Using Regression Learners to Predict Performance Problems on Software Updates: a Case Study on Elevators Dispatching Algorithms

Aitor Gartziandia
Ikerlan
agarciandia@ikerlan.es

Aitor Arrieta
Mondragon University
aarrieta@mondragon.edu

Aitor Agirre
Ikerlan
aagirre@ikerlan.es

Goiuria Sagardui
Mondragon University
gsagardui@mondragon.edu

Maite Arratibel
Orona
marratibel@orona-group.com

ABSTRACT

Remote software deployment and updating has long been commonplace in many different fields, but now, the increasing expansion of IoT and CPSoS (Cyber-Physcal System of Systems) has highlighted the need for additional mechanisms in these systems, to ensure the correct behaviour of the deployed software version after deployment. In this sense, this paper investigates the use of Machine Learning algorithms to predict acceptable behaviour in system performance of a new software release. By monitoring the real performance, eventual unexpected problems can be identified. Based on previous knowledge and actual run-time information, the proposed approach predicts the response time that can be considered acceptable for the new software release, and this information is used to identify problematic releases. The mechanism has been applied to the post-deployment monitoring of traffic algorithms in elevator systems. To evaluate the approach, we have used performance mutation testing, obtaining good results. This paper makes two contributions. First, it proposes several regression learners that have been trained with different types of traffic profiles to efficiently predict response time of the traffic dispatching algorithm. This prediction is then compared with the actual response time of the new algorithm release, and provides a verdict about its performance. Secondly, a comparison of the different learners is performed.

KEYWORDS

Machine learning, Performance bugs, Cyber-physical systems

ACM Reference Format:
Aitor Gartziandia, Aitor Arrieta, Aitor Agirre, Goiuria Sagardui, and Maite Arratibel. 2021. Using Regression Learners to Predict Performance Problems on Software Updates: a Case Study on Elevators Dispatching Algorithms. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3412841.3441894

1 INTRODUCTION

Elevators systems are among the most widely used transportation systems. An elevator system is composed of several subsystems that, interacting among them, have as a goal to transport passengers safely and by considering certain Quality-of-Service (QoS) measures [2]. To this end, new functionalities of elevators are increasingly implemented through software [1]. A key component to maintain QoS measures within acceptable values in a system of elevators refers to the dispatching algorithm. These algorithms are in charge of assigning an elevator to each passenger. Being a critical component to ensure the correct operation of a system of elevators, they must be continuously maintained and evolved, addressing issues like bug-fixes, new functionalities, adaptation to legislation changes, etc.

New communication technologies (e.g., 5G) allow for updating new software versions remotely. This facilitates and shortens the release time of new versions. Subsequently, updates can be performed frequently, every time a new functionality is included or a bug is fixed, as manual intervention is minimized. As any other type of software, the dispatching algorithms are not exempt of bugs, either functional or non-functional. Specifically, non-functional bugs are hard to detect [22], and often infeasible until they are deployed on the final target (i.e., the embedded processor). Thus, run-time monitoring techniques are paramount to enable the detection of such faults in operation.

In this context, we propose a performance problem detection approach for software updates, leveraged by Machine Learning (ML). Our approach predicts the performance of a new software version, based on what it learns about the behaviour of the old version. Then, it compares its prediction to the actual behaviour of the system and gives a verdict, indicating whether the behaviour of the new release is correct or a performance problem exists. To evaluate our approach, we have followed a performance mutation testing strategy, where different mutants with performance bugs are analyzed. Different Machine Learning algorithms were tested to identify which ones best fit our purpose.

This paper reports on the experience of applying our machine learning-based performance problem detection in an industrial use-case provided by Orona. The main contributions of this paper can be summarized as follows:

- We propose an approach based on regression learning algorithms that are trained with different types of passenger traffic profiles to predict the response time of the traffic dispatching algorithm of elevators. This prediction is later compared with the actual response time given by traffic dispatching algorithm in order to detect any possible inconsistency related to a performance bug. A trustworthy approach is employed to provide a verdict (i.e., PASS or FAIL) at run-time, which indicates whether a bug has been detected or not.

- The proposed approach is evaluated with an industrial case study provided by Orona. Specifically, performance mutation testing [8] is employed to seed performance bugs through the software program. We compare a total of five state-of-the-art Machine Learning algorithms to predict performance problems on elevators dispatching algorithms. We showed that three of them are competent enough in order to be employed in practice.

The rest of the paper is organized as follows. Section 2 provides background and related work, where we position our approach with the current state-of-the-art. Section 3 provides an overview of the performance requirements in the context of elevator dispatching algorithms. In Section 4 we present our method to predict performance problems in traffic dispatching algorithms. The evaluation of the approach is carried out in Section 5, where we used performance mutation testing to assess how different Machine Learning algorithm perform to detect potential non-functional inconsistencies. We conclude our paper in Section 6, summarizing the future work we envision.

2 BACKGROUND AND RELATED WORK

Performance issues consist on errors producing a behaviour degradation of applications in terms of execution time, response time, CPU usage, memory usage or energy consumption, without necessarily causing any fault on the expected results [8]. With the growth of the software complexity, ensuring the performance health in resource constrained applications is getting increasingly relevant. Identification of performance problems in the testing phase brings some limitations, such as the elicitation of proper performance requirements. In addition, many causes that lead to the manifestation of these problems may appear due to the system's interaction with the real environment [25]. At run-time, identifying performance problems may require a continuous monitoring of the execution, as problems may only be revealed under certain circumstances, for example, activation of specific modes or functionalities, when the system has been running for a long period, etc.

There exist a variety of root causes during implementation that can result in real-world performance problems. Jin et al. [14] identified (1) inefficient function calls, (2) skippable functions doing unnecessary work and (3) synchronization issues as main root causes for performance problems. Other less common causes are also mentioned, including wrong data structure usage, hardware architecture issues or high-level design errors [14]. By analyzing how these bugs are introduced by developers, they concluded that these errors are usually introduced due to an API or workload misunderstanding [14]. Zhang et al. [27] studied the different synchronization errors that can appear in distributed systems. According to their study, the main factors causing synchronization issues are time-consuming operation, nested loops, recursive calls and high frequency locks. A recent study by Delgado et al. [8] identified some common causes of performance problems and generated mutants based on them in order to conduct performance mutant testing.

