Context-tailored Workload Model Generation for Continuous Representative Load Testing

Henning Schulz
Dušan Okanović
Novatec Consulting GmbH, Germany

André van Hoorn
University of Stuttgart, Germany

Petr Tůma
Charles University, Czech Republic

ABSTRACT

Load tests evaluate software quality attributes, such as performance and reliability, by e.g., emulating user behavior that is representative of the production workload. Existing approaches extract workload models from recorded user requests. However, a single workload model cannot reflect the complex and evolving workload of today's applications, or take into account workload-influencing contexts, such as special offers, incidents, or weather conditions. In this paper, we propose an integrated framework for generating load tests tailored to the context of interest, which a user can describe in a language we provide. The framework applies multivariate time series forecasting for extracting a context-tailored load test from an initial workload model, which is incrementally learned by clustering user sessions recorded in production and enriched with relevant context information.

We evaluated our approach with the workload of a student information system. Our results show that incrementally learned workload models can be used for generating tailored load tests. The description language is able to express the relevant contexts, which, in turn, improve the representativeness of the load tests. We have also found that the existing workload characterization concepts and forecasting tools used are limited in regard to strong workload fluctuations, which needs to be tackled in future work.

ACM Reference Format:
Henning Schulz, Dušan Okanović, André van Hoorn, and Petr Tůma. 2021. Context-tailored Workload Model Generation for Continuous Representative Load Testing. In *Proceedings of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3427921.3450240

1 INTRODUCTION

The workloads of modern session-based [31] applications vary significantly [21]. This variation needs to be considered when load-testing [24] the application. Known from detecting contextual anomalies [10], contexts such as special offers, incidents, or weather conditions can influence the workload — also known as operational profile [34] —, resulting in different workload scenarios. For instance, while the normal workload of a webshop might comprise 1000 concurrent users of a certain mix, a special offer could cause a spike of up to 5000 users of a different mix. As contexts influence not only the number of users but also the workload mix, there may not be a single most demanding scenario to test. However, the fast release cycles of continuous software engineering (CSE) [22] contradict a testing approach that covers all scenarios that have already been observed or might occur in the future [5, 48].

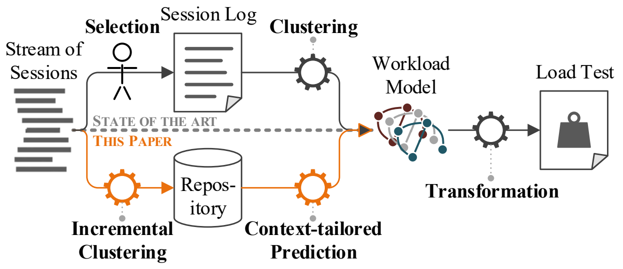


Figure 1: State-of-the-art load test generation methodology and its extension presented in this paper.

As illustrated in Fig. 1 (top part), the existing approaches [3, 8, 25, 28, 30, 42, 57] can generate a load test representing a workload scenario that occurred in the past. The workload scenario is determined by a human, who selects a finite subset of the continuous stream of sessions arriving at the production system. By clustering the sessions, the approaches extract a workload model — e.g., based on Markov chains — that represents the behavior of different user groups with a particular mix. A drawback of such generated load tests is that they only reconstruct past workload scenarios. Most workloads follow a global trend and seasonal variations [21], which need to be integrated into the load test. For that, approaches for time series modeling and forecasting [4, 21, 54, 58] could be used, which, however, mainly focus on the overall intensity, disregarding a varying workload mix. Also, they lack support for selecting the relevant scenarios and incorporating qualitative forecasting [30], e.g., for estimating unseen future scenarios. Finally, approaches used in CSE need to be automated and should abstract the technical details, such as clustering and forecasting, from the user [5].

Therefore, in this paper, we introduce an integrated, end-to-end framework for automatically generating load tests tailored to the contexts a user specifies. The workloads simulated in the load tests are predicted automatically, based on the provided context information. Thus, users can focus on the scenarios relevant to them and save the time needed for testing less relevant scenarios. As an example, the operator of a webshop will be interested in the Black Friday spike in early November, but the same spike is less relevant in December. Also, the user can add external knowledge as a qualitative forecast, e.g., planned changes to the platform.

We build upon the WESSBAS approach [57], which is based on the session-based workload characterization methodology by Menascé et al. [31], and add two extensions summarized in Fig. 1 (bottom part). First, to scale with the constantly increasing number of sessions, we cluster them incrementally and store the resulting clusters and their mix in a repository. Second, for predicting the future behavior of these clusters in a user-defined context, we leverage existing time series forecasting approaches, including Telescope [4] and Prophet [54]. For describing the context-tailored workload scenario to be predicted, we introduce the Load Test Context-tailoring Language (LCtL). LCtL instances serve as input to the forecasting approach, specify how to extract a workload scenario from a forecast or past data, and allow changing the scenario for qualitative forecasting. For integration with CSE, the generation of a context-tailored load test based on an LCtL description is fully automated.

We evaluated the framework with the student information system (SIS) of Charles University, Prague, regarding LCtL's expressiveness, the learned workload model's ability to predict future workloads, and the representativeness of workload models with forecasted intensities. We found that LCtL is suitably expressive for real-world scenarios and contexts. Also, we assessed the framework to be suited for generating representative context-tailored load tests. Predicted workload scenarios are particularly representative when considering the user groups separately and enriching the forecasts with contexts. However, we also identified limitations of the existing workload characterization and forecasting approaches, such as the handling of session timings and predictions of sharp spikes.

Our results pose the following challenges to be addressed in future work. First, the workload characterization methodology by Menascé et al. [31] and Vögele et al. [57] needs to be extended to handle fluctuating inter-request think times appropriately. While the methodology has been extensively evaluated to extract highly representative workload models from a single bunch of recorded user sessions, we show it impairs the representativeness of models incrementally learned from a large time frame. Second, time-series forecasting approaches, such as Telescope [4] and Prophet [54], can accurately predict steady-state future workloads but lack the prediction of sharp variations, including load spikes. To evaluate the effectiveness of improved approaches, which can smoothly be integrated into our framework, we provide supplementary material with our evaluation data and replication instructions online [51].

To summarize, our contribution extends state-of-the-art load test generation approaches to address the specific requirements of CSE in an end-to-end manner through these technical steps:

- we extend the workload clustering from [57] to support an incremental construction and maintenance of the workload model,
- we define the LCtL language for describing the context that the generated load test is to approximate, and
- we combine the context information with workload forecasting to generate models for contexts that were not previously observed.

The paper is structured as follows. Section 2 defines fundamental terms and provides examples, followed by the related work in Section 3. In Section 4, we introduce our framework, and in Section 5, we provide its evaluation. Section 6 concludes the paper.

2 DEFINITIONS AND EXAMPLES

A fundamental concept of our work is the context, which we presume to influence the workload. We define the *workload context* as a set of *workload context facets*. A workload context facet is a self-contained circumstance present to a significant number of users of an application with a well-defined state at each point in time.

A context-tailored load test aims at replaying a particular *workload scenario*, which we define as a workload segment of a fixed length with specific characteristics to be replayed in a load test, e.g., the constant workload of a specific mix or a workload spike. Workload scenarios can have occurred in the past, be expected in the future, or be hypothetical, e.g., for what-if analyses. The context influences a workload scenario.

We have collected multiple anecdotal examples of contexts that have influenced an application's workload. Particularly reported in news articles and blog posts are influences that caused high workload, which, in turn, lead to downtimes. The examples listed below include seven different context facets.

Some context facets *recur*, i.e., their state and influence on the workload are predictable from the past occurrences. These facets include high workload due to *special offers* (Christmas) [53]; high workload and downtime due to combinations of *releases of new products*, *weather conditions*, and *special offers* (TV series and rainy [60]/sneakers and Black Friday [19]/Prime Day [32]); varying workloads depending on the *weekday and holidays* [7]; and increased focus on a particular subject influenced by *current events* [62].

Similar are *continuous* facets, such as *weather conditions* [59, 60]. While their influence on the workload is typically predicable, the predictions themselves are often inaccurate and of limited horizon.

The future impact of *irregular* facets, in contrast, is unpredictable. An example is a *special incident* (recovery from message endpoint outage) leading to workload spikes [50]. Still, past occurrences can be used for predicting the impact on future workload at a hypothetical date, for conducting a what-if analysis.

Finally, there are *singleton* facets, which only occur once. As there are no past occurrences, quantitative methods cannot predict their influence on the workload. However, at least for some facets, e.g., [50], the future state is known and the influence can be added by manual estimation as a qualitative forecast [30]. Here, examples include increased workload due to planned *changes to the platform* (new devices added) [50], high workload and downtime due to a *special incident* (prominent posting) [6], or high workload due to a *special incident* (acquisition of competitor) [39].

