raw logs, all log entries have to contain a session identifier (otherwise, log entries without session identifiers would not be considered). We group log entries without session identifiers and assign log entries in one group the same session identifier. The following conditions apply for log entries in the same group:

- The log entries have the same IP address.
- The time difference between two subsequent log entries is a maximum of 30 minutes. We use the 30 minutes as timeout value between two requests [FGL15] in order to calculate the session identifiers.

We process log entries that already contained session identifiers the similar way, since we have identified a considerable amount of recorded sessions in the logs that last for several hours and where the user is inactive most of the time. Hence, we apply here also the 30 minutes timeout threshold and assign new session identifiers to these log entries if required. We do this the following way. If between two subsequent log entries a and b with same session identifiers the time difference is higher than 30 minutes, then, starting from b, all following log entries with the same session identifier will be considered as records of a separate session.

While we grouped log entries with the same session identifiers, we identified sessions with a huge amount of requests to the same endpoint, and lasting for several hours, or even days. We assume the system either performing batch jobs in this case, or the user making use of an API to update permanently, e.g., his calendar with exam dates. We omit these sessions, because they describe a constant load on the system.

We can reduce the amount of data to be processed further. We remove all records from the logs where static content was loaded, as the system performs no calculations when providing this content. This comprises all log entries where files with the following format were loaded: js, png, css, gif, jpg, ttf, ico, woff, svg, txt, and pdf. After we removed the static content, we still have several hundreds of SIS pages left that could have been recorded. The user follows navigation links in order to perform the desired task, and, thereby, moves from one page to another. Typical URLs of the website are, e.g., "/studium/login.php", "/studium/eng/index.php", and "/studium/predmety/index.php". (Almost all) URLs have the following structure:

"studium/eng/category/name_of_page"

where "eng" and "category" are not always used. "eng" is used, when the user sets the language of the application to English. We delete "eng" from all URLs, where it occurs. Category is used, when the user accesses a so called "module" of the application, that can be seen as a particular service, where the user can, e.g., look up information or download data. When the user is situated at the main page "studium" and did not access a module, then category is not used. Beside modules, further URLs can be found with the same structure. However, they do not describe services users can access, but are called by the system itself to, e.g., reset the duration of a session. Such URLs are called by the system when the user accesses a service, and, thus, go along with other services. They also extend the duration of sessions to hours, even when the user is not active. Categories in the affected URLs are, e.g., "stev" and "lib_class". Further URLs that do not describe any service contain, e.g., "lib", "lib_tiny" or "res" as category. As we investigated these URLs, we found out that these are mostly static content. In a few cases, the system executes a php script. We decide to discard URLs where the category does not describe a service, as otherwise sessions would last too long and requests would be included that are not part of the navigational pattern of the user. We group the remaining URLs into the categories. When no category is available, the user is situated at the main page, and we use "studium" as the category.

Overall, we identified 64 categories (services). WESSBAS clusters session entries in a session log by first converting each into a behavior model. Each service would correspond to one state in the behavior models. The number of features used for the clustering results from the amount of possible state transitions, which would be approximately 4000 in our case. For the clustering, all features of the behavior models are compared against each other, even state transitions with zero probability. Besides, a huge amount of session entries has to be processed. This results in memory problems when clustering the behavior models, even with a machine with over 100 GB of RAM. Calculations take extremely long time and the machine runs out of memory, when there are too many features. Hence, we have to decrease the amount of considered categories. We cannot

summarize categories, because we do not have enough knowledge about the endpoints. Therefore, we decide to omit some of the categories. Our requirement was to not run out of memory and that the calculation performs of maximum one day. We tested configurations and could perform the clustering on 3 weeks of data with a size of 1.5 GB, and 50 categories. The RAM utilization of the machine we use for our experiment was 98 % at the end of the calculation. The used machine has 128 GB RAM. The calculation took approximately eight hours. To not run out of memory, we cut out smaller logs with a maximum of approximately 1.5 GB of data from the six month SIS logs, and run our experiments on the smaller logs. Furthermore, we reduce the amount of categories to 50. For this purpose, we first count the amount of requests made to each service over the six month time period, and investigate, which of these services are requested the least (negligible workload). The least requested services (categories) are omitted, until 50 are left. The considered 50 categories are listed in Table 6.1 and Table 6.2, whereas the omitted categories can be found in the appendix in Table A.1. The categories are ordered based on the request counts, so that services with higher request counts are listed first. For each category we list the service that is accessed and the role we assume a user at least has to have to access the service. Not all services are accessible by normal students. Some require, e.g., admin permissions. When the role is “student”, we could access the page by ourselves. Otherwise, we see the title of the page and get a notification that the module cannot be accessed. Hence, for the other roles, we cannot guarantee, that our assumptions are correct. These are only guesses. In cases we could not infer the role at all we left the cell free. We hope WESSBAS can identify user groups with different roles with its clustering, since users with different roles perform different tasks in the system.

We pass chunks of the six-month log to our session log generator, which we extended for our evaluation through the rules described above. The logs do not include request end times, i.e., the request end time is set to the request start time. The output are session logs which can be processed by WESSBAS.

Category	Service	Role
studium	Main page	Student
predmety	Subjects	Student
rozvrhng	Schedule NG	Student
predm_st2	Subjects and schedule registration	Student
term_st2	Exam dates	Student
prijimacky	Admission	Student
dipl_st	Thesis (subject selection)	Student
zkous_st	Summary of exam results	Student
kdojekdo	Search for persons	Student
omne	Personal data	Student
zkous_uc2	Exam results	Student
phdisp	Individual degree program for Ph.D. students	Ph.D. student
soub_mana	File Manager	Student
dipl_uc	Student thesis	
nastenka	Notice-board	Student
ciselniky	Classifiers	Student
anketa	inquiry	Student
term_uc2	Exam dates - publishing	Professor/ assistant
szz_st	Final exams	Student
predm_uc	Students registration into subjects	Professor/ assistant
ekczv	Life-long education programs	Student
cml	Publications	
szz_uc	Protocols of final state exams	
komise	Committees	Student
grupicek	Study Group Roster	Student
szz	Invitations to state exams	Student
esc	Official journey registration	Professor/ assistant
podprij	Admission requirements	
wstip_st	scholarships	Student
akreditace_rvh	Acreditation RVH NAU	
promoce	Graduation	Student
wstip_uc	Scholarships	
skolitel	List of advisors	Student
grupik	- This service does not load -	
role	Selection of role	Admin
harmonogram	Harmonogram	Student
rozpis	Curriculum	Professor/ assistant

Table 6.1.: Considered categories (1)

Category	Service	Role
prezkumy_st	Study charges and petitions	
deda_zahost	Admission of a foreign guest	
vyspl	Listing of responsibilities	
sestavy	Learning sets	Professor/ assistant
diplmat	Diploma-matrix	
uchak	Candidate Commission	
ave	SIS Administration	Admin
deda_amu	Organisation of events with international participation	
staze_uc	Internships	
ave_uziv	SIS Administration: Roles and users	Admin
deda_strav	Meals	
transcript	Transcript	
bookmarks	Bookmarks	Student

Table 6.2.: Considered categories (2)

6.3. Setup of Experiments

Our approach is embedded into ContinuITy, which automatically can execute all the steps of our approach. However, in order to investigate intermediate results produced from individual parts of our approach more easily, we decide to perform the steps manually. We summarize the steps in the following and list the intermediate results obtained after each step:

1. **Step:** Convert request logs into a session log
   Result: Session log
2. **Step:** Input the session log to the WESSBAS behavior model extractor
   Result: behavior models
3. **Step:** Calculate the past workload intensity of each user group
   Result: Past workload intensities
4. **Step:** Calculate regressors from a context description
   Result: Regressors
5. **Step:** Input the regressors and the workload intensities to a time series forecaster
   Result: Forecasted workload intensities

6. **Step:** Processing the forecasted intensities
   Result: Load test(s)

Step 4 is omitted when the user did not pass a context description. We run four experiments in total. For a particular experiment, we do not have to perform all of the steps mentioned above. In the following sections, we describe for each experiment, which steps we perform, and which RQs we aim to answer with the experiment.

Hardware Infrastructure: Our experiment is run on a virtual machine with a CPU that has eight cores. The clock speed is 2.20 GHz. The machine has 128 GB RAM and the installed operating system on the machine is Microsoft Windows Server 2016 [Mic19].

6.3.1. Experiment 1: Simulation of the Forecasted Workload

With this experiment, we aim to answer RQ1. For this purpose, we perform steps 1, 2, 3, and 5. This means, we do not consider contexts (step 4) and will not process the forecasted intensities (step 6), i.e., we will replay the forecasted workload as it is. This corresponds to the "entire workload" load scenario described in Section 4.6, but here we do not need to calculate a behavior mix, as we do not generate a real load test with WESSBAS. We will simulate each forecasted intensity, and simulated users of a particular intensity always follow the behavior model describing the user group for which the intensity was forecasted. In order to be able to compare the forecasted workload against the real workload the system under test (SUT) will experience, we consider one part of a given request log as the already recorded log and the other part as the log that will be recorded in the future. For this purpose, we take one month of the SIS logs, and consider the first three weeks as the past log, and the last week as the future log. For the past log we perform the steps and then obtain the future intensity of each user group. The time interval for which intensities are calculated is set to one hour. The forecast range is one week, and, hence, the duration of a load test simulating the forecasted workload would be one week, too. Due to time restrictions, we are not able to execute a load test for one week. Thus, we decide to implement a load test simulator that writes sequences of requests users perform into a file. It simulates each user group by simulating its forecasted intensity, e.g., in the first hour 100 users, in the second hour 200 users, and so on. Each user starts a session and performs a sequence of requests as specified by the corresponding behavior model. When the session is finished, the user starts a new session. As think times we use the mean think times between state transitions as specified by the behavior models. For each resulting session, one session entry is created, that is written into a session log. The resulting session log comprises all requests of all user groups made in the forecasted week. From the session log, we

calculate different metrics that are associated with workload. We calculate the same metrics from the real log that we assumed as the future log. The results are compared against each other, and, thus, we are able to compare the forecasted workload against the real workload. The considered metrics are:

- Arrival rates: Number of sessions arriving per time interval
- Completion rates: Number of completing sessions per time interval
- Average number of sessions per time interval
- Request count of a service: Number of requests a service received
- Session length: Amount of requests made in a session or session duration in seconds
- Number of sessions
- Number of requests

From the six-month SIS logs, we cut out one month. The recorded requests start from 28/05/2018 00:00:00, and end at 24/06/2018 23:59:59, where the 28/05/2018 is a Monday. This is the first month of the six month SIS logs, that starts with a Monday. We are not able to consider more data, as the WESSBAS clustering would then utilize more RAM than available (see Section 6.2). We assume the first three weeks of the log as the past log, and the last week as the future log. In order to compare results obtained by using Prophet and Telescope, we run the experiment twice, once with Prophet as the forecaster, and once with Telescope as the forecaster. To simplify explanations, we name the resulting log of the experiment run where Prophet was the forecaster “Prophet log”, and the log of the experiment run where Telescope was the forecaster “Telescope log”. We change two configurations of Prophet. First, we set the width of the uncertainty intervals to 50 percent (default is 80 percent). Second, we set the time interval from days to hours. We do not change any other configurations in the time series forecasters. We execute the experiment as described above and collect results for the metrics.

There can be a slight error rate between the simulated intensity of a particular user group and its forecasted intensity. An error value between a simulated intensity value and the forecasted intensity value could occur, when from one hour to another hour there is a decrease between two consecutive forecasted intensity values. We start at each hour a number of users as specified by the forecasted intensity. Each of these users starts a new session when the previous session has finished. Sessions that have been started in one hour can still execute in the next hour (and also in the following hours). Therefore, the intensity value in the next hour could be too high, as too much sessions could be active. Our simulator tries to reduce the too high intensity value by starting less users at the beginning of the next hour, and also by finishing some sessions

that have started in the previous hour earlier. However, in the next hour the intensity value could be still slightly too high or now even slightly too low. We calculated the average error between a simulated intensity value and a forecasted intensity value and obtained as result approximately one percent. In this experiment, we assume that the simulated intensities conform to the forecasted intensities, even though the slight error rate remains, which results in an internal threat to validity. We provide the load test simulator as supplementary material to this thesis [Hid19].