From the testing perspective, testing non-functional properties is becoming paramount. Ferme et al. [10] emphasize the importance of integrating performance analysis inside the software life-cycle management process. Their main identified challenges encompass (1) definition of a method to establish which performance tests to perform in each life-cycle stage, (2) enabling users to declaratively define performance objectives and (3) providing fast performance feedback to improve the system. In order to design test cases, Weyuker et al. [25] summarize important aspect to be considered by practitioners. This includes (1) the design of test generation and selection strategies and algorithms, (2) definition of metrics to assess the effectiveness of performance testing strategies and (3) comparison of different hardware platforms for a given application. Besides testing, other approaches have focused on static analysis approaches to detect performance issues [18, 20]. Testing performance of software systems have been extended to many subfields, including cloud computing [16], distributed systems [9] or CPSs [24]. Unlike all these approaches, which focus on testing approaches before the software version is deployed in production, our contribution aims at detecting performance issues at run-time, once the software system has been deployed. We opted by this approach because according to domain experts, most of the performance problems are exhibited in operation because the software is highly configurable. In the case of Orona's software, it needs to be configured for each building by considering several parameters, such as the number of floors, the number of elevators and other building specific parameters.

In this sense, we have identified some tools that address the challenge of detecting errors on the deployment process. Gandalf is a service whose objective is to detect errors on cloud rollouts to stop them before they cause major failures [17]. To this end, it continuously monitors a wide range of infrastructure data to detect anomalies and when any is detected it determines if the anomaly is caused by a rollout or not by means of correlations. Finally, it uses a Gaussian discriminant classifier to decide whether the impact caused by the deployment is significant enough to stop it. FUNNEL is a tool that collects performance metrics for each software change (i.e., software update or configuration change) to detect behaviour changes [28]. It uses a Singular Spectrum Transform (SST) algorithm for the detection and a Difference-in-Difference (DiD) method, where it compares the relative performance of the treated group and a control group to decide if the software change is the cause of the behaviour change.

Our approach aims at detecting performance issues in CPSs, which might have critical impacts as compared to other less critical systems (e.g., web applications, mobile apps, etc.). Balasubramaniyan et al. proposed a methodology to design and verify CPSs with the aim of optimizing their reliability and performance mostly focused on timing issues [3]. Song et al. proposed a virtual testing environment to assess performance of CPSs [24]. The tool can simulate several configurations in parallel and identify the ones that lead to best and worst performance based on user-defined performance indicators. Markoska et al. proposed a performance testing framework oriented to smart buildings, which is offered as a service in the cloud [19]. Among the performance metrics they consider, it includes power consumption or timing constraints. Unlike all these approaches, our approach is based on Machine Learning to identify bugs at operation-time when the software version has been deployed on the real target.

Machine learning has already been used to detect performance bugs in other contexts. Gulenko et al. evaluated a total of 13 classification algorithms used for anomaly detection on a cloud environment running Virtualized Network Function services [12]. The performance metrics monitored are processing related (e.g., CPU usage), memory related (Disk I/O) and network related (Network IO). This work concludes that Machine Learning algorithms were able to predict anomalies with high precision and recall values, with an average F1 score of 92%. Sauvanaud et al. proposed an anomaly detection system for virtualized cloud services [21] by monitoring both system metrics as well as virtual machine specific performance metrics. Unlike these approaches, which used classification Machine Learning algorithms, we propose using regression learning algorithms.

Hu et al. proposed a Machine Learning based resource usage prediction for grid computing environments [13]. Specifically, multiple Machine Learning techniques are evaluated and compared to predict metrics related to CPU, memory, disk, and network for resource allocation and load balancing. Wieder et al. proposed a QoS prediction approach where run-time monitoring and Machine Learning techniques are used to predict performance problems in the cloud [26]. In addition to infrastructure metrics as memory or CPU usage, the system proposed also monitors application-level metrics such as response time or number of logins. This work proposes combining Machine Learning classification methods (Random forest, decision tree and SVM) with time series analysis methods (AR, ARIMA and ETS) to improve the prediction capacity of the system. The main difference between these studies and ours is that their approach aims to predict future inconsistencies and adapt the system to withstand those conditions. Conversely, in our case, the approach is solely focused on identifying the performance issue and notifying to engineers in order them to take the appropriate correct action. Furthermore, we assess our approach by using an industrial case study along with performance mutation testing [8].

3 CASE STUDY: PERFORMANCE REQUIREMENTS IN ELEVATOR DISPATCHING ALGORITHMS

Elevators are complex CPSs composed of different subsystems that collaborate to transport passengers vertically in a building. Controllers are in charge of managing both the vertical (from floor to floor) and the horizontal movements (doors opening and closing) of a single elevator. The traffic master is the software system in charge of the coordination of the controllers to serve the floor calls requested by passengers. The main responsibilities of the traffic master include the execution of the dispatching algorithm (i.e., the allocation of passenger calls to any of the available cars) and the overall system signalling (registration of the calls, information to the passenger), but it can also carry out additional functionalities such as access control (i.e. permission for the passengers to access certain floors) or management of special operating modes.

The traffic dispatching algorithm is the software component that selects the optimal elevator to serve a specific landing call. Thus, this component is critical to ensure the Quality of Service (QoS) of the elevator installation because of many reasons. Firstly, the assignment of the landing calls has a direct impact on the average waiting time and overall journey time of the passengers. Secondly, it affects to the energy consumption or transport capacity of the elevators. The traffic algorithm constantly evolves to be adapted to particular installations, improving the assignment process by including new rules or using new techniques such as artificial intelligence. Additionally, new criteria for the assignment such as the number of stops, load balancing or energy consumption are usually required for some installations.

The traffic dispatching algorithm is executed periodically to allocate all the active floor calls. Depending on the algorithm, already existing floor call allocations can be reallocated (to a different car) to optimize the overall cost function. This means that the allocation process is highly dynamic as it depends on the current system context. This fact, alongside with other context situations such as highly demanding traffic profile or a big number of floors can derive in a high consumption for the limited resources available on the allocation of landing calls in time. In this sense, the traffic algorithm should allocate the complete set of active landing calls in a limited time frame (response time) to provide an acceptable QoS. Moreover, the traffic algorithm is executed within a task that shares computing resources with other tasks that provide above mentioned functionalities (signalling, access control, etc.). Thus, system performance monitoring is crucial to ensure a proper system QoS. If such performance decreases, the overall system QoS degrades, and additional hardware resources should be considered [4].