Besides the differences in predictability, contexts also differ in the type of workload scenario. While some cause increased but generally steady workload [53], others entail a workload spike [50]. A particularly interesting case is [62], where the context influenced the workload mix. As a consequence, there is no most-demanding workload we could use in a load test for covering all scenarios. Instead, we need load tests tailored to those contexts and scenarios that are currently relevant.

3 RELATED WORK

Load Testing in DevOps. DevOps [64] poses a significant challenge to applying traditional load testing practices [24, 31], due to the organizational changes and increased velocity of software delivery. The integration of load testing in such an environment requires special attention [5, 48]. DevOps teams are in charge of developing, testing, and deploying their own microservices [35]. Load testing assumes testing the entire system under a realistic workload. In previous work [46], we developed an approach to generate load tests tailored to particular microservices.

Delivery in microservice-based software systems is more frequent and fast. Hence, it leaves very little time for the execution of time-consuming tasks such as load tests. Daly et al. [14] propose a change point detection approach to deal with a number of tests and analyzing their results.

In this paper, we aim to optimize the load test execution by selecting those tests that are relevant to the next delivery.

Test Case Selection/Prioritization. The testing of complex systems can be a challenge in terms of time and effort. To deal with increasing costs, many approaches have been developed for test selection and prioritization. The goal is to select and run tests that are most likely to identify errors. This is a well known technique in functional testing, as surveyed by Yoo and Harman [63]. For software performance tests, it is even more important, since they sometimes have to be run for a longer time, for performance issues to manifest themselves. For example, Ferme et al. [18] use test selection for obtaining fast performance feedback. Most of the work in the field, however, is oriented on regression testing. Hashemian et al. [20] present the IRIS approach, which, considering the testing budget, maximises the performance insights. The PerfRanker approach [33] is based on estimating the impact of the code added since the last revision. Reichelt et al. [41] propose to use code and trace analysis for test selection in order to reduce the time required to run tests.

In this approach, we select and generate load tests, which are tailored to specific future contexts, based on the historical workload characteristics and workload-influencing contexts.

Workload Characterization. Workload characterization deals with the extraction and representation of the workload from a production system [9, 31]. There are different formalisms for characterizing a workload, for example based on state-based models [31, 57] and time series [58]. These models can be used to generate and execute representative load tests [24].

In this paper, we extend the WESSBAS [57] approach for characterizing and forecasting workloads divided into user groups, by including workload-influencing contexts.

Workload Forecasting. An intrinsic property of the workload characterization approaches above is that they deal only with information from the past. To test the system with a future workload, we have to apply a suitable forecasting approach to predict how the workload will evolve over time.

There are different approaches that deal with quantitative workload forecasting, such as [21, 30], with many of them based on time series forecasting. These approaches predict only how the workload of a system will change based on the currently available, i.e., past data. They do not take into account potential future workload-influencing contexts. In our work, we rely on existing forecasting techniques, particularly Telescope [4] and Prophet [54]. With the inclusion of workload-influencing contexts, we also support qualitative workload forecasting [30].

Performance Testing Languages. The specification of load tests requires expertise and experience, as well as the knowledge of tools. Hence, researchers have developed approaches to ease the specification based on domain-specific and natural languages. As a part of the BenchFlow [17] approach, users can specify different types of load tests using the BenchFlow domain-specific language (DSL), without having to know the details of particular tools, which will be used to run them. In previous work [50], we proposed the Behavior-driven Load Testing (BDLT) language, which leverages BenchFlow and preliminary concepts from this work to reduce the effort and expertise required to specify and execute load tests. Furthermore, we introduced an annotation language to reduce the effort to maintain load tests and automate their generation [52]. An approach by Okanović et al. [37] allows the user to set up simple load tests based on their concerns using a natural language and obtain results tailored specifically to these concerns.

In this paper, we propose the LCtL language, to ease the task of specifying context-tailored load tests. Compared to the BDLT language, it does not include test execution but is more extensive regarding the tailored specification of the load test scenario, with the support of an automated framework to generate the test. For automation, the framework leverages our annotation language.

4 CONTEXT-TAILORED LOAD TEST GENERATION

As discussed in Section 2, the goal of our work is to generate load tests tailored to the currently relevant contexts and workload scenarios. For that, we introduce a framework that leverages workload model extraction and forecasting techniques for generating a context-tailored load test from a user's description. We first provide an overview in Section 4.1 and describe the details in the subsequent sections.

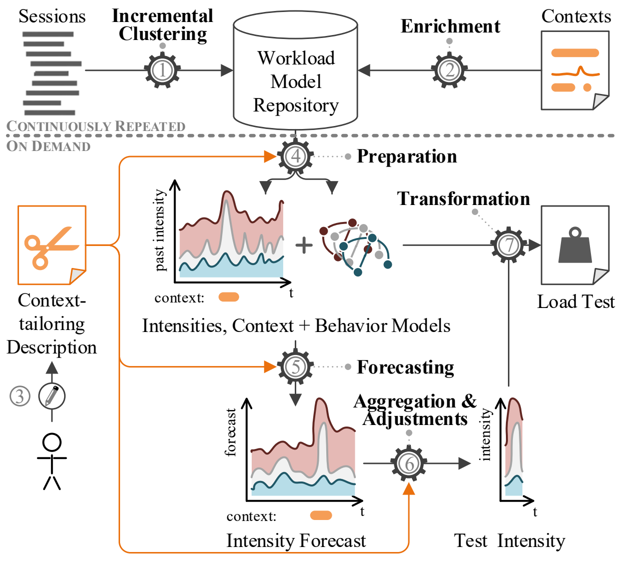


Figure 2: Framework overview.

4.1 Overview of the Approach

As Fig. 2 illustrates, our framework consists of two parts: one that is continuously repeated, and one that a user can trigger on-demand. In the former part, we collect all the information required for generating a load test. For that, we continuously collect the user sessions by monitoring the production system and cluster the stream of sessions incrementally ①. We obtain Markov chains representing the behavior of a particular type of user and the corresponding numbers of concurrent users (workload intensities [57]) over time, and store them into the workload model repository (WMR) for later use. Besides, we enrich the intensities with relevant contexts ②.

For generating a new load test, a user defines a tailoring description using our provided language ③. The description specifies the purpose of the load test and serves as an input for the load test generation process. This process consists of four steps. First, we prepare the workload and context, i.e., select all relevant data from the WMR and potentially adjust it based on the tailoring description ④. While the future context is known, the future values of the workload intensities are to be predicted. For that, we utilize time series forecasting tools, considering the context and tailoring description ⑤. Next, we aggregate the forecasted intensities and potentially adjust them based on the tailoring description ⑥. As a result, we obtain user groups represented by Markov chains plus the corresponding relative frequencies and intensities.

The final step is to transform the listed artifacts into an executable load test ⑦, which will replay the workload that these artifacts describe. Regarding this transformation, we refer to our previous work [52]. It automates the transformation, allowing for a fully automated processing of the user's request.

Our approach is implemented as part of the ContinuITy project [36, 48], with the source code being available online [44, 47, 49]. The technologies used are the following: Java for the major part of the approach, Python with its powerful scikit-learn [38] library for session clustering, R [40] for workload forecasting, and Elasticsearch [16] for the WMR. All data in the WMR is formatted in JSON.

4.2 Continuous Workload Model Learning

4.2.1 Incremental Clustering. Our approach builds on the workload model of WESSBAS [57], which targets closed workloads with sessions. Each endpoint accessed by a workload, e.g., the HTTP endpoints */login* and */home*, is modeled as one state in a Markov-chain-based representation. The sequences of requests that make up each session are modeled as transitions in the Markov chain, yielding one transition matrix per session. Similar sessions are grouped by clustering the matrices with k-means [31] or X-means [57]. After clustering, the centroids of the session clusters represent groups of users with similar behavior, and are transformed into Markov-chain-based behavior models by converting the absolute transition frequencies to probabilities. The think times — i.e., the times the users wait between submitting two requests — are computed per model and transition. Finally, the resulting *workload model* comprises the individual behavior models and their relative frequencies (mix).

As outlined, the existing approaches are not designed for updating an already-extracted workload model with newer sessions, yet we require exactly this feature for building the WMR. With a growing history of system execution typical for CSE, running the clustering algorithm repeatedly from scratch does not scale (the problem itself is known to be NP-hard, the specific complexity depends on additional details). Also, the clusters discovered by k-means or X-means are not necessarily stable, which can be a problem if the developers, e.g., use the once-discovered clusters in context specification (as we do in Section 5.3). Therefore, we modify the existing clustering algorithms to enable incremental updates in batches, preserving the once-discovered clusters and keeping the computational complexity of each update manageable.