6.3.2. Experiment 2: Regressors for Saturday and Sunday

With this experiment, we aim to answer RQ2, i.e, we want to investigate whether a context can increase the accuracy of a workload intensity forecast, and, thereby, also the future workload. We perform the steps 1, 3, 4 and 5. Step 3 is modified as we calculate exactly one past workload intensity directly from a session log, i.e., we do not calculate user groups. From the SIS logs, we consider recorded requests that start from 28/05/2018 00:00:00, and end at 10/06/2018 23:59:59. The last two days are Saturday (09/06/2018) and Sunday (10/06/2018). In this experiment, we forecast the workload intensity occurring on both of these days. This means, that we calculate the past workload intensity from the session log that contains the data from 28/05/2018 00:00:00 until 08/06/2018 23:59:59. In this time range, Saturday and Sunday each occur once. Saturday and Sunday describe an appropriate context, as the SIS is used much less on these days. We perform the intensity forecast twice, once without regressors, and once with regressors. We calculate two regressors, one for Saturday, which we refer to as Saturday regressor, and one for Sunday, which we refer to as Sunday regressor. The Saturday regressor values are 1 during Saturday, and otherwise 0. Analogous, the Sunday regressor values are 1 during Sunday, and otherwise 0. The regressors contain past as well as future values. They are passed with the workload intensity to a forecaster, which then predicts the future intensity based on a context. The intensity forecasts with and without regressors are compared against the real intensity that occurred on both days. For this purpose, we use the following metrics:

- Maximum intensity on Saturday
- Maximum intensity on Sunday
- Average intensity on Saturday
- Average intensity on Sunday

The forecaster we use for this experiment is Prophet.

6.3.3. Experiment 3: Extraction of Workload Intensities

With this experiment, we aim to answer RQ3. In experiment 1, we do not process the forecasted intensities. Here, we perform step 6 with the forecasted workload intensities from experiment 1. We execute the experiment twice, once with the forecasted intensities by Prophet, and once with the foreasted intensities by Telescope. The forecasted intensities are aggregated to exactly one intensity, which is the sum of the intensity values of all forecasted intensities at times t_1, ..., t_n, where t_1 is the timestamp of the first intensity values, and t_n is the timestamp of the last intensity values. We consider all load scenarios, except the "entire workload" scenario, as this scenario is already covered by experiment 1. For a particular load scenario, we obtain one or more intensity values from the aggregated intensity, which are set as the workload intensity of a load test simulating for that load scenario. The considered workload intensities comprise:

- Maximum intensity (single value)
- Average intensity (single value)
- Minimum intensity (single value)
- Sharpest increase in intensity (two consecutive values)
- Sharpest decrease in intensity (two consecutive values)

The testing effort only consists in the simulation of users, and, hence, we only investigate the extracted workload intensities for load tests. From the real intensity, we extract the same workload intensities for load tests, so that we can compare them against the workload intensities obtained from the forecasted data. In order to investigate if we cover the occurring maximum, average and minimum intensity in the future, and at the same time if the required testing effort is spent when simulating the users, we assume the following.

- If the single value of the intensity obtained from the forecasted data is roughly the same as the single value obtained from the real data, we covered the intensity correctly and spend the required testing effort. Roughly the same means up to approximately 10 percent difference.
- If the value of the intensity obtained from the forecasted data is clearly 10 percent higher than the real value, then we covered the intensity, but the testing effort is too high.
- If the value of the intensity obtained from the forecasted data is clearly 10 percent lower than the real value, then the intensity is not covered, and at the same time the testing effort was without success, and, thereby, too high.

The workload intensities consisting of two values (sharpest increase and decrease) are compared as follows.

- From the sharpest increase extracted from the forecasted data we calculate the percentage increase between the first and the second value. We do the same for the real sharpest increase. If the difference between both percentage increases is roughly the same (up to approximately 10 percent difference), and if the difference between the second values of the sharpest increases is also roughly the same (again up to approximately 10 percent difference), then we covered the sharpest increase correctly and spend the required testing effort.
- If the percentage increases are roughly the same, but the second value of the sharpest increase from the forecasted data is clearly over 10 percent higher than the real second value, then the sharpest increase is covered, but the testing effort was too high.
- If the percentage increase of the sharpest increase extracted from the forecasted data is clearly over 10 percent higher than the percentage increase of the real sharpest increase, and if the second value of the sharpest increase from the forecasted data is clearly over 10 percent higher than the real second value, then the sharpest increase is covered, but the testing effort was too high.
- If the second value of the sharpest increase from the forecasted data is clearly over 10 percent lower than the real second value, then the sharpest increase is not covered, and at the same time the testing effort was without success, and, thereby, too high.
- If the percentage increase of the sharpest increase extracted from the forecasted data is clearly over 10 percent lower than the percentage increase of the real sharpest increase, then the sharpest increase is not covered, and at the same time the testing effort was without success, and, thereby, too high.
- The sharpest decrease extracted from the forecasted data is compared to the real sharpeast decrease analogously, but here we compare the percentage decreases and the first values of the sharpest decreases.

In the cases where the real occurring intensity in the future is not covered by the returned load test, the user of our approach obtains false safety. The system is not tested correctly and, therefore, could not handle the real occurring load. This result is worse than investing more testing effort than required.

6.3.4. Experiment 4: Regressor for Special Event

With this experiment, we aim to answer RQ4. In experiment 1 and 3, the forecasted week ranges from 18/06/2018 00:00:00 until 24/06/2018 23:59:59. In real, the system experienced on the Tuesday of this week, at 20:00:00 o'clock, an extreme load. The intensity at this hour, which was in average 1418 active sessions, was clearly higher than normal. However, such a high value never was observed in the three weeks before that week. We expect that this high value will not be detected in experiment 3 as the maximum value. We assume for this experiment, that there was a special event when the high intensity occurred, and assume it as a context. However, our approach only functions when the special event was already observed in the first three weeks. As we investigated the intensity on the Tuesday in the week before the fourth week, we found four consecutive intensity values from 11:00:00 until 14:59:59 o'clock that are lower than the value of 1418 in the fourth week, but are quite close it. The values are 1139, 1126, 1081, and 1083. We assume that at that time there was also a special event, as the values are still clearly higher than normal. For this experiment, we assume that the special event was the same as in the fourth week. We input the intensity of the first three weeks to Prophet, and pass a regressor with the intensity. The regressor values are 1 at the timestamps where the high values occur, and 1 at the timestamp where the high value in the fourth week occurs. We then forecast the fourth week, and extract the maximum value from the result. We then compare the value against the real maximum value of 1418.

6.4. Experiment Results

In this section, we show the results of each experiment. In Section 6.5, we answer our RQs based on the results. We start with the results obtained from experiment 1.

6.4.1. Experiment 1: Simulation of the Forecasted Workload

The experiment is executed as described in Section 6.3.1. We obtain metrics for three logs: The real log, one calculated log using Prophet, and one calculated log using Telescope as the forecaster. The period of all three logs is the same. They start from 18/06/2018 00:00:00 and last until 24/06/2018 23:59:59. Table 6.3 lists the number of session entries (one session entry represents exactly one session) and the total number of request entries (one request entry represents exactly one request) in the session logs, and how long one session entry in the logs is. SD is the abbreviation for standard deviation.

	Real log	Prophet log	Telescope log
Number of sessions	667841	54124	37527
Number of requests	4368146	315907	232661
Mean session length (# requests)	6.54	5.84	6.2
SD session length (# requests)	332.6	11.1	15.67
Mean session length (in seconds)	240.76	3585	3767
SD session length (in seconds)	2607.63	5117	6054

Table 6.3.: Number of sessions and session length

The real log has a lot more session entries than the calculated logs, and for that reason also a higher number of request entries. The session length expressed through the average amount of requests in a session is similar for all three logs. In contrast, the average session length expressed through the session duration (in seconds) differs between the real log and the calculated logs. For the calculated logs it is approximately one hour, whereas for the real log it is four minutes. The differences between the number of sessions, number of requests, and the mean session length (in seconds) between the real log and the calculated logs are high. A simulated user by our simulator starts a new session when the previous session is finished. To obtain the high amount of session entries as in the real log, sessions would have to finish fast in the simulation, so that simulated users spawn lots of sessions. However, the length of sessions (in seconds) in the simulation is approximately one hour in average. This is due to the high think times between state transitions as specified by the behavior models. For example, it takes 15 minutes to transition from “studium” to “studium”, which is the most frequently executed state transition. We found that the think times calculated by WESSBAS are too high and decided to calculate them again, and then to restart the experiment with our own calculated think times. For this purpose, we consider the three weeks past log. We calculate the mean think time of a particular state transition as follows. We search for all its occurrences in the past log and sum up the think times. The resulting think time is divided by the amount of occurrences of this transition. The new calculated mean think times are actually smaller than specified by the behavior models. For example, the mean think time of the transition “studium” to “studium” is now 1.3 minutes. We explain in Section 6.5.1 why the mean think times calculated by WESSBAS are higher. Table 6.4 lists the new results obtained for the same metrics as in Table 6.3. As it can be seen, in comparison to Table 6.3, the values obtained from the calculated logs are now more similar to the values obtained from the real log. The number of session entries in the Prophet log is now higher than in the real log. In contrast, the number of session entries in the Telescope log is lower than in the real log. The same applies for the number of request entries. The mean session lengths (requests and in seconds) are similar for

	Real log	Prophet log	Telescope log
Number of sessions	667841	819508	612959
Number of requests	4368146	4814591	3743739
Mean session length (# requests)	6.54	5.88	6.11
SD session length (# requests)	332.6	12.26	11.48
Mean session length (in seconds)	240.76	234.89	229.07
SD session length (in seconds)	2607.63	314.51	315.76

Table 6.4.: Number of sessions and session length - Own calculated think times

all three logs. The real log, compared to the calculated logs, shows a high standard deviation in the session lengths.

Further presented results only include the results from the experiments where we used our own calculated think times for state transitions.

In the appendix, we provide two tables, Table A.2 and Table A.3, which list for each service how many requests it received, i.e., the request counts of the services. The services are ordered based on their actual request counts in the fourth week (real log column). In Figure 6.1, a bar chart of the request counts of the first 20 services is provided, so that differences between the request counts obtained from the logs can be spotted more easily (at least for the first 20 services). As it can be seen, there are a lot of similarities between the results. However, some major differences exist. For example, according to the real log the second highest called service is `predmety`, but according to the Prophet log it is `prijimacky`, and according to the Telescope log it is `term_st2`. For the arrival rates, completion rates, and average number of sessions per time interval (intensity) metric we obtained for each of these metrics a time series of values for each log, where a value was calculated for a time interval of one hour. The time series consist of 168 values, since we forecasted for one week ($7 * 24 = 168$). We want to compare the time series values obtained from the real log against the time series values obtained from the calculated logs, e.g., the arrival rates, and investigate if they are similar. For this purpose, we use the euclidean distance as similarity measure. The euclidean distance is defined as

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

where in our case $x_1, ..., x_n$ are the values of one time series, and $y_1, ..., y_n$ are the values of another time series. In the following, x describes the values of a time series calculated from the real log, a the values of a time series calculated from the Prophet log, and b the values of a time series calculated from the Telescope log. Table 6.5 shows the results. What can be seen is that the euclidean distances between the values of the time