A special type of traffic algorithms in which performance is specially relevant is destination algorithms. Unlike conventional algorithms, in which many passengers share the same uplanding or downlanding call, in destination algorithms each passenger registers his own call. This "destination call" is composed by a landing call and a destination floor, and more importantly, it remains active until the destination is reached. Therefore, a destination algorithm requires managing each destination call individually. Additionally, this type of dispatchers provides extra functionalities such as access control, that could affect the response time of the algorithm. Thus, in high-population buildings, the amount of active calls can be huge and can severely affect the system performance.

When a new version of the algorithm is released, performance under different traffic conditions and installations must be checked to identify issues related to a poor implementation of new or improved functionalities. Usually, benchmark installations with different number of floors and elevators, and some theoretical passenger profiles are used to validate the new release. These profiles represent some traffic demands for different types of buildings (offices, residential, hotels, etc.) during different periods of the day. Most commonly used office profiles include, (1) Morning UpPeak, representing the entrance to office in the morning (2) LunchPeak, representing the lunchtime, where there is a mix of passengers entering and exiting the building and (3) DownPeak, representing the exit of the office in the afternoon. There are also full day profiles that represent the traffic in an office during a working day and therefore, include a morning UpPeak, LunchPeak at midday and Afternoon DownPeak complemented with some inter-floor traffic (passenger flow between different floors of the building) during the morning and the afternoon [4].

However, detecting performance problems that could compromise response time in a new release of the algorithm is a complex task due to the following factors. Firstly, some functionalities of the algorithm are only activated under certain traffic demands, keeping potential performance issues hidden. For example, parking of empty elevators to heavy floors is only activated when a big demand from that floor is given. Secondly, performance is highly dependent on installation specific factors: number of controllers, number of floors in the building, etc. Lastly, performance is highly dependent on the passenger flow of each building. In summary, poor performance can be exhibited in some installations or traffic conditions but not in others, which makes it difficult to validate the software system.

Reproducing all the real scenarios to analyse the potential performance issues in the laboratory is unfeasible due to the effort it would mean. Moreover, there is often a lack of actual traffic profile information and thus, it is not easy to reproduce the operation conditions of the final installation in the laboratory. Therefore, it is necessary to include monitoring and detection mechanisms in operation to facilitate the detection of potential performance problems in a particular installation. This way, when releasing a new version, potential performance issues can be detected automatically before they compromise response time, and eventually, a rollback to a previous version could be performed.

4 PERFORMANCE PROBLEMS PREDICTION METHOD

Figure 1 shows the overall architecture of the proposed approach. The proposed solution is divided into two main phases: (1) the training phase and (2) the performance bugs identification phase.

The idea of the training phase is to train a regression learning algorithm by using performance data collected from operation. During the training phase (a) the current software version remains in operation and (b) the data generated is used to adapt the Machine Learning algorithm's internal parameters, so that its performance improves on future unseen input data. To train the Machine Learning algorithm, the data is categorized in two groups: the features, including the inputs of the studied piece of software and the label, with the measures of the performance metric we want to predict.

When the regression learning algorithm is trained, it yields a trained regression model, which is used in the performance problems identification phase to identify performance issues after a new release has been deployed remotely. This phase has three steps: (a) execution of the new release to collect the input data and performance metrics, (b) prediction by the regression model, which yields the expected performance bases on the input values, and (c) the arbitration process, which compares the expected performance obtained by the regression algorithm with the actual metrics collected during execution.

With this method, we aim at detecting performance issues that depend on the inputs of a software. We now explain the methodology and the developed implementation more in detail.

4.1 Performance metrics

The response time is the time required by the algorithm to assign a set of landing calls and communicate them to the elevators. The algorithm is invoked periodically and must provide a valid assignment every cycle. In every cycle, the algorithm receives all the information about the status of the elevators (e.g., position, velocity, doors status, etc.) and a set of car and landing calls. Car calls are assigned to a particular car and represent the destination of the passengers. In Orona's algorithm, car calls are not assigned by the algorithm, they are input information to decide the assignment of landing calls. Landing calls represent the origin of the passengers, and therefore, it is the algorithm's responsibility to decide which car is the most suitable to attend the landing calls.

There are two types of landing calls: conventional calls and destination calls. In destination algorithms, a call is registered by each passenger. Within an installation, the number of active calls impacts the response time of the algorithm. The lunch time, for example, is one of the most demanding periods for the algorithm in office buildings. During this period, there is a mix of passengers entering and exiting the building. During LunchPeak and, for example, during the morning UpPeak, the processing load of the algorithm is high. In addition to the processing of a high amount of calls, some other mechanisms such as parking or control of long waiting times are activated, which also increases the processing time.

Specially in destination algorithms, where the attendance to the passengers is individualized, the number of active calls influences the response time. Active calls can be in different states: (1) Registered: first cycle in which the algorithm receives a call and the assignment remains pending. (2) Assigned: the algorithm has already selected an elevator to attend the call but the elevator has not still arrived to answer the call. (3) Answered: the elevator has arrived at the origin of the passenger and the passenger is travelling to his destination. Finally, once the passenger arrives to his destination, the call is cancelled.

In this context, the performance metric that has been monitored is the response time of the periodic task that executes the dispatching algorithm. That is, the time frame in which the algorithm dispatches (allocates) the active calls to the available cars. This response time is directly related to the existing traffic: an increase in passenger traffic causes a rise in the number of active calls and this results in a higher demand of computing resources to allocate these calls. As a consequence, the response time of the task executing the algorithm increases and, eventually, a deadline loss can occur. Thus, we have chosen the number of active calls as the value to predict the response time of the algorithm. To measure the response time of the task, the algorithm code has been instrumented with high precision clocks provided by the operating system.

4.2 Regression learning algorithms

Machine Learning (ML) algorithms are a subset of Artificial Intelligence algorithms. They provide systems the ability to automatically learn and improve from past experiences without being explicitly programmed to. ML algorithms can be catalogued into two categories: supervised and unsupervised learning. While supervised learning algorithms aim to map input objects and output variables from labelled training data, unsupervised learning algorithms are