The exact implementation of the incremental clustering algorithm is available in [49]. The algorithm distinguishes between the first and further batches. In the first batch, we move the outliers (based on distance to nearest neighbor) into a separate noise cluster and then apply k-means++ [2] to cluster the remaining sessions, obtaining $k + 1$ clusters with their centroids and radiuses. For further batches, we first assign all sessions that are within the radius of some existing cluster, with a *tolerance factor* $\beta \geq 1$, to that cluster. We then loosely base upon Lloyd's Algorithm [27] to form a cluster of the remaining sessions, and finally add any still remaining sessions to the noise cluster.

After each clustering iteration, our approach calculates the *workload intensities* for each cluster and stores them in the WMR. The intensity is defined as the number of concurrent users in a configured interval, e.g., one minute.

4.2.2 Enrichment with Contexts. Users can enrich the learned intensities with workload contexts by providing past observations and known future states of the relevant context facets. These states are also stored in the WMR. This has two benefits: (1) a user can describe a time range from which they want to extract a load test based on the context facets' states; (2) our approach can use the future facet states for improving the intensity forecast.

The types of context facets we are interested in are those whose state or influence is predictable. Other facets cannot be used to describe the time range or improve the forecast. We store at least the past states we have observed; for those with a predictable state, we also store the future states. Furthermore, we provide a generic API endpoint that can be used for synchronizing with various data sources, e.g., calendars for sales events, corresponding platforms for weather forecasts, or application monitoring tools for platform incidents.

For technology-independent exchange, e.g., via Representational State Transfer (REST), we store the context facets in records in the JSON format, as shown in Listing 1. Context facets can be numeric, string, or boolean, and the values are stored in one record per point in time. For instance, the Black Friday is an event that either occurs or not; hence, it is a boolean facet. For product releases, one might differentiate between the products, which can be encoded as strings. The temperature is a numeric facet.

4.3 Load Test Context-tailoring Language

When a user wants to extract a load test from the WMR, they need to describe the workload scenario the test will simulate. For that, we provide the Load Test Context-tailoring Language (LCtL), which uses the YAML format [56] for viable integration with CSE. YAML

Listing 1: Workload context record example.

```
{
  "boolean": [
    "black_friday"
  ],
  "string": {
    "product_release":
      "sneakers"
  },
  "numeric": {
    "temperature": 21
  }
}
```

Listing 2: LCtL example.

```
timeframe:
- !<conditional>
  product_release:
    is: sneakers
context:
  weather:
  - is: rainy
  temperature:
  - added: -5
aggregation: !<maximum> {}
adjustments:
- !<users-multiplied>
  factor: 1.2
```

is a superset of JSON aiming to be more human-readable and is used by technologies such as Docker [15] and Kubernetes [55]. Fig. 3 provides the grammar of the language in extended Backus-Naur form (EBNF), visualized as syntax diagrams [23] and formatted to highlight the YAML structure. In the supplementary material [51], we provide the pure EBNF. Listing 2 provides an example, which describes the maximum workload expected during a sneakers release when the weather is cold and rainy, increased by 20 %.

The root clause of an LCtL instance is a `scenario`. It describes a workload scenario based on the contexts stored in the WMR in up to four sections, as described below.

4.3.1 Timeframe Section. Here, a user can specify the past or future time frame from which the workload model of the load test will be extracted. Hence, the section acts as a query language for selecting data from the WMR. We provide the following clauses, which define sequential restrictions or extensions to an unbound time frame. A `timerange` defines a time range starting at a defined date and time and having a potentially unbound duration. The `conditional` clause refers to the context facets stored in the WMR and selects the data where the specified `conditions` hold, e.g., the facets have a specific state, or have a numeric state greater or less a value. The `extended` clause extends the time frame defined by the previous clauses by a specified duration. This is especially useful if users are interested in the time before or after a specific state of a facet. The example (Listing 2) defines the time frame for the release of sneakers.

4.3.2 Context Section. This section is optional and allows users to modify the data queried with the `timeframe` section. They can change the facets' states for influencing the workload forecast as a what-if analysis, e.g., changing the expected weather conditions (see Listing 2). The section is a mapping of context facet names to one or multiple `context-def` clauses, which provides three keywords, whose usage depends on the type of facet: *is* sets the state to a specific value (all types); *added* and *multiplied* add or multiply the state of numeric facets with a value. The clause also has a *during* keyword for restricting the modifications to a time frame.

4.3.3 Aggregation Section. The third section is mandatory. It defines how to extract a workload scenario from the selected or forecasted per-group intensities. As the time frame can be large, a feasible aggregation is crucial. The section consists of a single clause, which we designed to be extensible. It refers to the unique *name* of

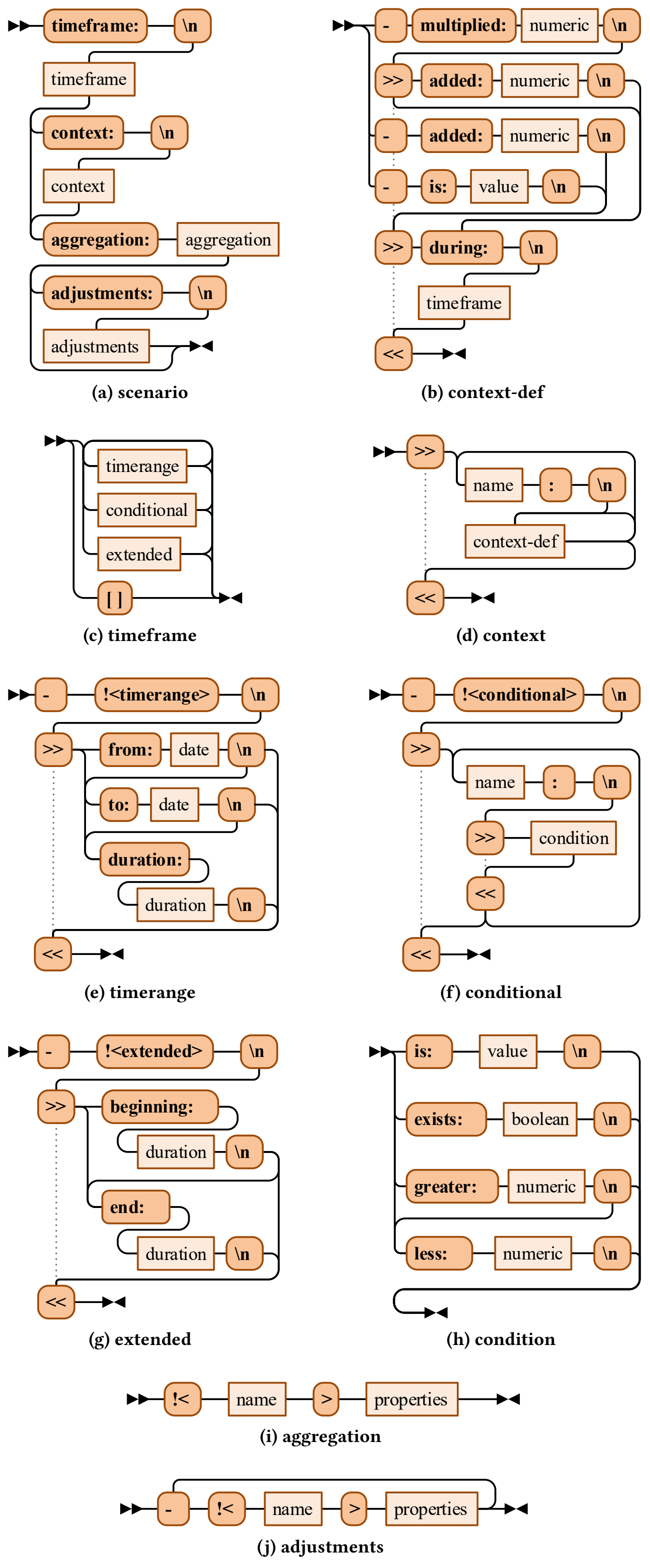


Figure 3: LCtL grammar. >> and << denote indents and dedents; \n denotes newlines respecting the indentation.

registered code snippets implemented in the R programming language [40], which get as input the (potentially forecasted) intensity values per group for the specified time frame and need to extract parts of it. Furthermore, the snippets can be parameterized with `properties`, which are either empty or key-value maps. By default, we provide the following aggregations: `as-is` returns the input it receives; `maximum` selects the steady-state workload with the maximum intensity (as in the example); `percentile` acts similarly but selects a defined percentile; `sharpest-spike` selects a part of the workload around the highest slope.