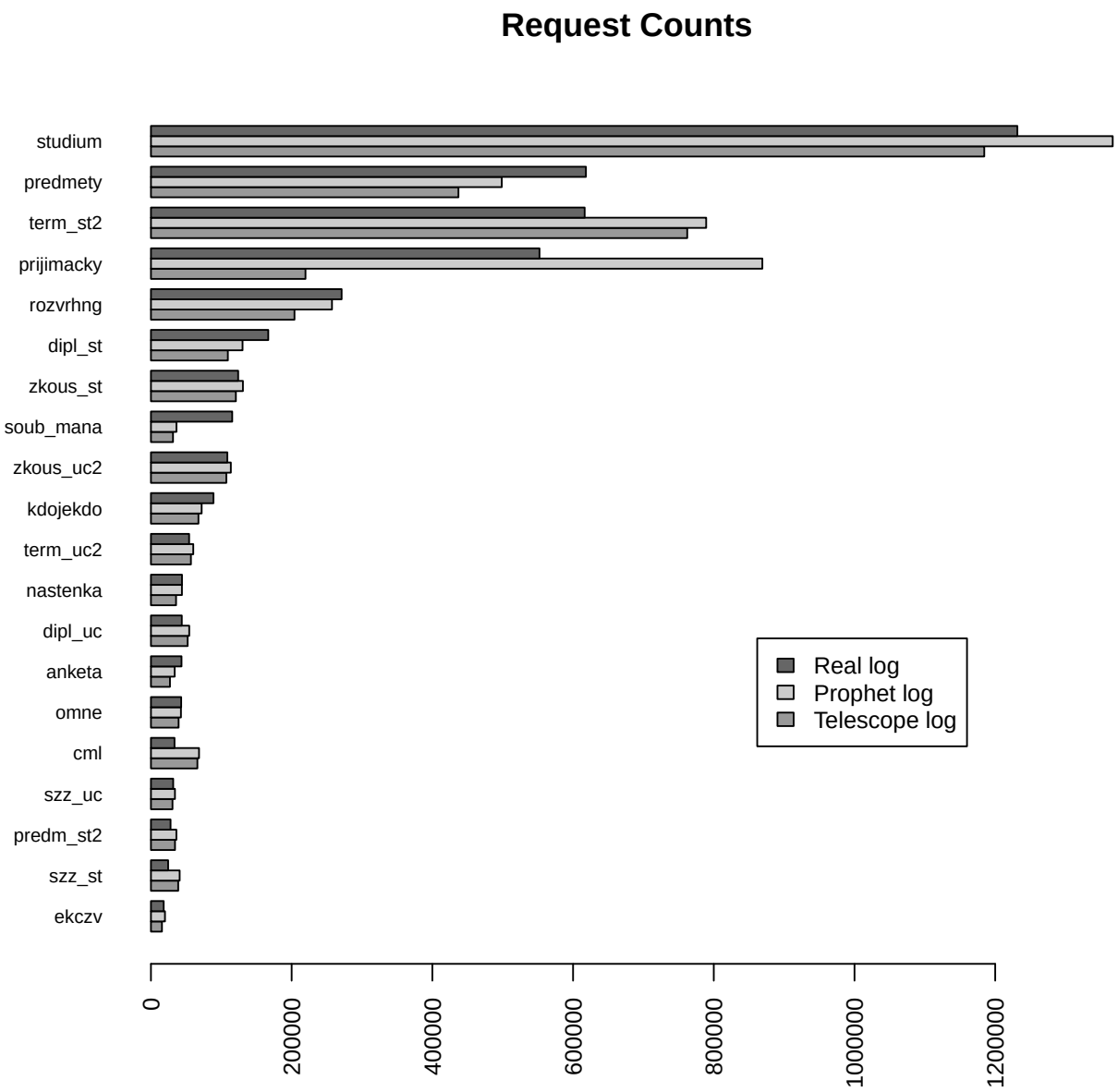

Figure 6.1.: Request counts of services - Bar chart

series calculated from the Telescope log and the real log are smaller than the euclidean distances between the values of the time series calculated from the Prophet log and the real log. The euclidean distances in the arrival and completion rates are clearly higher than the distances in the intensity, because thousands of sessions can arrive and complete in one time interval, but the average amount of sessions per time interval stays at large part below 1000.

	Prophet: $d(x, a)$	**Telescope:** $d(x, b)$
Arrival rates	17227.12	12111.7
Completion rates	17229.3	12155.78
Intensity	1591.22	1394.13

Table 6.5.: Euclidean distances between time series

6.4.2. Experiment 2: Regressors for Saturday and Sunday

Figure 6.2 shows the real workload intensity that occurred on 28/05/2018 00:00:00 until 10/06/2018 23:59:59. Each peak shows the intensity that occurred on a particular day (2 weeks = 14 days = 14 peaks). The last two peaks represent the workload intensity on Saturday and Sunday. The maximum value of the peak (maximum intensity) on Saturday is 222, whereas on Sunday it is 318. The average intensity on Saturday is 132.75, and on Sunday 174.75. We now forecast the intensity on both days. We start with the forecast without regressors. The result can be seen in Figure 6.3. Here, the last two peaks look very similar, and compared to the last two peaks of the real workload intensity, they are clearly too high. We calculate the same metrics as above. As maximum intensity on Saturday we obtain 564.42, which results in an error of 152.24 percent, as the real intensity is 222. For Sunday we obtain 563.51, which results in an error of 77.2 percent. Besides, the maximum intensity on Saturday is higher than on Sunday, whereas in the real intensity it is the other way round. The average intensities on both days are also clearly too high. On Saturday the average intensity is 309.91,

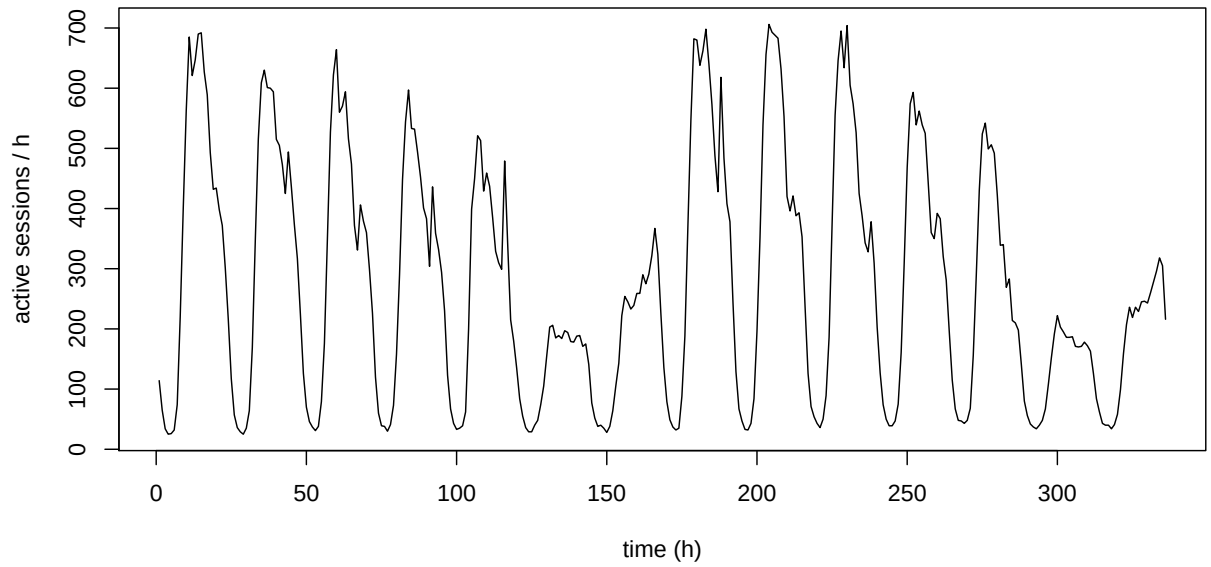

Figure 6.2.: Real workload intensity

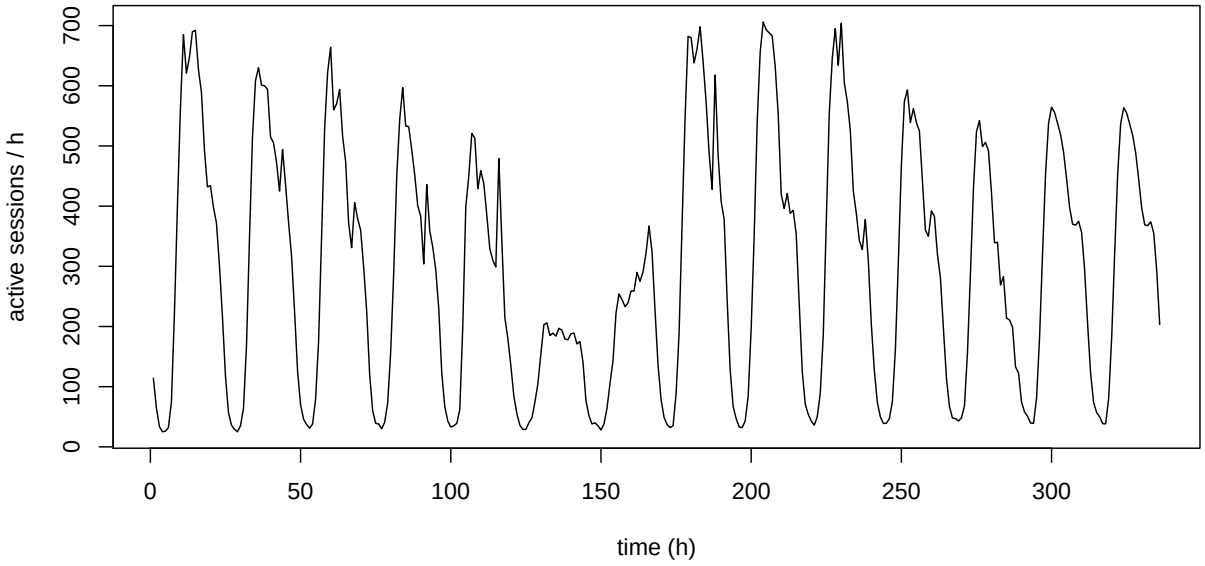

Figure 6.3.: Intensity forecast without regressors

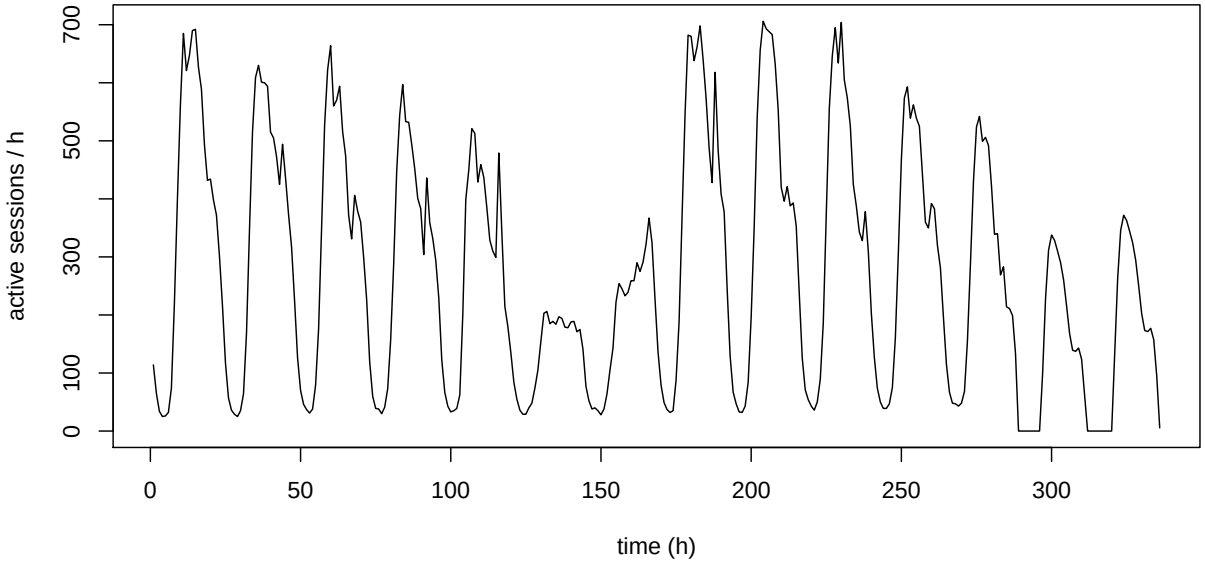

Figure 6.4.: Intensity forecast with regressors

and on Sunday 309. Both values, as well as the maximum intensity values, are very similar. Figure 6.4 shows the forecast with regressors. Some intensity values in the forecast had to be set to zero, as they were negative. The last two peaks, compared to the forecast without regressors, are now smaller. Also, the second of the last peaks is a little higher than the first. We calculate again the metrics. As maximum intensity on Saturday we obtain 337.74. Compared to the forecast without regressors, we were able to decrease the error from 152.24 percent to 52.14 percent. On Sunday, we obtain as

	Real intensity	Without regressors	With regressors
Maximum Saturday	222	564.42	337.74
Maximum Sunday	318	563.51	371.8
Average Saturday	132.75	309.91	131.35
Average Sunday	174.75	309	152.87

Table 6.6.: Workload intensities on Saturday and Sunday

maximum intensity 371.8. Here, we were able to decrease the error from 77.2 percent to 16.92 percent. Also, the maximum intensity on Saturday is now smaller than the maximum intensity on Sunday. The average intensity on Saturday is 131.35, and on Sunday 152.87. These intensities are clearly more close to the real average intensities than the average intensities obtained from the forecast without regressors.

With the results, we can infer that both peaks are now more close to the last two peaks of the real intensity. However, the peaks still differ. Table 6.6 summarizes the results of this experiment.