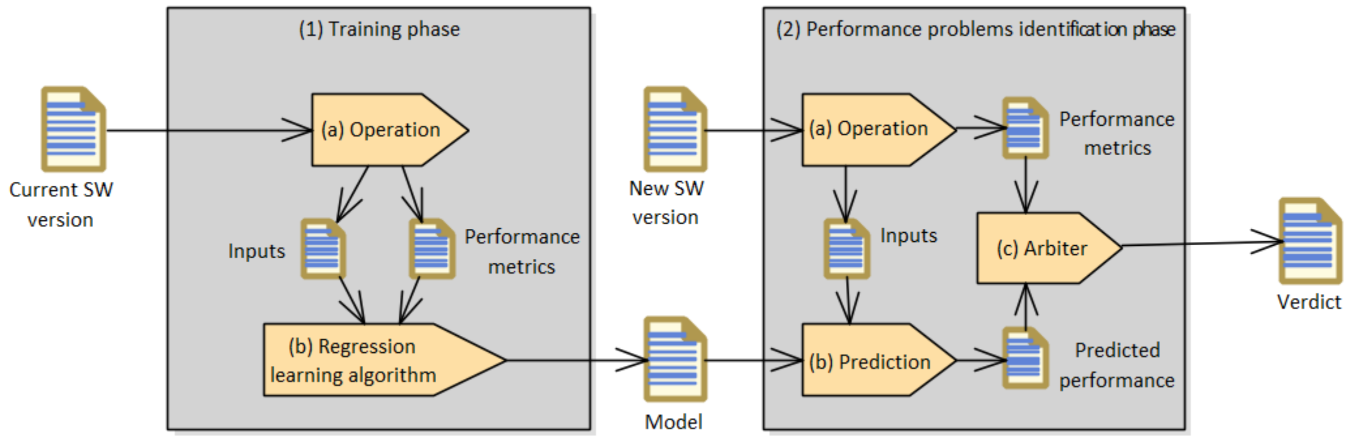


Figure 1: Architecture of the performance problem prediction method

trained based on input data only, acting as a clustering technique [7]. Supervised learning techniques can be categorized both in regression and classification techniques. Regression algorithms aim to map inputs to real-valued outputs (i.e., a number), whereas classification aims to map inputs into categorical outcomes [7]. Our approach uses regression learning algorithms because it is necessary to predict a quantitative value to detect a performance issue. The regression algorithms evaluated in this paper have been TRGP, Regression Tree, Ensemble, Stepwiselm and SVM. The latter is commonly used as a classifier but has also proven to be useful as a regression algorithm [6].

4.3 Training phase

In this phase, the input feature will be the number of active calls, labelled with the response time for each cycle of the algorithm every minute. To do so, we developed a script that automatically extracts this data from a database. Once, the data is extracted, the script launches the training phase by using the MATLAB Machine Learning toolbox. The regression learning algorithm yields a trained regression model, which can later be used in the testing phase to predict the response time of the algorithm based on the number of active calls.

4.4 Prediction phase

When a new version of the algorithm is released to operation, data of active calls and response time of the algorithm are collected every minute. Response time has to be less than the deadline of the task that executes the traffic algorithm. This is checked with a rule. However, other performance problems that can be indicators of bad implementations are more difficult to detect due to the dependence of the response time to the characteristics of the installation, the functionalities activated in the algorithm and the traffic flow. Therefore, active calls and response time measurements are collected to compare them with predicted data.

The execution of the new release provides the number of active calls every minute to the trained regression model. This model, estimates the response time over time based on the training produced during the training phase.

The arbitration process is in charge of detecting performance problems in a new release of the software. By comparing the data that is being collected from the execution of the new release and the prediction of the regression model, this module provides a verdict. The arbitration process has been designed considering three criteria: (1) average response time (2) maximum response time and (3) variance in response time. These criteria have been analyzed in two time-frames: (1) daily, with the response times of the algorithm in a whole day traffic flow and (2) every minute, with the traffic of the previous 5 minutes. Experts from Orona have set the tolerable variations in the variance, average and maximum response times that could be considered normal in the execution of the algorithm. In addition the variation percentages that should be classified as performance problems within the two time-frames were specified by the same domain experts. The arbitration process inputs the real response time of the new release and compares it to the prediction of the regression model. When an abnormal variation is detected in any of the criteria, it must be analyzed, and having the minute by minute verdict facilitates the detection of the performance problem, as it allows analyzing under which traffic conditions is the problem shown.

5 EMPIRICAL EVALUATION

This section evaluates the proposed approach by using an industrial case study. To this end, we aimed at answering the following Research Questions (RQs):

- RQ1: How does each of the selected regression learning algorithms perform for predicting performance problems on software updates?
- RQ2: Are there any differences when training the machine-learning algorithms with real field data or theoretical data?

5.1 Experimental setup

This subsection explains the proposed experimental design to answer the aforementioned RQs.

5.1.1 Case study. As a case study, we used the dispatching algorithm of Orona developed in C/C++. This algorithm is the component inside an elevator in charge of assigning a specific elevator to each call. Performance on this algorithm is critical, as it must provide a response in a limited time frame. Therefore, it is important that when software updates are performed in the dispatching algorithm of Orona, these are free of performance issues. The type of algorithm used for the evaluation is a destination algorithm and, therefore, there is one call per each passenger. For the evaluation, two types of passenger profiles have been used: full day theoretical profile, an actual traffic profile from an office building in Paris obtained from literature [23], and full day profile collected from a real building of Orona. The real building is a 10-floor office building with six elevators in it and the entrance in the ground floor.

5.1.2 Evaluation metrics. We used performance mutation testing to evaluate the approach [8]. To this end, a total of 45 performance mutants were generated by following the performance mutation operators proposed by Delgado et al. [8]. Mutation testing aims to generate a set of versions from the original program and adding a synthetic variation on it. When the outcomes of a test differ from the mutant to those of the original version, it is considered that the mutant is killed. This technique has been found to be a good substitute of real faults [15]. The difference between traditional mutation testing and performance mutation testing is that the mutation operators for the latter are focused on injecting performance problems and keeping the original functionality of the program [8], whereas the former focuses on changing the program functionality. To consider a mutant killed in performance mutation testing, performance service metrics are considered, such as execution time or memory usage [8]. We have systematically created mutants that could affect performance based on the performance mutation operators proposed in [8], which include: method call, loop perturbation and conditional execution. Specifically, we have created mutants by: simulating heavy operation in different parts of the algorithm, Move/Copy Statement into Loop, Removal of Stop Condition in Loop and Unnecessary Calculation of values. In this paper, we focused on the response time of the dispatching system as a measure for detecting performance problems.

Similar to related studies [5, 11], we used the precision, recall, accuracy and F-measure metrics to measure the performance of the proposed regression learning algorithms. We also developed a regression test oracle, which involves the original dispatching algorithm. When executing the tests, we obtained the running times over time of both the original dispatching algorithm as well as the one from the mutants. The running time from the dispatching was provided as input to the arbiter, forming this way the regression test oracle to classify the test as pass or fail without the prediction part, which is the core of this study.[1] This classification was catalogued as a True Positive (TP), True Negative (TN), False Negative (FN) or False Positive (FP) as defined below:

- TN: Both our approach and the regression test oracle returned a "PASS" verdict.
- TP: Both our approach and the regression test oracle returned a "FAIL" verdict.
- FN: Our approach returned a "PASS" and the regression test oracle returned a "FAIL".
- FP: Our approach returned a "FAIL" and the regression test oracle returned a "PASS".