4.3.4 Adjustments Section. In this section, a user can incorporate a qualitative forecast into the workload scenario extracted by the `aggregation`. It is optional. Again, it is extensible and users can register R code snippets. The adjustments we provide by default are `users-multiplied` and `users-added`, which apply the respective operations to the intensities of either all groups or specific ones. In the example, we increase all groups by 20 %.

4.4 On-demand Load Test Generation

4.4.1 Workload and Context Preparation. As a preparation for further steps, we select all relevant data from the WMR based on the time frame specified in the `timeframe` section. These are the behavior models per group that represent the behavior before the start of the time frame, the past per-group intensities and context, and the future context until the end of the time frame. The behavior models and intensities constitute the latest state of the incrementally learned workload model. We will use the past and future context for forecasting the workload to the future. If the LCtL instance contains a `context` section, we modify the context retrieved from the WMR. In our example (Listing 2), we use the past data to forecast the future workload in contexts where the WMR sets the *product_release* facet to *sneakers*, and modify the weather conditions the WMR describes for the sneakers releases.

4.4.2 Workload Forecasting. If the LCtL instance specifies a future `timeframe`, we need to forecast the future workload. For that, we apply multivariate time series forecasting to the data prepared in the previous step, i.e., the per-group intensities and context. We obtain the future intensities, which, in combination with the behavior models, build the workload mix. This procedure is superior over a pure forecast of the total intensity, as it can also predict the mix. We utilize the Telescope [4] forecasting tool, which can use the prepared context as *covariates* for a more relevant forecast. For evaluation purposes, we also provide a perfect forecast, which returns the actual intensities that lie in the future from a defined perspective.

4.4.3 Aggregation & Adjustments. The final step is to aggregate and potentially adjust the intensity forecast (or selected past intensities). First, we apply the R code snippet registered for the defined `aggregation` to obtain a workload scenario. Then, we apply the code snippets for the `adjustments`. In the example, we select the steady-state workload with the maximum intensity and increase it. The result is a per-group workload intensity that can be replayed in a load test.

5 EVALUATION

We evaluated our approach with the student information system (SIS) of the Charles University in Prague (around 50,000 students). We applied our framework to the request logs from half a year and the publicly available academic calendars for learning a workload model and addressing the following research questions in three studies. For **RQ1** — *How expressive is the Load Test Context-tailoring Language concerning workload scenarios of a production system?*, we conducted an expert survey to identify relevant scenarios and expressed them using LCtL. Furthermore, we investigated scenarios arising from special contexts, such as the COVID-19 pandemic. Addressing **RQ2** — *How well do the continuously learned workload models describe the future workload?*, we generated and executed (against a mock server) several context-tailored load tests using a perfect, i.e., simulated forecast and compared the results to the actual workload. Finally, for **RQ3** — *How representative for future workload scenarios are the workload models with forecasted intensities?*, we did the same but for time frames after the learned workload model using real forecasting tools [4, 54]. Hence, we evaluated the on-demand part of the framework. A replication package is available online [51].

In the following, we present the preparation, metrics used, the studies, a discussion of the research questions, and a discussion of threats to validity.

5.1 Preparation

Using a publicly available dataset of request logs of the SIS [29], we learned a workload model and intensities from May 24 to Nov. 22, 2018. Furthermore, we had access to the request logs of Sep. 25 and Oct. 2, 2019, to compare the context-tailored load tests with. We initialized our framework with the half year as input to the incremental session clustering, which processed a four-week batch for the initial clustering with $k = 20$ and $\alpha = 0.95$. Further batches comprised one week each and were processed with $\beta = 1.1$ and $m = 500$. These parameters need to be settled once per application; in this case, we identified them in informal experiments with different settings. Overall, the clustering found 26 groups, including the noise cluster. Fig. 4 illustrates the resulting per-group intensities. As context, we used the publicly available academic calendars [11, 12], which comprise the facets *tuition*, *examination*, *final_exam*, *vacation*, *course_enrolment*, and *graduation_ceremony*. Further entries in the calendar, such as an open day, we found not to influence the workload. We modeled all facets as boolean ones, except for *tuition* and *course_enrolment*. Tuition phases have a higher workload in the beginning. Accounting for that, we modeled the facet numerically with a value of 10 in the beginning and decreasing negative exponentially to 1 within three weeks. *Course_enrolment* is a string facet, which distinguishes between the *priority* and *open* mode enrolment. Finally, we labeled strong spikes, which occurred during the course enrolment, as having the *special* state.

A drawback of the dataset is that it only contains the workload of half a year, i.e., the length of one semester. However, forecasting tools require multiple seasons for properly predicting the future.

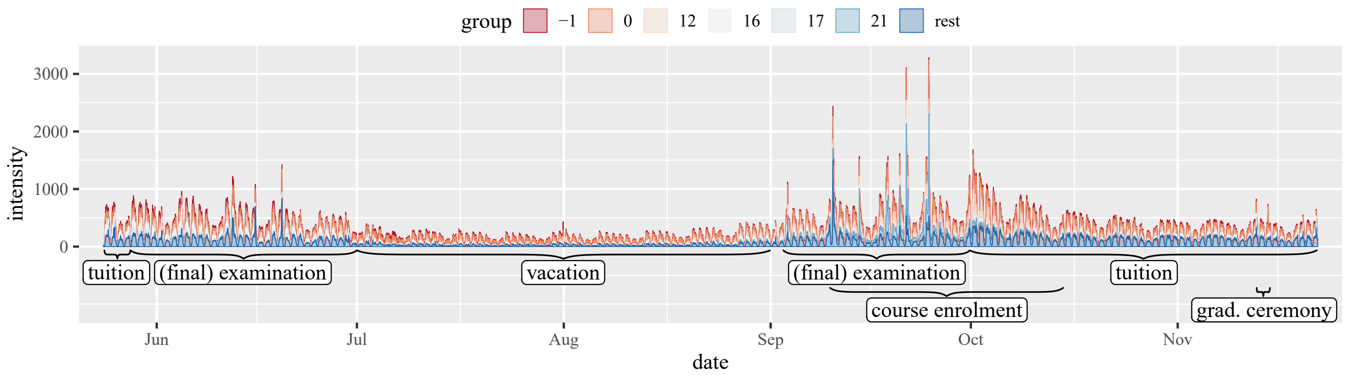


Figure 4: Intensities (number of concurrent users) of the 26 groups and context of the SIS (excluding augmentation).

Therefore, for the third study, we augmented the per-group intensities by another half year. As precisely documented in the supplementary material, we constructed the intensities according to the context and applied smoothing and jittering for randomization.

5.2 Metrics

5.2.1 Request Metrics. In previous work [52], we have introduced the ρ metric, which assesses the representativeness of a load test compared to a reference workload. It calculates the weighted and normalized difference of the number of requests per endpoint between the load test and the reference. Here, we utilize a slightly modified version $\overline{\rho}$, which considers the relative frequencies x_{test} and x_{ref} per endpoint instead of the absolute counts, i.e., $\overline{\rho} \approx \|(x_{\text{test}} - x_{\text{ref}})/x_{\text{ref}}\|$. The larger $\overline{\rho}$ is, the more the test's request mix differs from the reference. For a precise definition, we refer to [52]. To account for workload fluctuations, we calculate the metric per minute and use the average $\overline{\rho}$ for the whole test.

Even if the request mix is similar to the reference, the total number of requests might differ. The *request gap* is the difference in the overall number of requests per minute between the load test and the reference (summarized over all endpoints). Again, we use the sum for the whole test, expressed relative to the number of requests of the reference for comparison between tests.

5.2.2 Session Metrics. Further relevant metrics for session-based workloads are the session length (number of requests per session) and session duration (seconds a session lasts) [57]. We compare these between the test and the reference using the Kolmogorov-Smirnov (KS) statistic D. Throughout the paper, we use $\alpha = 0.05$, i.e., $c(\alpha) = 1.358$, and normalize D with $\sqrt{n_{\text{test}} \cdot n_{\text{ref}}/(n_{\text{test}} + n_{\text{ref}})}$ based on the sample sizes n_{test} and n_{ref} for inter-test comparison, such that the KS null hypothesis is rejected if the normalized D is greater than $c(\alpha)$.

5.3 Expressiveness Evaluation

5.3.1 Method. The study consists of two parts. First, we conducted an expert survey for collecting relevant workload scenarios, which we expressed using LCtL. The survey showed the participants the SIS workload, similar to Fig. 4, and asked for relevant load test scenarios. Precisely, we asked about (a) the participant's load testing experience, (b) relevant scenarios without knowing the context, and (c) relevant scenarios when knowing the context. We provide the survey questionnaire and individual answers in [51]. In order to obtain plausible scenarios, we invited known experts from industry and academia. Then, one of the authors defined LCtL instances for all specified scenarios, which a second author double-checked.