6.4.3. Experiment 3: Extraction of Workload Intensities

Table 6.7 shows the experiment results. We extracted workload intensities for load tests from the real intensity of the fourth week, and from the aggregated intensities. The aggregated intensity of the forecasted intensities from Prophet is called “Prophet intensity”, and the aggregated intensity of the forecasted intensities from Telescope “Telescope intensity”. With both, Prophet and Telescope, we do not cover the real maximum intensity, and the testing effort is without success, as both maximum values from the Prophet and Telescope intensity are not even half as high as the real maximum. With Prophet, we cover the average intensity, but the testing effort is too high, as the percentage difference to the real average is approximately 20 percent. With Telescope, we are also able to cover the average intensity, as the percentage difference to the real

	Real intensity	Prophet intensity	Telescope intensity
Maximum	1418	655	637
Average	265	321	235
Minimum	36	19	30
Sharpest increase	(89, 180)	(132, 266)	(129, 305)
Sharpest decrease	(1418, 584)	-	(181, 90)

Table 6.7.: Extracted workload intensities for load tests

average is approximately 12 percent, which we consider as acceptable, plus we spend the required testing effort. With both tools, the minimum intensity cannot be covered (61 and 18 percentage difference). The sharpest increase is covered by Prophet, but with too much testing effort. Both, the real and the sharpest increase of the Prophet intensity have a percentage increase of around 100 percent, but the second value of the sharpest increase of the Prophet intensity is clearly higher. The same applies for the sharpest increase of the Telescope intensity, but here also the percentage increase is clearly higher between both values, and, hence, the testing effort is even higher. From the Prophet intensity, no sharpest decrease is detected. The sharpest decrease of the Telescope intensity does not cover the real sharpest decrease, as its first value is clearly lower than the first value of the real sharpest decrease.

In the real intensity there are two consecutive high values (863 and 1418), that are higher than the other values. When they would not exist, the real maximum intensity would be 685. In this case, the maximum intensities from the Prophet and the Telescope intensity would cover the real intensity, and the required testing effort is spent.

6.4.4. Experiment 4: Regressor for Special Event

Figure 6.5 shows the real workload intensity that occurred on 28/05/2018 00:00:00 until 24/06/2018 23:59:59. Each peak shows the intensity that occurred on a particular day (4 weeks = 28 days = 28 peaks). As it can be seen, in the last two weeks on Tuesdays the corresponding peaks have very high values. In the week before the last

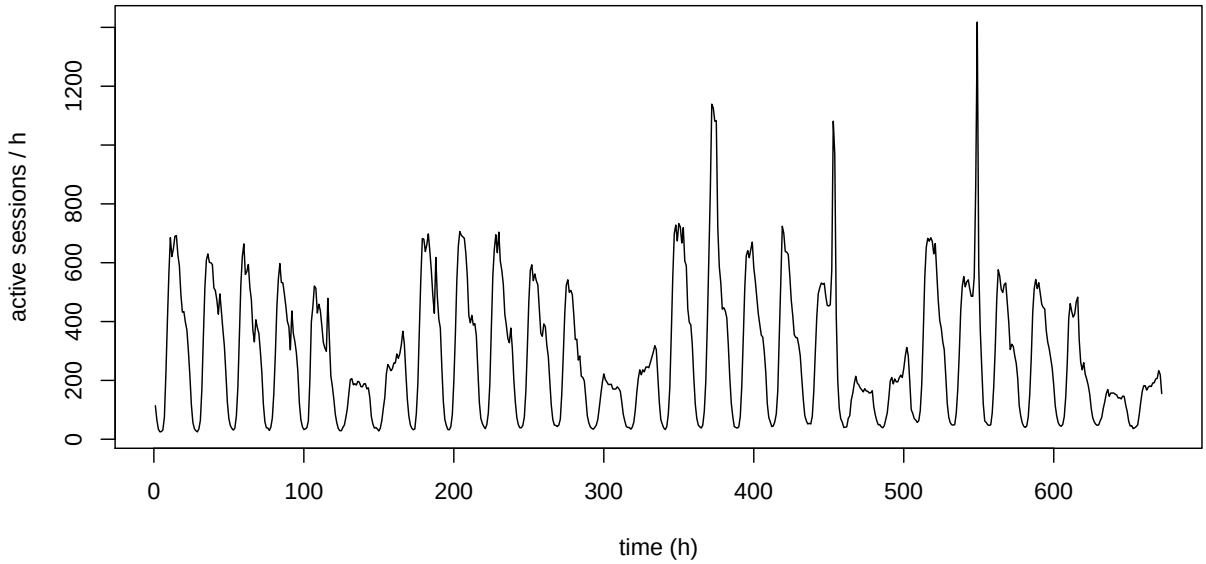

Figure 6.5.: Real intensity

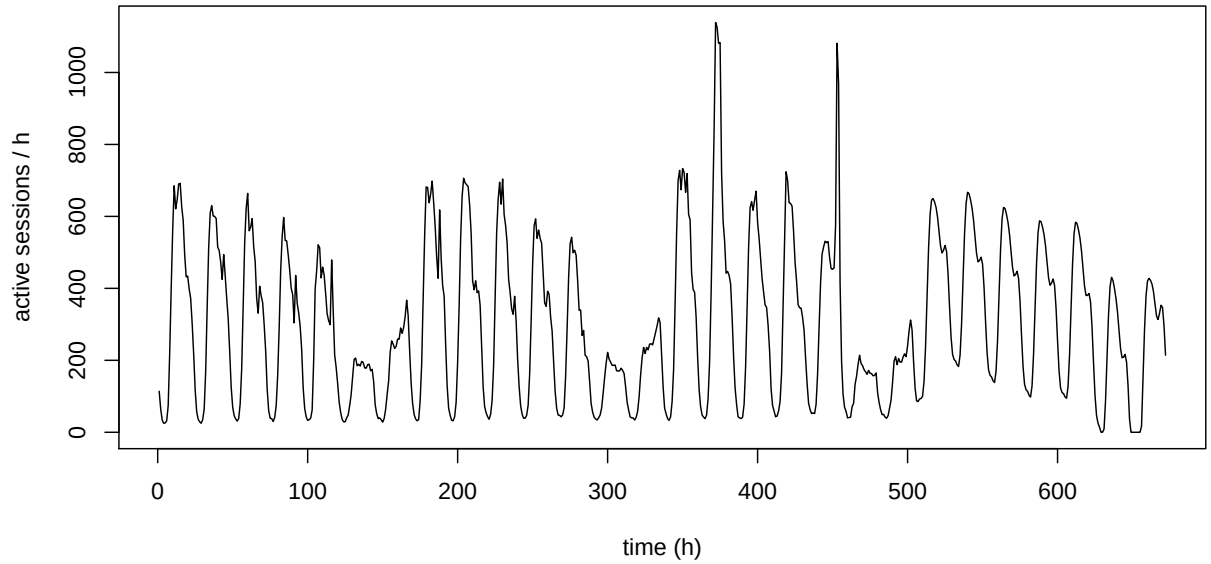

Figure 6.6.: Intensity forecast without regressor

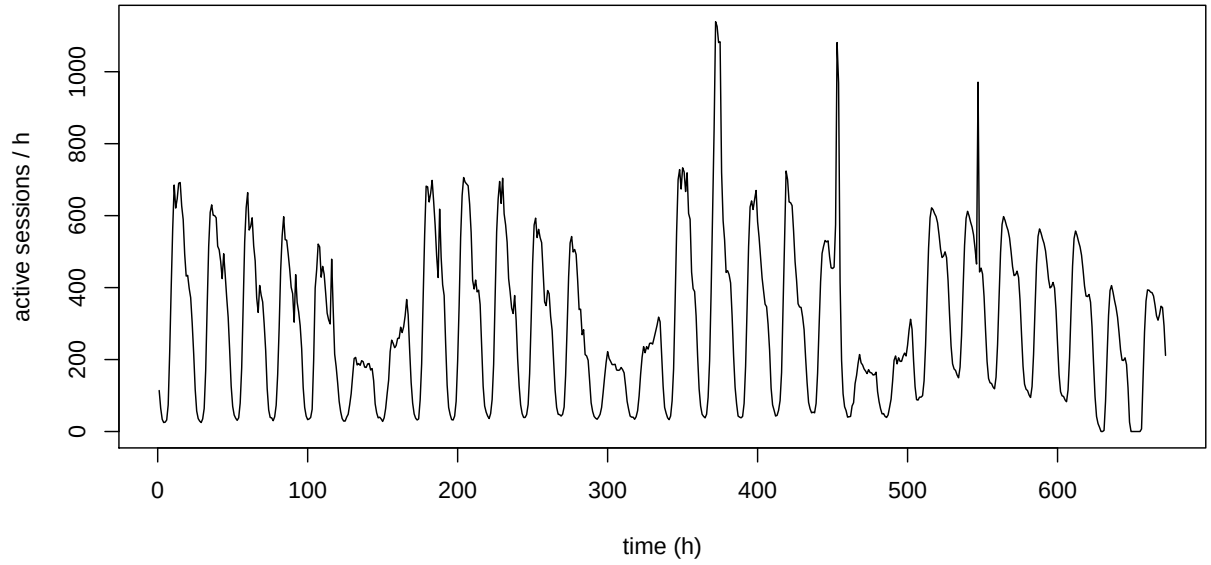

Figure 6.7.: Intensity forecast with regressor

week the values are 1139, 1126, 1081, and 1083, and in the last week the value is 1418. We first forecast the intensity of the fourth (last) week without regressors. The result can be seen in Figure 6.6. As it can be seen, the forecast does not cover the high value on Tuesday. The maximum value of the forecasted intensity is 666.9, which results in an error of 52.97 percent. Now, we forecast the fourth week again, but this time we add the regressor to the forecast. The result is shown in Figure 6.7. The maximum value we now obtain is 970.9. We were able to reduce the error from 52.97 percent to 31.53

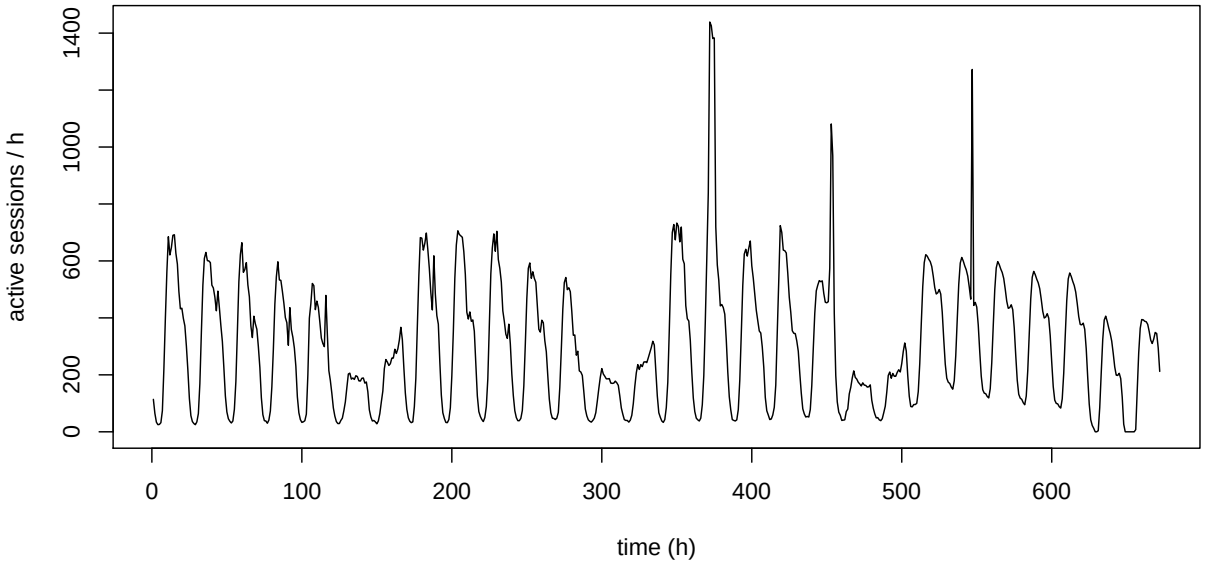

Figure 6.8.: Forecast with modified values

percent. However, the result is still not acceptable. It seems that the occurrences of a context have to lead to similar intensities. We want to investigate this and modify the high values in the third week so that they are similar to the high value in the fourth week. For this purpose, we add to all four values 300, so that the values are around 1400. This ensures that both special events lead to similar intensity. The modified values are 1439, 1426, 1381, and 1383. Figure 6.8 shows the forecast with the regressor again, but now with the modified values. The maximum value in the forecasted week is now 1272.3. The error is now 10.28 percent, which is acceptable, i.e., we cover now the maximum intensity that occurs in the future, and we spend the required testing effort.

6.5. Analysis of Results

In this section, we analyze the experiment results and answer the RQs. We start with the analysis of the results from experiment 1.