When mutants were catalogued as TN, TP, FN or FP, the precision, recall, F1 and accuracy were obtained following Equations 1, 2, 3 and 4.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = \frac{2 \times (precision \times recall)}{precision + recall} \tag{3}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

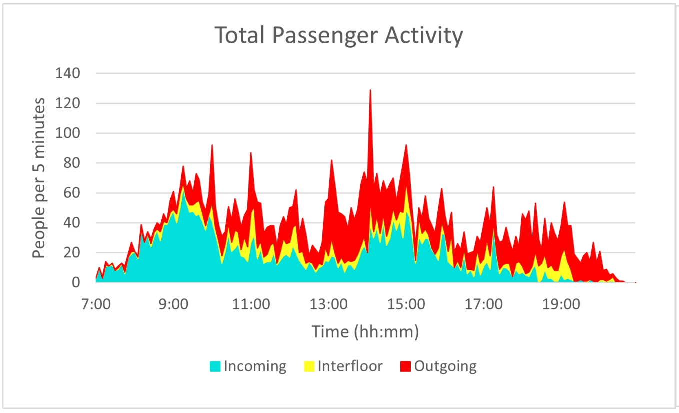
5.1.3 Training strategy. To perform the training of the model we have established three different scenarios. In the first scenario, the training was performed with four test cases, which were obtained from field data in a real office installation maintained by Orona (the same real office installation used in our evaluation). We had access to real passenger flows for 4 different days in an office building with 6 elevators and 10 floors. Therefore, we have been able to use this data to execute the algorithm and obtain the response time of these days. We have executed the passenger profiles 5 times as the execution time varies slightly depending on the other tasks of the dispatcher. We have used a destination dispatcher that requires an individualized treatment of each active call. Figure 2a illustrates the traffic collected in a working day in the office building. During the morning, most passengers go from the ground floor to the upper floors of the building, from 13:00 to 15:00 there is a mix of traffic going from and to the ground floor, and at the end of the working day, most passengers' destination is the ground floor.

In the second scenario, the training was performed with four test cases, which were obtained from Elevate™[2], a passenger traffic simulation tool. The used profile was Siikonen full day traffic profile that is based on a sample multi-tenant office building in Paris [23]. Each test case was executed 5 times as response time can differ slightly in different executions. Figure 2b illustrates the traffic flow of the siikonen profile.

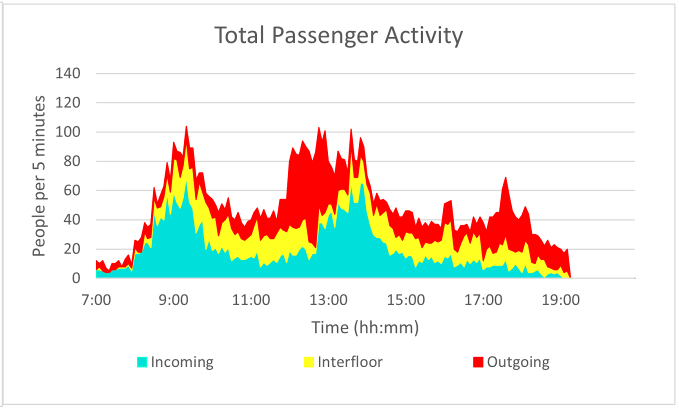
In the third scenario, both of the training sets from the previous scenarios have been used. Therefore, four profiles collected from the field and four profiles collected from the theoretical profiles where used to train the model. Each of them was executed 5 times. The analysis of these three scenarios provided us information about how important is to obtain real field data to predict performance.

[1]Notice that in practice, in operation, we lack this regression test oracle, and subsequently, it is only used for our evaluation

[2]https://www.peters-research.com/index.php/elevate



(a) Full day profile collected from real installation by Orona



(b) Siikonen Full day profile

Figure 2: Total passenger activity of real installation and theoretical profiles obtained with Elevate

5.1.4 Performance problems identification. As the goal is to identify performance problems in operation, we have used the field profiles as input for the prediction model. In the cases were field data was used to train the algorithms, to avoid bias, we used the k-fold cross validation. This means that we did not use the same dataset for training as well as for testing.

5.2 Analysis of the Results

Table 1 summarizes the obtained results of the selected regression algorithms for the three considered scenarios. The first scenario aimed at comparing the performance of the selected algorithms when trained with field data. For the five regression learning techniques, TRGP, Regression Tree and Ensemble showed the strongest results. These techniques had a precision of around 0.95, a recall of around 0.93, an accuracy of around 0.93 and an F1 measure of around 0.94. Conversely, SVM and Stepwiselm algorithms showed overall worse results. Despite having a high recall, their precision, accuracy and F1 were significantly lower than the rest of the algorithms.

Since field data is not always available, the second scenario aimed at investigating how the different algorithms performed with theoretical test data. Similar to the previous scenario, TRGP, Regression Tree and Ensemble were the algorithms showing the best results. SVM and Stepwiselm showed better recall measures but lower precision, accuracy and F1 measure, similar to the first scenario. It is noteworthy, however, that for the best algorithms (i.e., TRGP, Regression Tree and Ensemble), the recall, accuracy and F1 measures were dropped with respect to the previous scenario, although the precision increased. This means that when training with theoretical data, the number of false negatives slightly increased, whereas the number of false positives decreased.

The third scenario aimed at investigating how the different algorithms performed with a mixture field and theoretical test data. Similar to the previous scenarios, TRGP, Regression Tree and Ensemble were the algorithms showing best results. However, as in the second scenario, they showed lower recall, accuracy and F1 measures when compared to those results from the first scenario (i.e., when the algorithms were trained with field data). Nevertheless, their values were slightly higher than those obtained when trained solely with theoretical data (i.e., second scenario). Again, for this scenario, SVM and Stepwiselm showed low precision, accuracy and F1 values.