Second, we investigated the ability of LCtL to capture scenarios arising from the COVID-19 pandemic, which started in spring 2020. Due to measures taken by the Czech government, Charles University had to replace presence teaching with online teaching. We analyzed whether LCtL can be used for generating load tests that cover the affected SIS workload.

5.3.2 Results.

Expert Survey. The survey had six participants, who self-assessed as competent or expert in load testing. Overall, they specified 36 scenarios. Without knowing the context, most participants specified workload scenarios representing specific periods, many of which correspond to the phases defined by the context. One participant also distinguished between weekdays, i.e., workdays and weekends. When knowing the context, they defined similar scenarios but stated them more precisely, e.g., referring to context facets instead of dates. Several participants proposed to test the workload during the overlap of multiple facets, e.g., *examination* and *course_enrolment*.

Except for one case, we were able to express all scenarios by introducing two new LCtL clauses using the extension mechanisms, namely for selecting specific weekdays in the `timeframe` section and extracting a specific intensity curve in the `aggregation` section. The exception is one participant with expert knowledge, who stated to define the relevant scenarios *"based on assumed risk"* instead of the recorded workload. We presume they follow a different school of load design than we do, namely fault-inducing rather than representative [24], which our framework does not cover.

COVID-19 Pandemic. Among the measures the Charles University has taken due to the pandemic [13], we consider the following to be especially relevant for load testing. (1) As of Mar. 11, 2020, students were prohibited from attending classes in person. Teachers were recommended to continue teaching remotely via online tools. The students had to complete all assignments electronically. (2) As of Mar. 16, 2020, all employees had to work from home. (3) The

Listing 3: LCtL instance for online tuition.

```
timeframe:
- !<timeframe>
  from: 2020-03-11T00:00:00
  duration: P4W
  aggregation: !<percentile>
  p: 95
adjustments:
- !<users-multiplied>
  factor: 1.2
  group: 0
- !<users-multiplied>
  factor: 1.5
  groups: [12,14,21,22,23,24]
```

Listing 4: LCtL instance for the graduation ceremony.

```
timeframe:
- !<timeframe>
  from: XT00:00:00
  duration: P1D
context:
  graduation_ceremony:
    is: true
aggregation: !<sharpest-spike> {}
```

graduation ceremonies, which should have taken place on Apr. 21, 2020, were canceled. The university planned to have an alternative on a later date.

We deduce two load test scenarios, which cover the special circumstances. The first scenario follows from (1) and (2), which result in students and teachers accessing the SIS more than usual, especially for exchanging course material. Hence, we can test whether the SIS can handle such an increased steady-state workload. Listing 3 shows the LCtL instance for this scenario. As the workload expected under normal conditions, we selected the 95th percentile of the workload during four weeks starting from Mar. 11. For incorporating the workload increase, we analyzed the groups of the workload model, as our approach can differentiate between groups, leading to a more accurate forecast. In the first place, we expect the intensities of the groups related to course material exchange to increase — these are groups 12 and 21 to 24, because they frequently (9.3 % to 22.1 %) access the *course browser*, and group 14, which frequently (70.4 %) accesses the *student application*, which we identify as being relevant for online learning and study management. Group 0 users call many endpoints representing normal student activities, which are likely to increase, but less than the other groups. Hence, we can, e.g., increase the intensity of group 0 by 20 % and the intensities of the further listed groups by 50 %. These percentages illustrate a qualitative estimation to be made by the operators of the SIS. As a result, the load test reflects the workload predicted by quantitative forecasting, enriched with the qualitative forecast of online tuition. We refer to the groups by IDs, but we can also imagine labeling with user-friendly names, which would increase the comprehensibility of the LCtL instance.

A past occurrence of the graduation ceremony phase contained a workload spike. If this spike overlaps with other effects, it may stress the system extraordinarily. Hence, for measure (3), we can test the SIS under the workload at the new ceremony date. Listing 4 shows the LCtL instance, which uses X as a placeholder for the date. As opposed to the first scenario, we can rely on quantitative forecasting, which is influenced by the new ceremony date.

5.4 Experiments with Perfect Forecasting

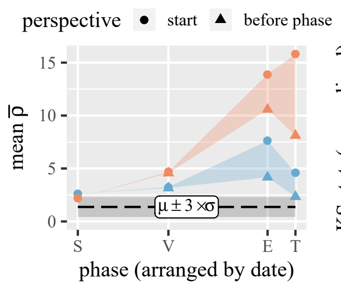
5.4.1 Experiment Process. For evaluating the ability of the incrementally learned workload model to predict the reference workload — i.e., requests and sessions of the SIS dataset —, we generated and executed 28 load tests using our framework with perfect forecasting, which varied in the following dimensions. (1) We predicted different phases using the LCtL `timeframe` section. First is the *start* phase shortly after the initial clustering period. We used it to assess the ability of the initial workload model to predict the workload it was built from. Further phases are *vacation*, *course_enrolment*, and *tuition* (see Fig. 4). (2) We generated steady-state and varying workloads using the 95th `percentile` and `sharpest-spike` aggregations. (3) We varied the perspective, i.e., the date of the workload model version used, between the *start* and directly *before* each phase. In doing so, we could assess how much incremental updates change the workload model and its ability to predict the reference workload. (4) For evaluating the benefits of predicting the workload mix in addition to the total intensity, we forecasted the groups' intensities *individually* or all in *total* using the workload mix of the workload model.

We executed the spike load tests for the duration of the spike and the steady-state tests for three hours (plus ramp-up and cool-down). For comparison with the reference, i.e., the real users' requests during the predicted period, we used the server-side request log produced by the test. Additionally, we executed the load test with the *start* phase and perspective, *individual* forecast, and `percentile` aggregation ten more times, and — with the first execution as the reference — used these executions to assess the normal variation of a single test.

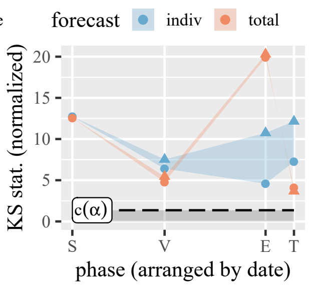
5.4.2 Experiment Setup. We used two bare-metal machines, which both had an Intel® Xeon® CPU E5620 with 2.40 GHz clock frequency, 4 cores, and 8 threads, and were connected via a 1 Gbit switch. The first machine had 32 GiB RAM and hosted our framework and JMeter [1] for executing the load tests. The second machine had 8 GiB RAM and hosted a mock server [45] that collected the requests the load tests submitted.

5.4.3 Results per Aggregation Used.

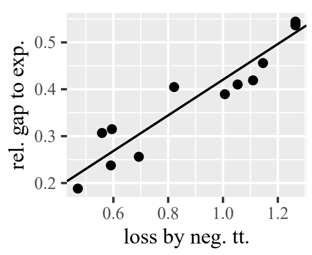
Percentile. Fig. 5 and Table 1 provide an overview of the request and session metrics. For the request metrics ($\overline{\rho}$ and request gap), we calculated baselines from the corresponding test executions. The average $\overline{\rho}$ is higher than the baseline μ for all tests and, except for one case, also higher than $\mu + 3\sigma$. Hence, the request mix differs (only slightly for some tests) from the reference. For the tests with individually forecasted intensities and perspective before the phase, the difference is the smallest. The phase with the overall largest difference is the course enrolment, whose workload sharply differs from other phases (see Fig. 4). The request gap is different from its baseline, too. Analyzing that further, we identified the gap to the expected numbers of requests, which we obtained by solving the Markov chains of the per-group behavior models, to be significant as well. As a potential reason, we found that the think time



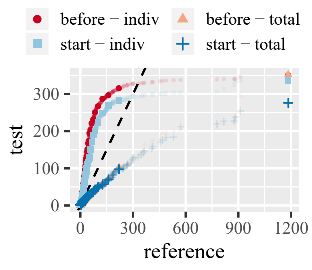
(a) overview of $\overline{\rho}$ metric



(b) overview of session length

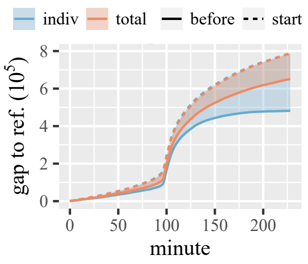


(c) relative request gap

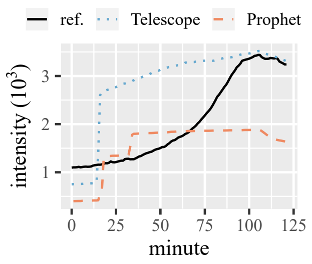


(d) session length Q-Q plot (vacation)

Figure 5: Plots of tests with `percentile` aggregation (S = start, V = vacation, E = enrolment, T = tuition).