6.5.1. Experiment 1: Simulation of the Forecasted Workload

Before we restarted the experiment we used the mean think times of state transitions calculated by WESSBAS, and found out that they are too high. The high think times in the behavior models result from a design decision of the WESSBAS approach for the think times calculation, which is inappropriate for the used data set (SIS logs) in

this evaluation. WESSBAS considers only the different think times of a particular state transition for the calculation of its mean think time. For example, when the think times of a particular state transition are $2s$, $2s$, and $116s$, then WESSBAS considers the $2s$ think time only once, and the resulting mean think time is $(116s + 2s)/2 = 59s$. Without the duplicate removal, the mean think time would be $(116s + 2s + 2s)/3 = 40s$. As we processed the SIS logs, we ensured that the time distance between two consecutive requests in a session is maximum 30 minutes. For a particular state transition, a wide range of think times can exist. From one second to 30 minutes everything could be covered. However, smaller think times still appear way more often than greater think times, but WESSBAS considers the same small think times only once for the mean calculation. In the processed SIS logs, wide ranges of think times for the most frequently occurring state transitions exist. For, e.g., the "studium" to "studium" transition, this results in a mean think time of 15 minutes. For our calculation of the mean think times, we did no duplicate removal.

We now want to investigate, why the Prophet log has more session and request entries than the real log, and why the Telescope log has less session and request entries than the real log (see Table 6.4). The Prophet log was the result of the load test simulation with forecasted intensity values from Prophet, and the Telescope log was the result of the load test simulation with forecasted intensity values from Telescope. For each of the simulations, we calculate how many users were simulated in average per hour. For this purpose, from the Prophet and the Telescope log, we first calculate the workload intensity. Then, for each log, we sum up all resulting intensity values, and divide the result by 168, which is the amount of hours of the forecasted week. The result is 318 users that are in average simulated per hour by our simulator when using the forecasted intensity values from Prophet. Using the Telescope intensities, we simulated 232 users in average. Without the error rate during the simulation of the forecasted intensities the average values would be 321 and 235 (see Table 6.7). The average intensity calculated from the real log is 265. This means, that the forecasted intensity values of Prophet in average are too high, and that the forecasted intensity values of Telescope in average are too low. Thereby, in the simulation with the values forecasted by prophet we simulate in average 53 users per hour too much, and in the simulation with the forecasted values by Telescope 33 users too little. This is also reflected in the number of session entries obtained from each log. The distance between the number of session entries in the Prophet log and the number of session entries in the real log ($819508 - 667841 = 151667$) is higher than the distance between the number of session entries in the Telescope log and the number of session entries in the real log ($667841 - 612959 = 54882$), as the error value of 53 users is higher than the error value of 33 users. The results hypothesize that Telescope forecasts the workload intensity with higher accuracy, and, hence, seems to be more appropriate than Prophet for our approach. The mean session length regarding both, the amount of requests made in the session and the session duration in seconds, is

similar in all three logs. This indicates that the calculated behavior models are correct, and that our calculated think times are meaningful. The standard deviation in session lengths of the real log is very high, as in the processed SIS logs still a lot of sessions exist that are very long.

Table A.2 and Table A.3 list the request counts of all services, and Figure 6.1 shows a bar chart of the request counts of the first 20 services listed in the tables. We can see that there are some similarities between the results obtained from the logs, but also differences. We first want to investigate the major differences. The greatest difference we observe is the request count of the `prijimacky` service. We want to investigate this in detail. We found out that in the second of total five behavior models, which occurs with approximately 15 percent probability during workload generation, there is a 100 percent probability that the start state transitions to `prijimacky`, i.e., the first request of a user following that model always addresses this service. The transition `prijimacky` to `prijimacky` is then chosen with approximately 68 percent probability, and with approximately 22 percent the behavior model transitions from `prijimacky` to the end state. We now have a look on the forecasted workload intensity of the user group described by the second behavior model. We start with the intensity forecasted by Prophet. Figure 6.9 shows the forecast. The first 504 points (21 peaks = 21 days = 3 weeks) are the past intensity values we have calculated, and the last week is the forecast. The curve starts at Monday (28/05/2018), 12 am. The overall curve shows the intensity of exactly four weeks. As it can be seen from the diagram, in the forecasted week the decrease of a peak is not as strong as the decreases of the other peaks in the first three weeks. This means, when we simulate the curve, that there is always a considerable

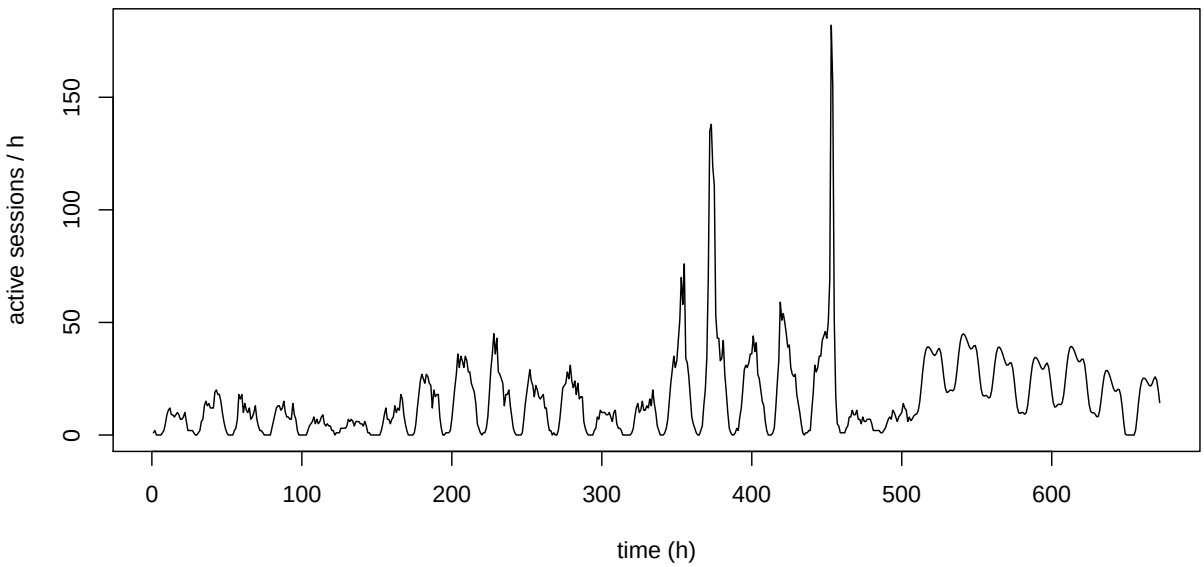

Figure 6.9.: Inaccurate intensity forecast by Prophet

high amount of users on the system which performs requests to the `prijimacky` service, as with high probability the transition from `prijimacky` to `prijimacky` is chosen, or the session finishes. In case the session finishes, the same user would start a new session and would again request the `prijimacky` service first. Besides, in comparison to the other peaks, in the most cases the forecasted peaks are higher. This further increases the number of requests received by the `prijimacky` service. To verify that the forecasted intensity in the last week is too high, we calculate the average and the median from the intensity in the first three weeks and also from the intensity in the last week. We calculate the median, as in the first three weeks there are a few high values which seem to be outliers. They could have a too strong impact on the average. The average intensity in the first three weeks is 13.88. In the last week, the average is 23.83, which is approximately 72 percent higher than the real average. The median value in the first three weeks is 8, whereas in the last week it is 24.28, which is more than three times higher. We infer that in the last week the forecasted intensity is too high.

In Figure 6.10, we see again the forecasted intensity of the user group described by the second behavior model. This time Telescope is the forecaster. The peaks of the last week (forecast) look very similar to the peaks of the first week, but are much smaller than the peaks of the second and the third week. It seems to be that the intensity in the last week is too low, and, hence, less users are simulated that send requests to the `prijimacky` service. To verify this observation with measures, we again calculate the average and the median from the intensity in the last week. In the first three weeks the average intensity is 13.88, and this time the average intensity in the last week is 5.81, which is approximately 58 percent lower. The median is 6.19 and approximately 23

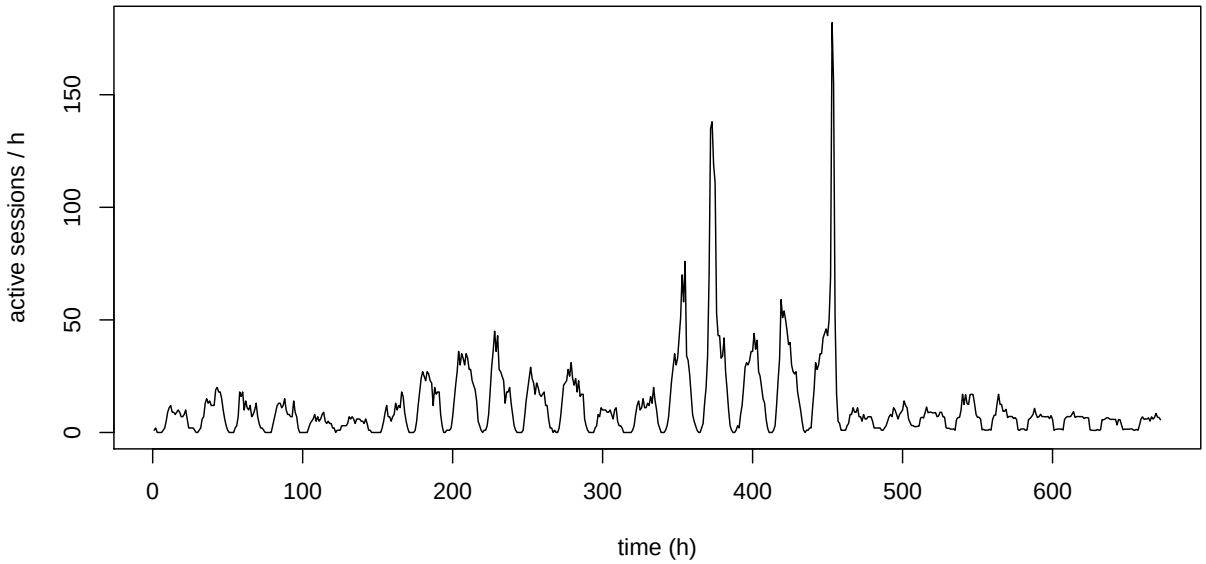

Figure 6.10.: Inaccurate intensity forecast by Telescope

percent lower than the median in the first three weeks. We infer that in the last week the forecasted intensity is too low. The inaccuracies in the forecasts lead to totally different request counts of the `prijimacky` service.

As we compared the order of the request counts between the logs, we noticed that the orders differ from each other. We assume the order of the request counts of the Prophet and the Telescope log to be more similar to the order of the request counts of the log containing the data from the first three weeks. To investigate this, we calculate from this log the request counts. The order of the services based on the request counts can be seen in the appendix in Table A.4 and Table A.5 for each of the logs. Real log (1) is the log containing the data from the first three weeks, and real log (2) contains the data from the fourth week. To show that the order of the request counts obtained from the simulations is more similar to real log (1) than real log (2), we again create a bar chart, this time with the request counts of the real log (1), and the Prophet and the Telescope log. We divide the request counts of real log (1) by three, to obtain the average request counts of one week. The bar chart shows the request counts of the 20 services that had the highest request counts in the first three weeks. It is depicted in Figure 6.11. As it can be seen, not only the order is more similar, but also the request counts by themselves. Hence, we can confirm our assumption. We conclude, that WESSBAS' behavior model extraction functions as expected. The reason is that WESSBAS clustered the data of the first three weeks, and obtained request counts of services are very similar to the request counts in the first three weeks. That the order of the request counts is more similar to real log (1) than real log (2) is a problem of inaccurate forecasts of the intensities of user groups. Inaccurate intensity forecasts even result in major differences in the request counts (as has been shown for the `prijimacky` service).

The results from Table 6.4 hypothesized that Telescope forecasts the workload intensity with higher accuracy than Prophet. The results from Table 6.5 strengthen this assumption further, as the euclidean distances between the values of the time series calculated from the Telescope log and the real log are smaller. No Baseline exists that tells us whether the obtained distances are high or low. However, the euclidean distances of the intensities at least show that the intensity forecasts were not optimal. When we divide, for example, 1591.22 by 168, we obtain a mean square error of approximately 10 in the simulated users per hour, i.e., per hour 10 users are simulated too little or too much (the value would have been even higher when we would have calculated the linear distances between the real and the simulated intensities). With higher forecast accuracy not only the euclidean distances between the real intensity and the simulated intensity would be lower, but also the euclidean distances between the arrival and completion rates, as they depend on the amount of simulated users.