5.3 Discussion

From the results, it can be seen that Regression Tree, Ensemble and TRGP are overall algorithms showing best results. On the downside, SVM and Stepwiselm showed the worst precision, accuracy and F1 values for the three scenarios, despite having a higher recall value. This means that there were a high number of false positives, which results in a high number of mutants catalogued as failing when they should have been catalogued as non-failing. An insight behind the obtained results is that the problem to solve is non-trivial, as the SVM, which is commonly taken as a baseline algorithm to determine the difficulty of the problem, yields bad results. We can thus answer the first RQ as follows:

Overall, Regression Tree, Ensemble and TRGP are the best algorithms for predicting performance problems in software updates in elevator dispatching algorithms, while SVM and Stepwiselm are not valid.

Results for TRGP, Regression Tree and Ensemble seem promising when trained with field data, although the use of theoretical test data for training them seems to distort the recall, accuracy and F1 measures of the algorithms. When having a close view on the training sets, we figured out that the training data in theoretical profiles are significantly different to the data used in the real installation. Specifically, this led not to train certain areas that covered scenarios appearing in all real installations. Instead, the theoretical profiles focused on stressing the system, increasing significantly the number of active calls at certain times during the full day. Conversely, the real traffic profiles had a more constant passenger flow signals, without strong peaks, having an average lower number of active calls per minute, as can be seen in Figure 2. This is specially remarkable during morning UpPeak and midday LunchPeak.

Table 1: Results summary for the different scenarios of our approach [Scenario 1, Scenario 2, Scenario 3]

	TRGP	REGRESSION TREE	ENSEMBLE	SVM	STEPWISELM
PRECISION	[0.95, 1.00, 1.00]	[0.95, 0.99, 0.97]	[0.96, 1.00, 1.00]	[0.58, 0.61, 0.58]	[0.62, 0.60, 0.59]
RECALL	[0.94, 0.67, 0.75]	[0.94, 0.70, 0.73]	[0.92, 0.65, 0.73]	[1.00, 1.00, 1.00]	[1.00, 1.00, 1.00]
ACCURACY	[0.94, 0.82, 0.87]	[0.94, 0.83, 0.84]	[0.94, 0.81, 0.85]	[0.58, 0.61, 0.58]	[0.65, 0.60, 0.59]
F1	[0.95, 0.81, 0.86]	[0.95, 0.82, 0.84]	[0.94, 0.79, 0.85]	[0.74, 0.76, 0.74]	[0.77, 0.75, 0.74]

In our evaluation we show again the differences between theory and practice, instantiated in a real-life example in a widely used domain, i.e., the vertical transport domain. Many assumptions are made in theory to explain the phenomenon and concepts, yet, in real-life, assumptions and conditions do not always hold and are not unique. In areas like web or mobile engineering, Design-Operation Continuum methods (e.g., DevOps) are widely used, bringing advantages like testing "on-the-fly" or taking data from operation to development-time for a more extensive analysis. In our study, we show that this concept is required to be extended to wider engineering areas, such as the ones related to embedded and Cyber-Physical Systems (CPS).

Having discussed this, we can answer the RQ2 as follows:

Results significantly differ depending on the training type used. Training them with field data is the best option, whereas the use of theoretical test data shall be further analyzed.

5.4 Lessons learned

The results show that collecting information from the operation of the traffic algorithm can be used to detect performance problems when a new version is released. Usually, changes in the software of the algorithm consists on (1) including new strategies to improve the QoS of the algorithm, (2) adapting existing functionalities to special installations (e.g., multi entrance floors building), (3) extending the algorithm to consider new assignment rules or (4) adapting new legislative changes. Engineers in charge of the updates and extensions are not always original developers of the algorithm, therefore, misunderstanding about the usage of data structures and control structures is not uncommon. In addition, some changes must be performed quickly to support operational installations.

The algorithm includes complex control structures to manage the floors, controllers and calls and an inefficient implementation of these structures can result in performance problems in some installations or under some kind of traffic flow. To check the functional behaviour of a new version, a simulator named Elevate is used. In simulation, several validations can be performed semi-automatically. This provides a good confidence level that the updates will behave as expected. However, an exhaustive checking of the performance of the new release is usually not feasible. The performance is dependent on the features of the installations and the traffic profile, and requires real time validation, therefore, the number of executions that can be run in the laboratory are reduced. Consequently, usually static performance analysis is used to guarantee the response time under the worst scenario.

The method presented in this paper uses previous information collected in the installations to detect performance problems that can be exhibited during operation. The results show that regression learning algorithms can be used to train a model to predict the response time of an installation. This information can be used to identify performance problems on the release of a new version by using rules provided by the domain experts. The number of active calls has proven to be a good input factor for the model. By using this method in installations of Orona, an overview of the performance problems detected in different installations can be obtained. By analyzing this information, implementation inefficiencies that have a negative impact in performance can be detected. From the experiment carried out on the first scenario we obtain that Regression Tree results in 5 false positive out of 180. The type of mutants that result in false positives are two mutants simulating heavy operation in parts of the algorithm that are activated conditionally, two related to the Removal of Stop Condition in Loop and one of Unnecessary Calculation of values. Regarding false negatives, 6 out of 180 were detected. Overall, 94% of the tests provided a good results.

We highly recommend using this method systematically especially in office buildings and hospitals where the traffic is more demanding than in the residential sector and performance problems have a higher probability of being exhibited. It is important to remark that the analysis shows that the training should be specific for a particular installation.

This would imply: (1) continuously collecting the number of active calls and response time during the normal operation of the dispatcher in the installations, (2) training the model with this information and (3) integrating in the dispatcher the model and the rules to identify performance problems. To continuously monitor the active calls and response time, a persistence mechanism shall be provided. Due to performance reasons, we think that the training phase shall be performed at the laboratory, not in the installations. Therefore, data collected in the installation shall be sent to the laboratory for analysis.