(a) request gap (perfect forecast)



(b) intensity (real, individual forecast with context)

Figure 6: Plots of tests with `spike` aggregation.

Table 1: Metric Summary of Selected Test Scenarios

test scenario	mean $\overline{\rho}$	gap	KS (len)	KS (dur)
		perfect forecasting		
S/perc/start/indiv	2.627	0.346	12.701	22.857
S/perc/start/total	*2.200*[*]	0.505	12.565	23.485
E/perc/before/indiv	4.176	0.242	10.749	18.558
E/perc/before/total	10.575	0.480	20.302	11.308
T/perc/before/indiv	2.338	0.229	12.153	17.425
T/perc/start/total	15.798	0.649	4.101	14.153
S/spike/start/indiv	*1.974*[*]	0.339	10.596	20.569
V/spike/before/indiv	5.554	0.677	6.656	3.075
E/spike/before/indiv	9.185	0.447	21.418	11.098
T/spike/before/indiv	9.221	0.459	14.858	13.901
		real forecasting (Telescope)		
T/perc/indiv/context	6.297	0.606	13.123	18.006
T/perc/indiv/pure	7.703	0.796	9.855	12.419
T/perc/total/context	21.419	0.757	11.521	13.623
T/perc/total/pure	20.203	0.872	7.513	11.445
E/spike/indiv/context	6.093	0.437	10.137	15.897
		real forecasting (Prophet)		
T/perc/indiv/context	8.366	0.479	12.218	18, 540
T/perc/indiv/pure	11.595	0.696	9.600	14.526
T/perc/total/context	22.083	0.634	12.218	16.591
T/perc/total/pure	22.623	0.787	10.607	13.429
E/spike/indiv/context	10.377	0.681	10.165	5.655
E/spike/total/context	25.990	0.716	26.603	16.720

[*]below $\mu + 3\sigma$ of the respective baseline (2.311 for $\overline{\rho}$ and 0.206 for request gap)

specifications between two endpoints — which are normal distributions, similar to existing work [57] — had a significant negative portion stemming from large variances. As a load test cannot simulate negative think times, it replaces them with zero, resulting in a biased mean think time. Our explanation is supported by the fact that the lost time correlates with the gap to expected requests (see Fig. 5c) with a Pearson correlation coefficient of 0.952. Indicating that it does not stem from bottlenecks in the load driver, the gap does not correlate with the number of concurrent users, where the coefficient is −0.131.

The session metrics (length and duration) give a slightly different picture. As shown in Fig. 5b, the KS statistics for the tests with total forecast are mostly below the individual forecasts, but still above $c(\alpha)$. An exception is the course enrolment phase, highlighting its peculiarity. For the other phases, quantile-quantile (Q-Q) plots (Fig. 5d) show that the sessions of tests with individual forecasts are longer, except for the tail, while the ones with total forecasts are shorter than the reference.

Sharpest Spike. In general, the results of the spike tests have similar characteristics as the tests with the percentile aggregation. A new aspect, however, can be seen in tests with notably sharp spikes. As an example, the cumulative request gap of the tests for the course enrolment phase sharply increases after about 100 minutes (see Fig. 6a). So does the $\overline{\rho}$ metric. The reason is that at that time, the request rate of the reference workload sharply increases, without a corresponding increase in the number of concurrent sessions. In contrast with the request rate, the number of concurrent sessions generated by all tests is close to the reference.

5.5 Experiments with Real Forecasting

5.5.1 Experiment Process. For evaluating the ability of workload models with forecasted intensities to predict a reference workload, we generated and executed sixteen further load tests using the augmented intensities (see Section 5.1). The tests differed in the following dimensions. (1) They replayed the workload scenarios predicted for two specific days after the end of the (augmented) intensities (May 23, 2019), namely the highest workload during *course enrolment* (Sep. 25, 2019) and the first *tuition* day (Oct. 2, 2019), for which we had access to the request logs. For the tuition day, we predicted the 95th `percentile`; for the enrolment, we predicted the `highest-spike`, which is a more robust variant of the `sharpest-spike` (instead of locating the spike with the sharpest increase, it detects a spike pattern around the peak load). We applied this aggregation because we found that the tools used calculated forecasts that lead to wrongly predicted sharpest spikes (see results section). (2) Similar to before, we forecasted the intensities *individually* and in *total*. (3) We used the *Telescope* [4] and *Prophet* [54] tools for computing the intensity forecasts. (4) We expect the context to improve the intensity forecast. For validating this hypothesis, we supported the forecasting tools with the *context* or executed it *purely*, i.e., without context. In the enrolment case with context, we defined a special enrolment from 9:30 to 11:00 using the `context` section of the LCtL.

For analyzing the load test results, we compared the requests and sessions executed by the tests with the corresponding ones from the two days. For the course enrolment, we aligned the intensity peaks of the load tests and the reference. For the tuition, we extracted the 95th percentile.

5.5.2 Experiment Setup. Similar to Section 5.4.2.

5.5.3 Results per Scenario.

Tuition Percentile. Table 1 provides a summary of the load test results. The request and session metrics are in a similar range as for the tests with perfect forecasting, but higher, e.g., the average $\overline{\rho}$ for the test with Telescope and individual forecast considering the context is 6.297, as opposed to 2.338 for its counterpart with perfect forecasting. In general, Telescope generates more representative workload predictions than Prophet. An exception is the request gap, which is smaller for Prophet than for Telescope. However, all metric values for the two tools are close to each other.

The individual forecast has the strongest effect on the request metrics. It reduces $\overline{\rho}$ by 48 % to 71 % compared to the total forecast. Also, the request gap is smaller with individual forecasting compared to total forecasting, with differences between 8 % and 25 %. The data basis has a small effect. In all cases except one, using the context reduces $\overline{\rho}$ by 2 % to 39 %. For the test with Telescope applied to the total intensities, the context slightly increases $\overline{\rho}$ by 6 %. However, the context reduces the request gap in all cases.

As before, the total forecast produces smaller session metrics, with Q-Q plots similar to Fig. 5d. Besides, Telescope predicts workloads with slightly smaller session metric values than Prophet does in most cases. As all tests' workloads significantly differ from the reference workload, we cannot deduce significant differences between Telescope and Prophet regarding the session metrics.

Enrolment Spike. Predicting the sharpest spike during the course enrolment appeared to be challenging for the forecasting tools. For Telescope with total forecasting and Prophet with the pure data basis, the predicted workload did not contain a notable spike. In these cases, the sharpest spike patterns contained in the forecasts lasted between ten and twenty hours. Therefore, we only analyze the remaining test scenarios. The resulting metrics (see Table 1) show high variation among the tests. The most representative workload was predicted by Telescope with individual forecasting, using the context. Its metrics are close to the enrolment spike predicted with perfect forecasting. The request metrics are even lower. With Prophet, both tests have a similar request gap, but the individual forecast generates a better $\overline{\rho}$.

However, as Fig. 6b illustrates, the curve shape of the number of concurrent sessions generated by the tests differs significantly from the reference. Mainly, there is an extremely sharp increase shortly after the start, which does not exist in the reference. The increase correlates with the start of the *special* enrolment. Still, Telescope predicted the peak intensity accurately when using the context.

5.6 Discussion of Research Questions

5.6.1 RQ1 — Expressiveness. Due to its flexibility, LCtL is suitably expressive for workload scenarios of the SIS. For one, we were able to express all scenarios from the expert survey. In few cases, we had to extend the language with further clauses, for selecting weekdays and for a specific aggregation. Additionally, we could define unusual scenarios arising from the COVID-19 pandemic. Here, the mix of quantitative and qualitative forecasting was required for estimating the impact of the online tuition. A limitation of LCtL is non-representative workloads, such as fault-inducing [24], which our framework does not cover by design.

5.6.2 RQ2 — Describing Future Workload. The incrementally learned workload model describes the reference workload accurately, except for two influences. First, the think time specifications have too large variance, which induces a significant request gap. As the user groups are affected differently, the request mix and, thus, the $\overline{\rho}$ metric are affected, too. Taking that into account, the workload model perspective learned just before the forecast phase, in combination with individual forecasting, generates representative load tests, while especially total forecasting gives a worse result. Hence, predicting the workload mix is superior over predicting the total intensity only. The fact that the sessions of the individually forecasted tests tend to be longer than the reference, while the tests with total forecasts have shorter sessions, supports this finding, as Markov chains have generally been found to generate long sessions [57]. Second, the workload model is limited regarding predictions of sharp spikes. While the corresponding load tests executed the correct number of concurrent sessions, the request rate and mix were different from the reference. A reason might be that the think times during the spike differ from other phases.