Summarized, based on the experiment results and the analysis, we can infer that inaccurate intensity forecasts made by the used time series forecasters hinder our

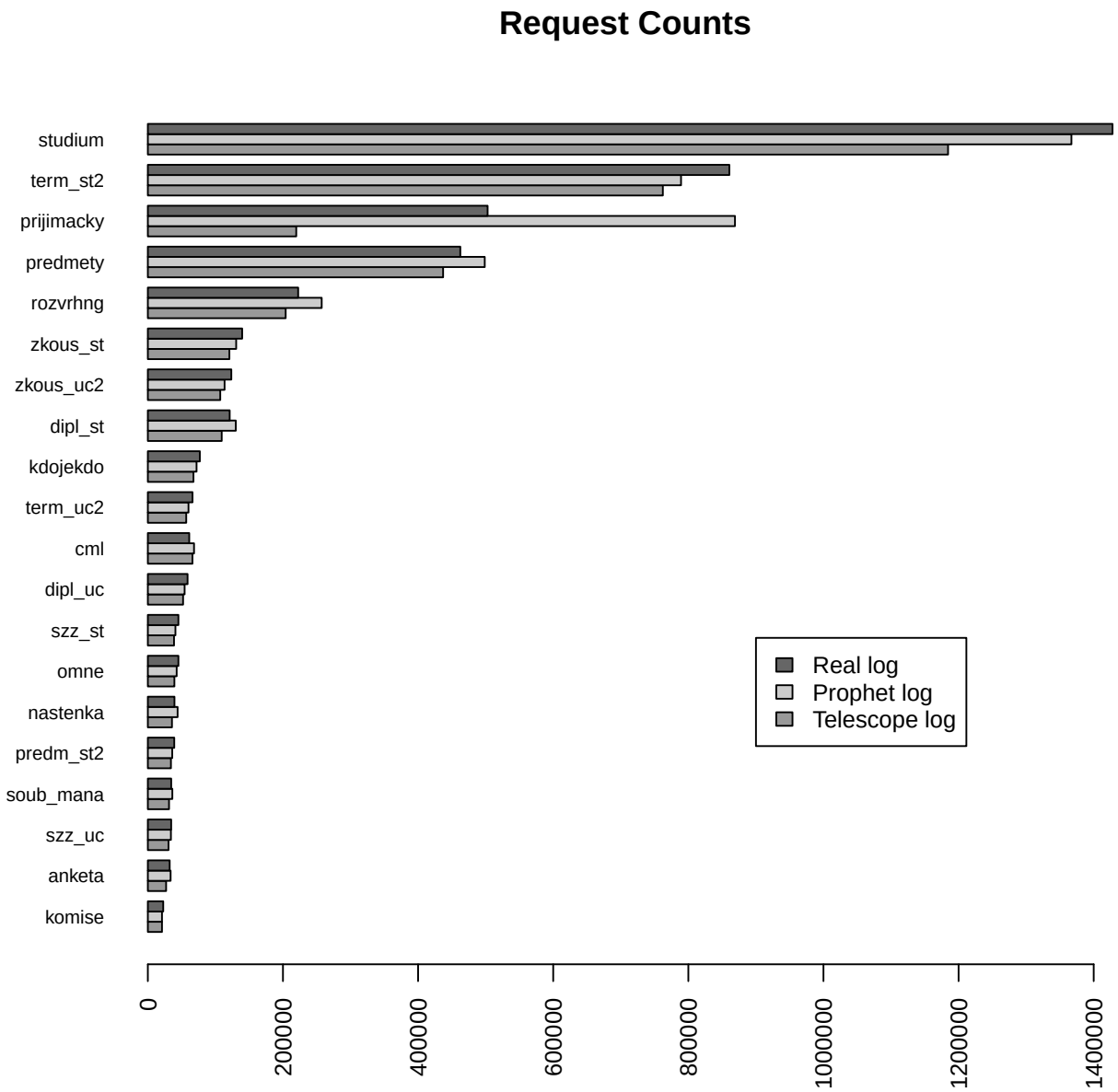

Figure 6.11.: Request counts of services - Bar chart (2)

approach to do accurate workload forecasts. With more accurate intensity forecasts, the simulated workload could be more close to the real workload. In its current state, our approach has limitations regarding the accuracy of the forecasted workload. This answers RQ1.

6.5.2. Experiment 2: Regressors for Saturday and Sunday

With our experiment results, we investigate whether a context can increase the forecast accuracy. This also would answer RQ2, which asked what the impact of contexts on the forecast accuracy is. As we answered RQ1, we already found out that the intensity forecasts hinder our approach to forecast the future workload accurately. Thereby, it is meaningful to investigate whether contexts can help to increase the accuracy of an intensity forecast. The experiment results show that this is the case. In the intensity forecast without regressors, the forecasted intensity is too high. For example, the maximum intensity on Saturday is 564.42, and the average intensity 309.91. In the real intensity, the maximum is 222, and the average is 132.75 on Saturday. Also, the workload intensity on Sunday should be higher than on Saturday, as can be seen from the real intensity, which is not the case. After we passed the context in form of regressors to the tool, the maximum intensities on Saturday and Sunday are clearly lower and did get more close to the real maximum intensities. The same applies for the average intensities. Also, the intensity on Sunday is higher than on Saturday now, which conforms with the real intensity. As both peaks are clearly lower now, we also managed to reduce the testing effort. When we return a load test to the user that tests for, e.g., the maximum intensity during Saturday and Sunday, the load test simulates a lower amount of users as it would have tested when no context would have been considered. The reason is that the more users are simulated with a load test, the more system resources are utilized, i.e., the resource usage increases. Tools that execute load tests, like Apache JMeter [AJ19], include also a ramp-up time that defines the amount of time it will take to add all simulated users to the test execution [Bla17]. The more users are simulated, the higher the ramp-up time, and, hence, the higher the test execution time. We infer, that contexts can decrease the testing effort. This answers in part RQ4.

Summarized, the results have shown that contexts can increase the accuracy of the intensity forecast, and hence can increase the accuracy of the forecasted workload. This is the answer to RQ2. Besides, contexts can decrease the testing effort.

6.5.3. Experiment 3: Extraction of Workload Intensities

With the analysis of the results of experiment 1 we already found out that the forecasted intensities by the used time series forecasters are not accurate. This is also reflected in the results we obtained with experiment 3. Due to inaccuracies, occurring load scenarios in the future are either not covered at all, or the testing effort is too high.

In the current state, our approach in some cases returns load tests to the user, which do not cover the requested load scenarios that occur in the future. Hence, when a user

executes the load test, the effort he spends is without success, but what is more worse is that he obtains false safety. It could be, that the SUT cannot handle the real occurring load, as the user did not test for it. For example, when his web store cannot handle the real occurring load, it could crash, and the user of our approach could lose customers and money. This is not unlikely when the maximum workload the system will experience is not covered by a load test.

With the experiment results we have shown that contextual information in some cases is necessary. The real maximum intensity value of 1418 is not covered by our approach. When we assume that during the occurrence of this value there was a special event, then the used time series forecasters have no knowledge about it, and, therefore, cannot forecast this value. In experiment 4, we have seen that we did get more close the real maximum intensity value after we have provided a context.

Summarized, the answer to RQ3 is that in its current state, our approach in some cases is not able to cover load scenarios occurring in the future, and that in the most cases the spent testing effort is too high.

6.5.4. Experiment 4: Regressor for Special Event

As we analyzed experiment 2, we already found out that a context can help to decrease the testing effort. The results of experiment 4 show that a context can also help to cover a requested load scenario. However, we have also found out that we only get an acceptable result when the occurrences of a context lead to similar intensities. With the modified values in the third week and a regressor we were able to decrease the error from 52.97 percent to 10.28 percent. With the real values and a regressor we were able to decrease the error from 52.97 percent to 31.53 percent. In both cases, the obtained maximum is clearly more close to the real maximum. We have shown that contexts help to focus only on the relevant workload. However, without the modified values, the result was still not acceptable and we were not able to cover the real maximum. We infer, that contexts help to cover a requested load scenario, but the occurrences of a context have to lead to a similar intensity.

Summarized, contexts can help to decrease the testing effort and to increase the coverage. This is the answer to RQ4.

6.6. Threats to Validity

In this section, we investigate whether our evaluation is valid. For this purpose, we analyze threats to validity. We start with internal threats.

Internal threats: We processed the SIS logs before we have used them in our experiments. We applied rules to the logs to decrease the amount of data and to calculate session identifiers. We cannot guarantee, that the processed logs are still representative for the real workload the SIS experienced. We suggest to execute the experiments again with another real world data set that is not processed, and to investigate whether the evaluation results will be the same.

We have built a load test simulator that simulates the forecasted workload intensities. We assumed that the simulated intensity conforms to the forecasted intensity, but there is an error rate between both intensities, which could impact the obtained experiment results. However, we calculated the average error between a simulated intensity value and a forecasted intensity value and obtained as result only one percent. Thus, we assume that the impact on the experiment results is very low. Nevertheless, to be sure, we suggest to run experiment 1 again and to execute a real load test that simulates users based on the forecasted intensities.

External threats: All of our experiments were executed with only one data set. Hence, our evaluation results cannot be generalized. We cannot guarantee that the results are the same for other data sets. We suggest to execute the experiments again and to test the approach with more data sets.

Construct threats: In experiment 4, we assumed that a special event occurred during the high observed workload. The reason for the high workload could have been another. Furthermore, we assumed that the same event occurred one week earlier, so our approach can consider the event as a context. The experiment should be executed again with a data set annotated with a real recurring event that is known.

We only changed two configurations of Prophet. We did not change any other configurations in the used time series forecasters. Especially Prophet provides many configuration options [fac19]. Maybe the forecast accuracy could have been increased when the tools were configured properly. We suggest to investigate which configuration options exist, and to try out different configurations of the tools.

Conclusion threats: For our experiment, we restricted to fundamental statistical measures such as single values like the mean and the maximum. For future work, we suggest analyzing our results with more rigorous statistics.

Chapter 7

Conclusion

In this chapter, we provide a summary of this thesis, discuss whether we have achieved our goals, and propose future work. In Section 7.1, the summary can be found. In Section 7.2, the retrospective, we discuss whether we were able to achieve our goals. In Section 7.3, we propose work for the future we were not able to complete in the scope of this thesis.

7.1. Summary

In this thesis, we developed an approach to forecast the future workload based on historical workload and a given context. The goal of this work was to extract one or more load tests from the forecasted workload that test for relevant occurring load scenarios in the future.

The historical workload is calculated from recorded request logs of a production system. We have build a session log generator to first generate a session log from the recorded request logs, that is a representation of request logs WESSBAS can process. We use WESSBAS to extract different user groups in form of behavior models from the session log. The behavior models are the result of a clustering of the session entries within the session log, where each session entry represents exactly one session. We extended the clustering process to retain the information of which session entry was considered for which resulting cluster of the clustering process. Each resulting cluster is converted into one behavior model. After the clustering, we have for each behavior model a list of session entries. For each behavior model, we calculate with the session entries the part workload intensity of the user group described by the behavior model. The past workload intensity is a time series of values, that can be the input to an existing time series forecaster. We have developed a context description language (CDL) to provide

context descriptions containing contextual information. Contextual information includes, for example, that in the time range the user wants to test for an event will occur, which will have an impact on the experienced workload. When such a context description is provided by the user, our approach converts the contextual data into one or more regressors, which are a representation of the data that can be understood by time series forecasters Prophet and Telescope. The regressors and calculated workload intensities are passed to one of these forecasters, which then forecasts the future workload intensity of each user group. From the forecasted workload intensities, we then identify occurring load scenarios in the future. The user gets a load test for each load scenario he wants to test for. When he wants to test for, e.g., high workload, we extract the maximum value of the forecasted intensities, and set it as the workload intensity of the load test we will return to the user. Furthermore, we calculate a behavior mix holding occurrence probabilities of the behavior models during workload generation. The occurrence probabilities are calculated from the forecasted intensities. We pass the workload intensity of the load test and the behavior mix to WESSBAS, which finally generates from the behavior models, the behavior mix and the workload intensity a load test, that is then returned to the user.

We implemented the forecast process and embedded it into the ContinuITy approach. The evaluation has shown, that contexts help to focus on the relevant workload and to decrease the testing effort. However, we have also found out that our approach in its current state has limitations regarding the accuracy of the forecasted workload.

7.2. Retrospective

In Section 1.2, we listed the goals we wanted to achieve with this thesis. We now discuss for each goal, if we were able to achieve it.

The first goal of this thesis was to develop a description language to express occurring contexts in the future. We developed such a description language. The language can be used to express occurring events in the future, or to express a time series that impacts the experienced workload (e.g., temperature curve, or price history of a product). We achieved this goal, but the language is restricted to events that were already observed in the past. We suggest for future work to extend the language to express also events that have never been observed in the past.

The second goal was to find a suitable workload representation for our approach. The workload representation should cover the user behavior and the workload intensity, should be usable to forecast the future workload, and it should be able to extract load tests from the workload. We successfully have found such a workload representation. We

represented the user behavior through behavior models (Markov chains), and calculated for each of them a time series of values representing the workload intensity.

The third goal was to forecast the future workload accurately. The evaluation has shown, that our approach in its current state has limitations regarding the accuracy of the forecasted workload. However, we were able to increase the forecast accuracy by providing contexts to the approach. Hence, we achieved this goal in part.