Figure 3 depicts the physical infrastructure needed to accomplish the proposed approach, i.e., the deployment of the architecture proposed in Fig.1. Oronas cloud could be used as a bridge both for deployment and telemetry purposes. The software developed in the lab would be remotely deployed to the real installation using a microservice based approach. Due to security issues, the real installation equipment should not have open ports and thus, a notification-based schema based on MQTT is proposed to alert the edge gateway that a new version is available in Oronas docker registry. This way, the edge gateway downloads and installs the new software in the traffic master. Besides that, in operation, the

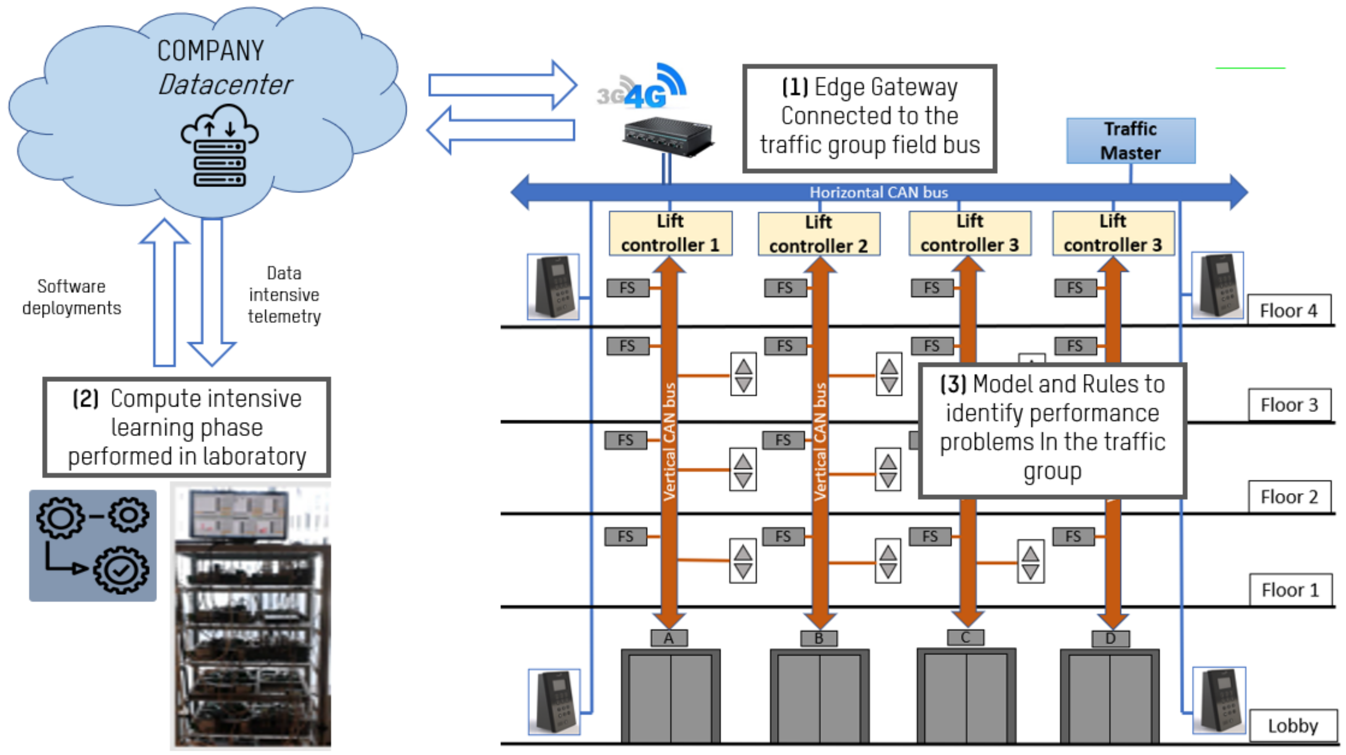


Figure 3: Physical infrastructure to carry out the proposed approach

edge gateway collects the required operational data coming from the real installation and feeds it back to the laboratory for training purposes, closing the loop and enabling a continuous validation schema in operation.

We believe that in office buildings sending the information weekly could be sufficient. In order to select the input data to train the algorithm, the type of building shall be taken into account. Traffic flow in office buildings can be dependent on the season (for example, vacation periods, etc.). Besides, in multi-tenant buildings, traffic flow can be more variable over the time than in single tenant buildings. Therefore, the option of having a different training according to the season or features shall be further analyzed.

Lastly, we envision two alternatives for the identification of performance problems: (1) execute in the laboratory and (2) execute with the dispatcher in the installation. The verdict to identify a performance problem in this version uses the information of the response time of a full day and every 5 minutes. This could be done either in the laboratory or in the execution in the installation. In this case, we think that the best option is to perform this process in the installation as it allows to detect performance problems at operation time and take contingency actions if needed.

5.5 Threats to validity

Internal validity: Our evaluation is subject to some internal threats. One such threat relates to the configurations of the used Machine Learning algorithms. To reduce this threat, we used the default parameters provided by the MATLAB framework to train the algorithms. In addition, the arbiter uses certain thresholds that require configurations. The value of these parameters could significantly change the results of our evaluation. To reduce this threat, these values were discussed with domain experts.

External validity: A potential external validity threat in our evaluation relates to the used benchmark dataset and to only using a single case study. However, this case study is a real industrial case study of high complexity. Furthermore, the dataset is the one used by Orona to test their dispatching algorithms. We are thus using a benchmark with real industrial data and within a real-world setting. Another external validity threat in our approach could relate to how the training was performed. To avoid bias in the results, we did not use the same dataset for training and for testing an algorithm. In those cases were necessary, the k-fold cross validation techniques were used.

6 CONCLUSIONS AND FUTURE WORK

This paper has investigated the use of Machine Learning techniques to detect performance problems after new software release deployments. Then, proposed approach has been tested in an industrial use case, specifically in elevator traffic dispatching algorithms, in collaboration with Orona. The conclusions obtained from the experiments conducted, are the following:

- We have proven that applying regression learners to data collected on elevators installation in order to predict response time is a valid mechanism to detect performance problems.

- The method does not consider time-critical concerns, so may be only applicable to non time-critical CPSs. The use of the method in a time-critical context requires further analysis.
- The regression algorithms which showed the best results for all the experimental setups where TRGP, Regression Tree and Ensemble, while SVM and Stepwiselm have proved to be the least appropriate.
- The best way to train a regression algorithm to predict performance problems in dispatching algorithms is to train them with real field data from the installation, rather than theoretical profiles.

In future, we plan to continue this research by evaluating the effectiveness of this method in its application on different types of installations. In addition, we shall investigate the different training strategies. In this sense, two aspects must be taken into account are (1) the frequency of the training, as people flow is dynamic over the time and (2) the need of training the model with different training sets depending on the season of the year, as traffic may change depending on vacation periods, etc. Lastly, the use of different theoretical traffic profiles will be further analyzed. In addition to the full day profiles used in this work to evaluate the approach, the use of UpPeak, LunchPeak, DownPeak and Interfloor theoretical profiles will be further investigated to analyze whether they can be used to predict the performance.

ACKNOWLEDGMENT

This publication is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871319. This work has been partially supported by the Basque Government through the Elkartek program under the DIGITAL project (Grant agreement no. KK/2019-00095). Aitor Arrieta and Goiuria Sagardui are part of the Software and Systems Engineering research group of Mondragon Unibertsitatea (IT1326-19), supported by the Department of Education, Universities and Research of the Basque Country.