Future work needs to address the mentioned limitations by better integrating the think times into the workload model. For instance, the session clustering should consider them, for reducing the variance. Also, this might improve predictions of sharp spikes, as such modified clustering can recognize a different user behavior during the spike.

5.6.3 RQ3 — Representativeness with Forecasting. Load tests with forecasted intensities can be mostly as representative as with perfect forecasting when considering the context and per-group intensities. For steady-state workloads, the tests are only slightly less representative compared to perfect forecasting. Here, the individual forecasts made the largest difference. When predicting spikes, it is crucial to use the context, as all other tests did not contain a notable spike. We presume that, on the one hand, the context helps the forecasting tool separating the spikes from anomalies, while, on the other hand, the group's individual intensities allow detecting the spike better, as it occurs differently in different groups.

A limitation of the spike forecast is predicting transitions between context facet states, e.g., *open* to *special* course enrolment, which can cause a sharp change in the intensity curve. We presume the reason is that the tool learns the average influence of a specific state in the past and applies it to the whole state in the future. For our purposes, we rather need a forecasting approach that also learns the change in the intensity curve, which needs to be addressed by future work.

Regarding the forecasting tools used, Telescope predicts slightly more representative workloads than Prophet does. For steady-state workloads, this is mainly reflected in the request metrics. For spike workloads, Telescope also predicts intensity curves that are closer to the reference, even if it suffers from the context transitions and the uncorrelated request rates. However, Telescope necessarily requires the context and individual per-group intensities, while Prophet performs better on total intensities.

5.7 Threats to Validity

5.7.1 Conclusion Validity. We have not applied statistical testing for assessing the representativeness of the generated load tests. Statistical tests are not suited for comparing predicted workload scenarios with a reference workload, as the predictions naturally deviate from the reference. Instead, we have measured the distance to the reference with metrics introduced and successfully applied in existing work [52, 57].

5.7.2 Internal Validity. During three short-term (up to few hours) periods of the session clustering, we lost a low percentage ($< 1\,\%$) of data due to platform issues and clock change. However, the losses are negligible compared to the total size of the dataset. During test execution, we prevented interactions between the load driver and the system under test by using separate machines. Also, we validated that the load driver did not run into overload situations by correlating the request gap with the intensity. For applying real forecasting, we augmented the SIS dataset, which might have influenced the forecasts. Using a dataset with a longer period of data available would be beneficial. However, to the best of our knowledge, the SIS dataset is unique regarding the information content and extent of real-world session-based workloads. Also, time series augmentation is commonly applied [61].

5.7.3 Construct Validity. We identified several limitations of existing workload modeling and forecasting approaches, which are reasonable, as we have chosen state-of-the-art concepts and tools that have extensively been evaluated [4, 54, 57]. Particularly, Markov chains have been assessed a reasonable modeling concept for load tests [26]. Also, the limitations do not stem from our implementations, as they are related to the session clustering and Markov-chain-based workload modeling introduced by Menascé et al. [31] and Vögele et al. [57], and time-series forecasting, which we have applied as a black box. Besides, our framework allows using different tools, which can be evaluated in future work.

5.7.4 External Validity. The SIS dataset has particular characteristics, such as semester phases sharply differing in the workload. The workloads of other systems can have different characteristics leading to different results. Therefore, future work should aim at finding more datasets related to the one used in this paper and evaluate our framework on them.

6 CONCLUSION

In this paper, we introduce a framework for generating load tests tailored to the varying workload scenarios and workload-influencing contexts of session-based applications. The framework provides an end-to-end approach that allows users to describe context-tailored load tests in the Load Test Context-tailoring Language (LCtL), which abstracts from the technical details of the load test generation. For automated processing of an LCtL instance, the framework integrates incremental session clustering based on existing work and forecasting tools.

Applied to the Charles University's SIS, we evaluated LCtL to be suitably expressive for real-world load test scenarios. Besides, we conclude that the framework is suited for generating representative load tests that replay past or future workload scenarios. Notably, we have found the context and separation of the workload into user groups to improve the workload predictions significantly. However, some limitations of existing workload characterization and forecasting concepts, such as think time modeling and predictions of sharp spikes, reduce the representativeness of the generated tests.

Therefore, future work needs to find solutions for these limitations. We suggest investigating session clustering algorithms that integrate the think times and, thus, can react to short-term changes in the session timings, such as during workload spikes. Besides, modeling an open instead of closed workload [43] — i.e., considering the session arrival rates instead of concurrent sessions — might help to predict spikes, which, however, requires careful modeling of the session length and duration. As our and previous evaluations [57] indicate, Markov chains might not be suited for that. Finally, future work should address multivariate time series forecasting that can predict the shape of sharp spikes.

ACKNOWLEDGMENT

This work is being supported by the German Federal Ministry of Education and Research (grant no. 01IS17010, ContinuITy) and the European Union's Horizon 2020 research and innovation programme (grant no. 825040, RADON).

REFERENCES

[1] Apache Software Foundation. 2020. Apache JMeter. (2020). http://jmeter.apache.org/

[2] David Arthur and Sergei Vassilvitskii. 2007. K-Means++: The Advantages of Careful Seeding. In *Proc. SODA 2007.* SIAM, 1027–1035.

[3] Marcelo De Barros, Jing Shiau, Chen Shang, Kenton Gidewall, Hui Shi, and Joe Forsmann. 2007-06. Web Services Wind Tunnel: On Performance Testing Large-Scale Stateful Web Services. In *Proc. DSN 2007.* IEEE Computer Society, 612–617.

[4] André Bauer, Marwin Züfle, Nikolas Herbst, Samuel Kounev, and Valentin Curtef. 2020. Telescope: An Automatic Feature Extraction and Transformation Approach for Time Series Forecasting on a Level-Playing Field. In *Proc. ICDE 2020.*

[5] Cor-Paul Bezemer, Simon Eismann, Vincenzo Ferme, Johannes Grohmann, Robert Heinrich, Pooyan Jamshidi, Weiyi Shang, André van Hoorn, MÃşnica Villavicencio, Jürgen Walter, and Felix Willnecker. 2019. How Is Performance Addressed in DevOps?. In *Proc. ICPE 2019.* ACM, 45–50.

[6] James Billington. 2014. Ellen DeGeneres selfie at Oscars 2014 breaks Twitter. https://www.news.com.au/technology/online/ellen-degeneres-selfie-at-oscars-2014-breaks-twitter/news-story/da9c8e1fe7d8e73ed791143872f10bbe. (2014).

[7] Bradley Web Group. 2016. Does your Website Traffic go down at the weekend? https://bradleywebgroup.com/website-traffic-go-weekend/. (2016).

[8] Yuhong Cai, John Grundy, and John G. Hosking. 2007. Synthesizing Client Load Models for Performance Engineering via Web Crawling. In *Proc. ASE 2007*. ACM, 353–362.

[9] Maria Carla Calzarossa, Luisa Massari, and Daniele Tessera. 2016. Workload Characterization: A Survey Revisited. *ACM Comp. Surv.* 48, 3 (2016), 48:1–48:43.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly Detection: A Survey. *ACM Comp. Surv.* (2009).

[11] Charles University, Faculty of Mathematics and Physics. 2017. Academic Calendar 2017/2018. https://www.mff.cuni.cz/en/students/academic-calendar/academic-calendar-2017-2018. (2017).

[12] Charles University, Faculty of Mathematics and Physics. 2018. Academic Calendar 2018/2019. https://www.mff.cuni.cz/en/students/academic-calendar/academic-calendar-2018-2019. (2018).

[13] Charles University, Faculty of Mathematics and Physics. 2020. Measures regarding the coronavirus SARS-CoV-2 and the COVID-19 disease. http://mff.cuni.cz/coronavirus. (2020).

[14] David Daly, William Brown, Henrik Ingo, Jim O'Leary, and David Bradford. 2020. Change Point Detection in Software Performance Testing. In *Proc. ICPE 2020*. 67–75.

[15] Docker Inc. 2020. Docker: Empowering App Development for Developers. https://www.docker.com/. (2020).

[16] Elasticsearch B.V. 2020. Elasticsearch: The Official Distributed Search & Analytics Engine. https://www.elastic.co/elasticsearch. (2020).

[17] Vincenzo Ferme and Cesare Pautasso. 2017. Towards Holistic Continuous Software Performance Assessment. In *Comp. ICPE '17*. 159–164.