The fourth goal was to extract relevant load tests from the forecasted workload. The evaluation has shown that due to inaccuracies in the forecasted workload, our approach in some cases is not able to cover real load scenarios that will occur in the future. Furthermore, in the most cases the testing effort would be too high when the user would execute load tests obtained from our approach. We were not able to achieve this goal.

The fifth goal was to implement our approach and to embed it into ContinuITy. We successfully achieved this goal. A user of ContinuITy is now able to generate context-aware load tests by passing a context description to it.

The sixth goal was to evaluate our approach. We found out, that our approach in its current state is not able to do accurate workload forecasts, and that returned load tests do not always cover real load scenarios that will occur in the future. Furthermore, we found out that contexts can help to increase the forecast accuracy and the coverage of load scenarios. We successfully achieved this goal.

7.3. Future Work

In this section we propose work for the future, that we could not complete in the scope of this thesis.

The first work we recommend is to extend the CDL to be able to test for an event that is assumed to occur in the future and that has never been observed in the past. Since this event was never observed in the past and the workload that will be experienced cannot be foreseen (except the event is very similar to another event that already occurred), the user would have to provide assumptions in the context description. An idea would be to include the possibility to express that in the future an event will occur, and during this event the workload will be, for example, twice as high as normal.

We suggest to add more workload scenarios a user could want to test for to our approach. Currently, only the entire, high, medium and small workload scenarios can be requested, as well as a sharp increase and a sharp decrease in workload. Two further meaningful load scenarios would be slow increase and slow decrease in workload. During the simulation of these scenarios it can be observed whether a scalable application scales

normally up and down. For a static application it could be interesting to investigate whether the application functions normally when workload increases and decreases.

During our evaluation we have found out that the execution of the WESSBAS behavior model extractor requires a lot of RAM to cluster a big amount of session entries and when the data set includes to many different services. With a machine that has 128 GB RAM we were able to cluster only three weeks of data with 50 different services. We suggest to replace the WESSBAS for the calculation of the Markov chains by another approach, which is able to process more data. We recommend to calculate the Markov chains using data stream clustering [GMMO00].

The evaluation has shown that time series forecasters Prophet and Telescope seem to not suit well to forecast the future workload intensities of user groups. However, as already discussed in Section 6.6, it could be that the tools were not configured properly. We suggest to try out different configurations of the tools in order to see if the forecast accuracy can be increased. Besides, further time series forecasters could be researched that are able to forecast the future intensity based on contexts, and to investigate whether they suit better for our approach.

In Section 6.6, we listed further threats that could have impacted our evaluation results. For each threat, we suggested how to eliminate it. For future work, we recommend to consider the provided suggestions. Summarized these are:

- Execute the experiments again with another real world data that is not processed
- Execute a real load test instead of simulating a load test
- Execute the experiments with more data sets
- Consider for the experiments data sets annotated with real reccurring events
- Try out different configurations of the used time series forecasters
- Analyze the experiment results with more rigorous statistics

Appendix A

Appendices

In this chapter, we attach further outcomes of this thesis.

A.1. Omitted categories

Table A.1 lists the categories from the SIS log that we did not consider for our evaluation.

Category	Meaning	Role
info2	Support for students with special needs in the UK	
rozcestnik	Navigation for third party application	Student
konzultace	Office hours	
odmeny	Remuneration for the trainers	
ddipl	Diploma supplements	Diploma students
term_st	- not accessible, maybe not a service -	
dotaznik	Questionnaire	
ave_motd	SIS report - News	Admin
dbschema	Database schema	Admin
akreditace	Acreditation	Student
predmety_en	Edit English names of objects	Admin
erb	Error buster	Admin
whois	Search for persons and organizational units	Student
pisemnosti	PDF documents	

Table A.1.: Omitted categories

A.2. Request Counts of Services

Table A.2 and Table A.3 list the request counts of services. The services are ordered based on their actual request counts in the fourth week, which lasts from 18/06/2018 00:00:00 until 24/06/2018 23:59:59.

Service	Real log	Prophet log	Telescope log
studium	1231485	1367028	1184431
predmety	618280	498638	437050
term_st2	616488	789187	762270
prijimacky	552242	869029	219656
rozvrhng	271023	257326	204011
dipl_st	166770	130201	109325
zkous_st	123994	130739	120702
soub_mana	115514	36253	31350
zkous_uc2	108605	113570	107166
kdojekdo	88844	72085	67518
term_uc2	54263	60148	56819
nastenka	44129	44058	35697
dipl_uc	43860	54423	52171
anketa	43451	33703	27039
omne	42967	42657	39306
cml	33478	68350	66033
szz_uc	31634	34134	30616
predm_st2	27851	36147	34178
szz_st	24430	40833	38753
ekczv	18095	19957	15647
ciselniky	17997	4209	3396
szz	13948	14159	11400
komise	9663	20933	20918
phdisp	8446	9874	9041
esc	8205	9186	8359
grupicek	5852	9818	9143
akreditace_rvh	5511	4147	3951
promoce	4899	4549	3811

Table A.2.: Request counts of services (1)

Service	Real log	Prophet log	Telescope log
skolitel	4865	3468	2612
wstip_uc	4130	4293	3996
predm_uc	3990	4338	4210
podprij	3546	3796	2909
wstip_st	3062	2913	2602
grupik	1880	2474	2564
harmonogram	1866	977	790
role	1824	2078	1744
diplmat	1457	111	92
sestavy	1317	1614	1264
vyspl	1306	1393	786
deda_amu	1169	482	381
staze_uc	1121	319	268
prezkumy_st	1089	884	781
deda_zahost	958	1324	1208
transcript	614	1069	1030
uchak	547	5886	4709
ave	538	829	874
ave_uziv	504	77	111
rozpis	167	142	143
deda_strav	165	629	732
bookmarks	107	154	176

Table A.3.: Request counts of services (2)

A.3. Order of Services based on Request Counts

Table A.4 and Table A.5 list the order of services based on their request counts. Real log (1) contains the recorded data from 28/05/2018 00:00:00 until 17/06/2018 23:59:59, and real log (2) the recorded data from 18/06/2018 00:00:00 until 24/06/2018 23:59:59 from the SIS logs.

Real log (1)	Real log (2)	Prophet log	Telescope log
studium	studium	studium	studium
term_st2	predmety	prijimacky	term_st2
prijimacky	term_st2	term_st2	predmety
predmety	prijimacky	predmety	prijimacky
rozvrhng	rozvrhng	rozvrhng	rozvrhng
zkous_st	dipl_st	zkous_st	zkous_st
zkous_uc2	zkous_st	dipl_st	dipl_st
dipl_st	soub_mana	zkous_uc2	zkous_uc2
kdojekdo	zkous_uc2	kdojekdo	kdojekdo
term_uc2	kdojekdo	cml	cml
cml	term_uc2	term_uc2	term_uc2
dipl_uc	nastenka	dipl_uc	dipl_uc
szz_st	dipl_uc	nastenka	omne
omne	anketa	omne	szz_st
nastenka	omne	szz_st	nastenka
predm_st2	cml	soub_mana	predm_st2
soub_mana	szz_uc	predm_st2	soub_mana
szz_uc	predm_st2	szz_uc	szz_uc
anketa	szz_st	anketa	anketa
komise	ekczv	komise	komise
ekczv	ciselniky	ekczv	ekczv
szz	szz	szz	szz
grupicek	komise	phdisp	grupicek
phdisp	phdisp	grupicek	phdisp
esc	esc	esc	esc
predm_uc	grupicek	uchak	uchak
promoce	akreditace_rvh	promoce	predm_uc

Table A.4.: Order of requested services in each log (1)

Real log (1)	Real log (2)	Prophet log	Telescope log
uchak	promoce	predm_uc	wstip_uc
akreditace_rvh	skolitel	wstip_uc	akreditace_rvh
wstip_uc	wstip_uc	ciselniky	promoce
podprij	predm_uc	akreditace_rvh	ciselniky
ciselniky	podprij	podprij	podprij
wstip_st	wstip_st	skolitel	skolitel
skolitel	grupik	wstip_st	wstip_st
grupik	harmonogram	grupik	grupik
role	role	role	role
sestavy	diplmat	sestavy	sestavy
vyspl	sestavy	vyspl	deda_zahost
deda_zahost	vyspl	deda_zahost	transcript
harmonogram	deda_amu	transcript	ave
prezkumy_st	staze_uc	harmonogram	harmonogram
diplmat	prezkumy_st	prezkumy_st	vyspl
ave	deda_zahost	ave	prezkumy_st
transcript	transcript	deda_strav	deda_strav
deda_strav	uchak	deda_amu	deda_amu
ave_uziv	ave	staze_uc	staze_uc
deda_amu	ave_uziv	bookmarks	bookmarks
rozpis	rozpis	rozpis	rozpis
staze_uc	deda_strav	diplmat	ave_uziv
bookmarks	bookmarks	ave_uziv	diplmat

Table A.5.: Order of requested services in each log (2)

Appendix A

Bibliography

[19a] *Introducing JSON*. 2019. URL: https://www.json.org/ (cit. on pp. 15, 53).

[19b] *YAML 1.2*. 2019. URL: http://yaml.org/ (cit. on pp. 15, 37).

[AATP12] F. Abbors, T. Ahmad, D. Truscan, I. Porres. "MBPeT: a model-based performance testing tool." In: *2012 Fourth International Conference on Advances in System Testing and Validation Lifecycle*. 2012 (cit. on p. 20).

[AJ19] Apache Software Foundation. *Apache JMeter*. 2019. URL: https://jmeter.apache.org/ (cit. on pp. 10, 100).

[ASSH16] H. M. AlGhmadi, M. D. Syer, W. Shang, A. E. Hassan. "An automated approach for recommending when to stop performance tests." In: *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2016, pp. 279–289 (cit. on p. 63).

[BGR18] BGR. *Amazon's website went down just as Prime Day 2018 kicked off*. 2018. URL: https://bgr.com/2018/07/16/amazon-website-down-prime-day-2018-best-deals/ (cit. on p. 33).

[Bib19] Bibliographisches Institut GmbH. *Duden*. 2019. URL: https://www.duden.de/ (cit. on p. 33).

[Bla17] BlazeMeter. *JMeter Ramp-Up - The Ultimate Guide*. 2017. URL: https://www.blazemeter.com/blog/jmeter-ramp-up-the-ultimate-guide (cit. on p. 100).

[Bla19] BlazeMeter. *YAML Tutorial*. 2019. URL: https://gettaurus.org/docs/YAMLTutorial/ (cit. on p. 15).

[Bos16] J. Bosch. *Continuous Software Engineering*. Springer, 2016 (cit. on p. 1).

[BRA16] BRADLEYWEBGROUP. *Does your Website Traffic go down at the weekend?* 2016. URL: https://bradleywebgroup.com/website-traffic-go-weekend/ (cit. on p. 32).

[Bre19] A. Brenecki. *Learn X in Y minutes*. 2019. URL: https://learnxinyminutes.com/docs/yaml/ (cit. on p. 15).

[CG16] T. Chen, C. Guestrin. "Xgboost: A scalable tree boosting system." In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM. 2016, pp. 785–794 (cit. on p. 11).

[CGD18] W. Cappelli, S. Ganguli, F. De Silva. "Magic quadrant for application performance monitoring." In: *Gartner, March* (2018) (cit. on p. 2).

[Cha19a] Charles University. *Charles University*. 2019. URL: https://www.cuni.cz/UKEN-1.html (cit. on p. 75).

[Cha19b] Charles University. *Student Information System*. 2019. URL: https://is.cuni.cz/studium/eng/ (cit. on p. 75).

[CP19] ContinuITy-Project. *ContinuITy*. 2019. URL: https://github.com/ContinuITy-Project/ContinuITy (cit. on pp. 65–67).

[CS93] M. Calzarossa, G. Serazzi. "Workload characterization: A survey." In: *Proceedings of the IEEE* 81.8 (1993), pp. 1136–1150 (cit. on p. 9).

[DD98] C. J. Date, H. Darwen. "The SQL standard." In: *SQL/92 mit den Erweiterungen CLI und PSM* (1998) (cit. on p. 14).

[DGH+06] D. Draheim, J. Grundy, J. Hosking, C. Lutteroth, G. Weber. "Realistic load testing of web applications." In: *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*. IEEE. 2006, 11–pp (cit. on p. 20).

[EUS18] EUSurvey. *Public consultation on summertime arrangements*. 2018. URL: https://ec.europa.eu/eusurvey/runner/2018-summertime-arrangements (cit. on p. 33).

[fac17] facebook research. *Prophet: forecasting at scale*. 2017. URL: https://research.fb.com/prophet-forecasting-at-scale/ (cit. on pp. 12, 26, 71).