REFERENCES

[1] Jon Ayerdi, Aitor Garciandia, Aitor Arrieta, Wasif Afzal, Eduard Enoiu, and Aitor Agirre. Towards a Taxonomy for Eliciting Design-Operation Continuum Requirements of Cyber-Physical Systems. In *IEEE 28th International Requirements Engineering Conference*. IEEE, 2020.

[2] Jon Ayerdi, Sergio Segura, Aitor Arrieta, Goiuria Sagardui, and Maite Arratibel. Qos-aware metamorphic testing: An elevation case study. In *International Symposium on Software Reliability Engineering (ISSRE 2020)*. IEEE, 2020.

[3] Sreram Balasubramaniyan, Seshadhri Srinivasan, Furio Buonopane, B. Subathra, Jüri Vain, and Srini Ramaswamy. Design and verification of Cyber-Physical Systems using TrueTime, evolutionary optimization and UPPAAL. *Microprocessors and Microsystems*, 42(2016):37–48, 2016.

[4] G. C. Barney. 2003.

[5] Wing Kwong Chan, Jeffrey CF Ho, and TH Tse. Finding failures from passed test cases: Improving the pattern classification approach to the testing of mesh simplification programs. *Software Testing, Verification and Reliability*, 20(2):89–120, 2010.

[6] Vladimir Cherkassky and Yunqian Ma. Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, 17(1):113–126, 2004.

[7] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.

[8] Pedro Delgado-Pérez, Ana Belén Sánchez, Sergio Segura, and Inmaculada Medina-Bulo. Performance mutation testing. *Software Testing Verification and Reliability*, 2020.

[9] Giovanni Denaro, Andrea Polini, and Wolfgang Emmerich. Early performance testing of distributed software applications. *Proceedings of the Fourth International Workshop on Software and Performance, WOSP'04*, pages 94–103, 2004.

[10] Vincenzo Ferme and Cesare Pautasso. Towards holistic continuous software performance assessment. *ICPE 2017 - Companion of the 2017 ACM/SPEC International Conference on Performance Engineering*, pages 159–164, 2017.

[11] Ahmet Esat Genç, Hasan Sözer, M Furkan Kıraç, and Barış Aktemur. Advisor: An adjustable framework for test oracle automation of visual output systems. *IEEE Transactions on Reliability*, 2019.

[12] Anton Gulenko, Marcel Wallschlager, Florian Schmidt, Odej Kao, and Feng Liu. Evaluating machine learning algorithms for anomaly detection in clouds. *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*, pages 2716–2721, 2016.

[13] Liang Hu, Xi Long Che, and Si Qing Zheng. Online system for grid resource monitoring and machine learning-based prediction. *IEEE Transactions on Parallel and Distributed Systems*, 23:134–145, 2012.

[14] Guoliang Jin, Linhai Song, Xiaoming Shi, Joel Scherpelz, and Shan Lu. Understanding and detecting real-world performance bugs. *ACM SIGPLAN Notices*, 47:77–87, 2012.

[15] René Just, Darioush Jalali, Laura Inozemtseva, Michael D Ernst, Reid Holmes, and Gordon Fraser. Are mutants a valid substitute for real faults in software testing? In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 654–665. ACM, 2014.

[16] Rakesh Kumar Lenka, Pranali Bhanse, and Utkalika Satapathy. Load performance testing on cloud platform. *Proceedings - IEEE 2018 International Conference on Advances in Computing, Communication Control and Networking, ICACCCN 2018*, pages 414–419, 2018.

[17] Ze Li, Qian Cheng, Ken Hsieh, Yingnong Dang, Microsoft Azure, Peng Huang, Johns Hopkins University, Pankaj Singh, Xinsheng Yang, Qingwei Lin, Microsoft Research, Youjiang Wu, Sebastien Levy, and Murali Chintalapati. Gandalf: An Intelligent, End-To-End Analytics Service for Safe Deployment in Large-Scale Cloud Infrastructure. *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020.

[18] Yepang Liu, Chang Xu, and Shing Chi Cheung. Characterizing and detecting performance bugs for smartphone applications. *Proceedings - International Conference on Software Engineering*, (1):1013–1024, 2014.

[19] Elena Markoska and Sanja Lazarova-Molnar. Towards smart buildings performance testing as a service. *2018 3rd International Conference on Fog and Mobile Edge Computing, FMEC 2018*, pages 277–282, 2018.

[20] Oswaldo Olivo, Isil Dillig, and Calvin Lin. Static detection of asymptotic performance bugs in collection traversals. *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2015-June:369–378, 2015.

[21] Carla Sauvanaud, Mohamed Kaâniche, Karama Kanoun, Kahina Lazri, and Guthemberg Da Silva Silvestre. Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned. *Journal of Systems and Software*, 139:84–106, 2018.

[22] Sergio Segura, Javier Troya, Amador Durán, and Antonio Ruiz-Cortés. Performance metamorphic testing: motivation and challenges. In *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track*, pages 7–10. IEEE Press, 2017.

[23] ML Siikonen. On traffic planning methodology. *Lift Report*, (March), 2001.

[24] Zhen Song, Philippe Labalette, Robin Burger, Wolfram Klein, Sudev Nair, Suhas Suresh, Ling Shen, and Arquimedes Canedo. Model-based cyber-physical system integration in the process industry. *IEEE International Conference on Automation Science and Engineering*, 2015-October(September 2016):1012–1017, 2015.

[25] Elaine J. Weyuker. Experience with performance testing of software systems: issues, an approach, and case study. *IEEE Transactions on Software Engineering*, 26:1147–1156, 2000.

[26] Philipp Wieder, Edwin Yaqub, Ramin Yahyapour, and Ali Imran Jehangiri. Distributed predictive performance anomaly detection for virtualised platforms. *International Journal of High Performance Computing and Networking*, 11:279, 2018.

[27] Chen Zhang, Jiaxin Li, Dongsheng Li, and Xicheng Lu. Understanding and Statically Detecting Synchronization Performance Bugs in Distributed Cloud Systems. *IEEE Access*, 7:99123–99135, 2019.

[28] Shenglin Zhang, Ying Liu, Dan Pei, Yu Chen, Xianping Qu, Shimin Tao, Zhi Zang, Xiaowei Jing, and Mei Feng. FUNNEL: Assessing Software Changes in Web-Based Services. *IEEE Transactions on Services Computing*, 11(1):34–48, 2018.