[18] Vincenzo Ferme and Cesare Pautasso. 2018. A Declarative Approach for Performance Tests Execution in Continuous Software Development Environments. In *Proc. ICPE 2018*. ACM.

[19] Nina Godlewski. 2017. Nike Site Down On Black Friday, How To Place An Order. https://www.ibtimes.com/nike-site-down-black-friday-how-place-order-2619427. (2017).

[20] Raoufehsadat Hashemian, Niklas Carlsson, Diwakar Krishnamurthy, and Martin Arlitt. 2017. IRIS: Iterative and Intelligent Experiment Selection. In *Proc. ICPE '17*. 143–154.

[21] Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. 2013. Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning. In *Proc. ICPE 2013*. ACM, 187–198.

[22] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education.

[23] Kathleen Jensen and Niklaus Wirth. 1975. *Pascal User Manual and Report, Second Edition*. Springer.

[24] Zhen Ming Jiang and Ahmed E. Hassan. 2015. A Survey on Load Testing of Large-Scale Software Systems. *IEEE Trans. on Softw. Eng.* 41 (2015).

[25] Diwakar Krishnamurthy, Jerome A. Rolia, and Shikharesh Majumdar. 2006. A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems. *IEEE Trans. on Softw. Eng.* 32 (2006), 868–882.

[26] Zhao Li and Jeff Tian. 2003. Testing the Suitability of Markov Chains as Web Usage Models. In *Proc. COMPSAC 2003*. IEEE Computer Society, 356–361.

[27] Stuart P. Lloyd. 1982. Least Squares Quantization in PCM. *IEEE Trans. on Information Theory* 28 (1982), 129–136.

[28] Christof Lutteroth and Gerald Weber. 2008. Modeling a Realistic Workload for Performance Testing. In *Proc. ECOC 2008*. IEEE Computer Society, 149–158.

[29] Martin Maňásek and Petr Tůma. 2019. Charles University SIS Access Log Dataset. (2019). https://zenodo.org/record/3241445

[30] Daniel A. Menascé and Virgilio A. F. Almeida. 2002. *Capacity Planning for Web Services: Metrics, Models and Methods*. Prentice Hall.

[31] Daniel A. Menascé, Virgilio A. F. Almeida, Rodrigo Fonseca, and Marco A. Mendes. 1999. A Methodology for Workload Characterization of E-Commerce Sites. In *Proc. EC 1999*. ACM, 119–128.

[32] Chris Mills. 2018. Amazon's website went down just as Prime Day 2018 kicked off. https://bgr.com/2018/07/16/amazon-website-down-prime-day-2018-best-deals/. (2018).

[33] Shaikh Mostafa, Xiaoyin Wang, and Tao Xie. 2017. PerfRanker: Prioritization of Performance Regression Tests for Collection-Intensive Software. In *Proc. ISSTA 2017*. 23–34.

[34] John D. Musa. 1993. Operational Profiles in Software-Reliability Engineering. 10, 2 (1993), 14–32.

[35] Sam Newman. 2015. *Building Microservices - Designing Fine-Grained Systems* (1st ed.). O'Reilly.

[36] Novatec Consulting GmbH and University of Stuttgart, Software Quality and Architecture Group. 2020. ContinuITy: Automated Performance Testing in Continuous Software Engineering. https://continuity-project.github.io/. (2020).

[37] Dušan Okanović, Samuel Beck, Lasse Merz, Christoph Zorn, Leonel Merino, André van Hoorn, and Fabian Beck. 2020. Can a Chatbot Support Software Engineers with Load Testing? Approach and Experiences. In *Proc. ICPE '20*. 120–129.

[38] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-Learn: Machine Learning in Python. 12, 85 (2011), 2825–2830. http://jmlr.org/papers/v12/pedregosa11a.html

[39] Bogdan Popa. 2018. GitLab says it imported 100,000 repositories after Microsoft's GitHub takeover. https://news.softpedia.com/news/gitlab-says-it-imported-100-000-repositories-after-microsoft-s-github-takeover-521433.shtml. (2018).

[40] R Core Team. 2019. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. https://www.R-project.org/

[41] David Georg Reichelt and Stefan Kühne. 2018. Better Early Than Never: Performance Test Acceleration by Regression Test Selection. In *Comp. ICPE '18*. 127–130.

[42] Giancarlo Ruffo, Rossano Schifanella, Matteo Sereno, and Roberto Politi. 2004. WALTy: A User Behavior Tailored Tool for Evaluating Web Application Performance. In *Proc. NCA 2004*. IEEE Computer Society, 77–86.

[43] Bianca Schroeder, Adam Wierman, and Mor Harchol-Balter. 2007. Open versus Closed: A Cautionary Tale. In *Proc. NSDI 2006*. USENIX.

[44] Henning Schulz. 2020. ContinuITy-Project/Forecastic: Release at Project End (v0.5.3). https://doi.org/10.5281/zenodo.3966834. (2020).

[45] Henning Schulz. 2020. ContinuITy-Project/sutmock: SUT Mock version v0.2.1. https://doi.org/10.5281/zenodo.3966903. (2020).

[46] Henning Schulz, Tobias Angerstein, Dušan Okanović, and André van Hoorn. 2019. Microservice-tailored Generation of Session-Based Workload Models for Representative Load Testing. In *Proc. MASCOTS 2019*. 323–335.

[47] Henning Schulz, Tobias Angerstein, Manuel Palenga, and Alper Hidiroglu. 2020. ContinuITy-Project/ ContinuITy: Release at Project End (v2.9.346). https://doi.org/10.5281/zenodo.3966805. (2020).

[48] Henning Schulz, Tobias Angerstein, and André van Hoorn. 2018. Towards Automating Representative Load Testing in Continuous Software Engineering. In *Comp. ICPE 2018*. 123–126.

[49] Henning Schulz and An Dang. 2020. ContinuITy-Project/Clustinator: Release at Project End (v0.7.4). https://doi.org/10.5281/zenodo.3966829. (2020).

[50] Henning Schulz, Dušan Okanović, André van Hoorn, Vincenzo Ferme, and Cesare Pautasso. 2019. Behavior-driven Load Testing Using Contextual Knowledge - Approach and Experiences. In *Proc. ICPE 2019*. 265–272.

[51] Henning Schulz, Dušan Okanović, André van Hoorn, and Petr Tůma. 2020. Supplementary Material. https://doi.org/10.5281/zenodo.4355190. (2020).

[52] Henning Schulz, André van Hoorn, and Alexander Wert. 2020. Reducing the Maintenance Effort for Parameterization of Representative Load Tests Using Annotations. *Softw. Test. Verif. Reliab.* (2020).

[53] Taylor Soper. 2012. Tons of traffic: Amazon dominates online retail during Christmas week. https://www.geekwire.com/2012/christmas-day-2012-retail-visits-increase-27-compared-2011/. (2012).

[54] Sean J. Taylor and Benjamin Letham. 2018. Forecasting at Scale. *The American Statistician* 72 (2018), 37–45.

[55] The Linux Foundation. 2020. Production-grade container orchestration - Kubernetes. https://kubernetes.io/. (2020).

[56] The YAML Project. 2020. The Official YAML Web Site. http://yaml.org/. (2020).

[57] Christian Vögele, André van Hoorn, Eike Schulz, Wilhelm Hasselbring, and Helmut Krcmar. 2018. WESSBAS: Extraction of Probabilistic Workload Specifications for Load Testing and Performance Prediction—a Model-Driven Approach for Session-Based Application Systems. *Software and System Modeling* 17 (2018), 443–477.

[58] Jóakim von Kistowski, Nikolas Roman Herbst, and Samuel Kounev. 2014. Modeling Variations in Load Intensity over Time. In *Proc. LT 2014*. ACM, 1–4.

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

[60] Elizabeth Weise. 2016. Netflix down for about 2.5 hours Saturday. https://eu.usatoday.com/story/tech/news/2016/10/01/netflix-goes-down-saturday-afternoon/91396200/. (2016).

[61] Qingsong Wen, Liang Sun, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. 2020. Time Series Data Augmentation for Deep Learning: A Survey. (2020). arXiv:2002.12478 http://arxiv.org/abs/2002.12478

[62] Wikipedia. 2020. Wikipedia:Pageview statistics. https://en.wikipedia.org/wiki/Wikipedia:Pageview_statistics. (2020).

[63] S. Yoo and M. Harman. 2012. Regression Testing Minimization, Selection and Prioritization: A Survey. *Softw. Test. Verif. Reliab.* 22, 2 (2012), 67–120.

[64] Liming Zhu, Len Bass, and George Champlin-Scharff. 2016. DevOps and its practices. *IEEE Software* 33, 3 (2016), 32–34.