[fac19] facebook. *PROPHET*. 2019. URL: https://facebook.github.io/prophet/docs (cit. on pp. 13, 102).

[Fas18] FasterXML. *Jackson Project Home*. 2018. URL: https://github.com/FasterXML/jackson (cit. on p. 71).

[FGL15] A. Filieri, L. Grunske, A. Leva. "Lightweight adaptive filtering for efficient learning and updating of probabilistic models." In: *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press. 2015, pp. 200–211 (cit. on pp. 29, 75).

[Gee12] Geek Wire. *Tons of traffic: Amazon dominates online retail during Christmas week*. 2012. URL: https://www.geekwire.com/2012/christmas-day-2012-retail-visits-increase-27-compared-2011/ (cit. on p. 32).

[GMMO00] S. Guha, N. Mishra, R. Motwani, L. O'Callaghan. "Clustering data streams." In: *Foundations of computer science, 2000. proceedings. 41st annual symposium on*. IEEE. 2000, pp. 359–366 (cit. on p. 108).

[GSML06] K. Goševa-Popstojanova, A. D. Singh, S. Mazimdar, F. Li. "Empirical characterization of session–based workload and reliability for web servers." In: *Empirical Software Engineering* 11.1 (2006), pp. 71–117 (cit. on p. 8).

[Har75] J. A. Hartigan. "Clustering algorithms." In: (1975) (cit. on p. 11).

[HF10] J. Humble, D. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education, 2010 (cit. on pp. 1, 7).

[HH19] W. Hasselbring, A. van Hoorn. *Kieker*. 2019. URL: http://kieker-monitoring.net/ (cit. on p. 9).

[HHKA14] N. R. Herbst, N. Huber, S. Kounev, E. Amrehn. "Self-adaptive workload classification and forecasting for proactive resource provisioning." In: *Concurrency and computation: practice and experience* 26.12 (2014), pp. 2053–2078 (cit. on pp. 2, 21, 22).

[Hid19] A. Hidiroglu. *Supplementary Material Master's Thesis "Context-Aware Load Testing in Continuous Software Engineering"*. 2019. URL: https://doi.org/10.5281/zenodo.2538268 (cit. on pp. 75, 82).

[HK+07] R. J. Hyndman, Y. Khandakar, et al. *Automatic time series for forecasting: the forecast package for R*. 6/07. Monash University, Department of Econometrics and Business Statistics, 2007 (cit. on p. 11).

[Inf19a] InfluxData. *Influx Query Language (InfluxQL)*. 2019. URL: https://docs.influxdata.com/influxdb/v1.7/query_language/ (cit. on p. 35).

[Inf19b] InfluxData. *InfluxDB*. 2019. URL: https://www.influxdata.com/time-series-platform/influxdb/ (cit. on pp. 14, 71).

[Inf19c] InfluxData. *InfluxDB 1.7 documentation*. 2019. URL: https://docs.influxdata.com/influxdb/v1.7/ (cit. on p. 14).

[Inf19d] InfluxData. *Time Series Database (TSDB) Explained*. 2019. URL: https://www.influxdata.com/time-series-database/ (cit. on p. 14).

[Int17] International Business Times. *Nike Site Down On Black Friday, How To Place An Order*. 2017. URL: https://www.ibtimes.com/nike-site-down-black-friday-how-place-order-2619427 (cit. on p. 32).

[JH15] Z. M. Jiang, A. E. Hassan. “A survey on load testing of large-scale software systems.” In: *IEEE Transactions on Software Engineering* 41.11 (2015), pp. 1091–1118 (cit. on pp. 1, 7).

[Joh13] S. Johnston. *Release Management Best Practices at Amazon*. puppet, 2013. URL: https://puppet.com/blog/release-management-best-practices-at-amazon (cit. on p. 1).

[JSB97] D. F. Jerding, J. T. Stasko, T. Ball. “Visualizing interactions in program executions.” In: *Proceedings of the 19th international conference on Software engineering*. ACM. 1997, pp. 360–370 (cit. on p. 9).

[KHK14] J. v. Kistowski, N. R. Herbst, S. Kounev. “Modeling variations in load intensity over time.” In: *Proceedings of the third international workshop on Large scale testing*. ACM. 2014, pp. 1–4 (cit. on pp. 19, 21).

[KRM06] D. Krishnamurthy, J. A. Rolia, S. Majumdar. “A synthetic workload generation technique for stress testing session-based systems.” In: *IEEE Transactions on Software Engineering* 32.11 (2006), pp. 868–882 (cit. on pp. 8, 19, 20).

[LT03] Z. Li, J. Tian. “Testing the suitability of Markov chains as Web usage models.” In: *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*. IEEE. 2003, pp. 356–361 (cit. on pp. 8, 19).

[LW08] C. Lutteroth, G. Weber. “Modeling a realistic workload for performance testing.” In: *Enterprise Distributed Object Computing Conference, 2008. EDOC’08. 12th International IEEE*. IEEE. 2008, pp. 149–158 (cit. on p. 20).

[MA02] D. A. Menasce, V. A. Almeida. *Capacity Planning for Web Services: metrics, models, and methods*. Prentice Hall PTR Upper Saddle River, NJ, 2002 (cit. on pp. 8, 9).

[MAFM99] D. A. Menascé, V. A. Almeida, R. Fonseca, M. A. Mendes. “A methodology for workload characterization of e-commerce sites.” In: *Proceedings of the 1st ACM conference on Electronic commerce*. ACM. 1999, pp. 119–128 (cit. on pp. 8, 19, 20, 28).

[Mic19] Microsoft. *Windows Server 2016*. 2019. URL: https://www.microsoft.com/de-de/licensing/product-licensing/windows-server-2016.aspx (cit. on p. 80).

[Mit10] T. Mitsa. *Temporal data mining*. Chapman and Hall/CRC, 2010 (cit. on p. 14).

[Mus93] J. D. Musa. “Operational profiles in software-reliability engineering.” In: *IEEE software* 10.2 (1993), pp. 14–32 (cit. on p. 8).

[new14] news.com.au. *Ellen DeGeneres selfie at Oscars 2014 breaks Twitter*. 2014. URL: https://www.news.com.au/technology/online/ellen-degeneres-selfie-at-oscars-2014-breaks-twitter/news-story/da9c8e1fe7d8e73ed791143872f10bbe (cit. on p. 32).

[NTC19] NovaTec Consulting GmbH. *inspectIT*. 2019. URL: http://www.inspectit.rocks/ (cit. on pp. 9, 68).

[OHH+16] D. Okanović, A. van Hoorn, C. Heger, A. Wert, S. Siegl. "Towards performance tooling interoperability: An open format for representing execution traces." In: *European Workshop on Performance Engineering*. Springer. 2016, pp. 94–108 (cit. on p. 68).

[Ora19] Oracle. *Java*. 2019. URL: https://www.java.com/en/ (cit. on pp. 15, 71).

[Piv19] Pivotal. *RabbitMQ*. 2019. URL: https://www.rabbitmq.com/ (cit. on p. 65).

[PM+00] D. Pelleg, A. W. Moore, et al. "X-means: Extending k-means with efficient estimation of the number of clusters." In: *Icml*. Vol. 1. 2000, pp. 727–734 (cit. on p. 45).

[Pyt19] Python Software Foundation. *python*. 2019. URL: https://www.python.org/ (cit. on p. 13).

[Rad18] G. Rademacher. *Railroad Diagram Generator*. 2018. URL: http://www.bottlecaps.de/rr/ui (cit. on p. 37).

[RDG11] N. Roy, A. Dubey, A. Gokhale. "Efficient autoscaling in the cloud using predictive models for workload forecasting." In: *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE. 2011, pp. 500–507 (cit. on pp. 2, 22).

[RFo19] RForge. *JRI - Java/R Interface*. 2019. URL: https://www.rforge.net/JRI/ (cit. on p. 71).

[RK18] D. G. Reichelt, S. Kühne. "How to Detect Performance Changes in Software History: Performance Analysis of Software System Versions." In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. ACM. 2018, pp. 183–188 (cit. on p. 18).

[RTL19] RTL.de. *Wer wird Millionär?* 2019. URL: https://www.rtl.de/cms/sendungen/wer-wird-millionaer.html (cit. on p. 33).

[SAH18] H. Schulz, T. Angerstein, A. van Hoorn. "Towards Automating Representative Load Testing in Continuous Software Engineering." In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018*. 2018, pp. 123–126 (cit. on pp. 1, 9).

[SGMM17] H. Spieker, A. Gotlieb, D. Marijan, M. Mossige. “Reinforcement learning for automatic test case prioritization and selection in continuous integration.” In: *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM. 2017, pp. 12–22 (cit. on p. 17).

[SKF06] M. Shams, D. Krishnamurthy, B. Far. “A model-based approach for testing the performance of web applications.” In: *Proceedings of the 3rd international workshop on Software quality assurance*. ACM. 2006, pp. 54–61 (cit. on p. 20).

[ST02] A. Srivastava, J. Thiagarajan. “Effectively prioritizing tests in development environment.” In: *ACM SIGSOFT Software Engineering Notes*. Vol. 27. 4. ACM. 2002, pp. 97–106 (cit. on p. 18).

[Sta96] Standard, EBNF Syntax Specification. “Ebnf: Iso/iec 14977: 1996 (e).” In: *URL https://www.cl.cam.ac.uk/ mgk25/iso-14977.pdf* 70 (1996) (cit. on p. 37).

[t3n18] t3n. *Vor Microsoft-Übernahme: Software-Projekte verlassen Github in Scharen*. 2018. URL: https://t3n.de/news/ms-projekte-verlassen-github-1084562/ (cit. on p. 33).

[The18] The Apache Software Foundation. *Commons Lang*. 2018. URL: https://commons.apache.org/proper/commons-lang/ (cit. on p. 71).

[The19a] The Apache Software Foundation. *Apache HTTP Server Project*. 2019. URL: https://httpd.apache.org/ (cit. on p. 75).

[The19b] The R Foundation. *The R Project for Statistical Computing*. 2019. URL: https://www.r-project.org/ (cit. on pp. 13, 71).

[TL18] S. J. Taylor, B. Letham. “Forecasting at scale.” In: *The American Statistician* 72.1 (2018), pp. 37–45 (cit. on pp. 12, 13, 51).

[TTP06] TechTarget. *Performance*. 2006. URL: https://whatis.techtarget.com/definition/performance (cit. on p. 8).

[Uni18] Universität Würzburg. *Telescope*. 2018. URL: http://descartes.tools/telescope (cit. on pp. 11, 26, 71).

[Uni19] Univocity. *Welcome to univocity-parsers*. 2019. URL: https://www.univocity.com/pages/univocity_parsers_tutorial (cit. on p. 71).

[USA16] USA TODAY. *Netflix down for about 2.5 hours Saturday*. 2016. URL: https://eu.usatoday.com/story/tech/news/2016/10/01/netflix-goes-down-saturday-afternoon/91396200/ (cit. on p. 32).

[VHS+18] C. Vögele, A. van Hoorn, E. Schulz, W. Hasselbring, H. Krcmar. "WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction - a model-driven approach for session-based application systems." In: *Software and System Modeling* 17.2 (2018), pp. 443–477 (cit. on pp. 8, 9, 19, 20, 28, 42, 45, 71).

[W3C17] W3C. *XQuery 3.1: An XML Query Language*. 2017. URL: https://www.w3.org/TR/2017/REC-xquery-31-20170321/#EBNFNotation (cit. on p. 37).

[Wea14] WeatherAds. *Weather and eCommerce: How Weather Impacts Retail Website Traffic and Online Sales*. 2014. URL: http://www.weatherads.io/blog/2014/august/weather-and-ecommerce-how-weather-impacts-retail-website-traffic-and-online-sales (cit. on p. 32).

[WIR18] WIRED. *GitHub Survived the Biggest DDoS Attack Ever Recorded*. 2018. URL: https://www.wired.com/story/github-ddos-memcached/ (cit. on p. 32).

[ZBH+17] M. Züfle, A. Bauer, N. Herbst, V. Curtef, S. Kounev. "Telescope: A Hybrid Forecast Method for Univariate Time Series." In: (Sept. 2017) (cit. on pp. 11, 12, 51).

[Züf17] M. Züfle. *Dynamic Hybrid Forecasting for Self-Aware Systems*. Master Thesis. Würzburg, Germany: Department of Computer Science, Universität Würzburg, 2017 (cit. on p. 11).

[ZZL14] J. Zhou, B. Zhou, S. Li. "LTF: A model-based load testing framework for web applications." In: *Quality Software (QSIC), 2014 14th International Conference on*. IEEE. 2014, pp. 154–163 (cit. on p. 21).

All links were last followed on January 15, 2019.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature