computers

MDPI

Article

A Taxonomy of Techniques for SLO Failure Prediction in Software Systems

Johannes Grohmann [1,*], Nikolas Herbst [1], Avi Chalbani [2], Yair Arian [2], Noam Peretz [2] and Samuel Kounev [1]

[1] Chair of Software Engineering, University of Würzburg, Am Hubland, 97074 Würzburg, Germany; johannes.grohmann@uni-wuerzburg.de (J.G.); nikolas.herbst@uni-wuerzburg.de (N.H.); samuel.kounev@uni-wuerzburg.de (S.K.)

[2] Tel-Aviv Yafo research center for Huawei Technologies, 45101 Hod Hasharon, Israel; avi.chalbani@huawei.com (A.C.); yair.arian@huawei.com (Y.A.); noam.peretz@huawei.com (N.P.)

* Correspondence: johannes.grohmann@uni-wuerzburg.de;

Received: 31 December 2019; Accepted: 5 February 2020; Published: 11 February 2020

Abstract: Failure prediction is an important aspect of self-aware computing systems. Therefore, a multitude of different approaches has been proposed in the literature over the past few years. In this work, we propose a taxonomy for organizing works focusing on the prediction of Service Level Objective (SLO) failures. Our taxonomy classifies related work along the dimensions of the prediction target (e.g., anomaly detection, performance prediction, or failure prediction), the time horizon (e.g., detection or prediction, online or offline application), and the applied modeling type (e.g., time series forecasting, machine learning, or queueing theory). The classification is derived based on a systematic mapping of relevant papers in the area. Additionally, we give an overview of different techniques in each sub-group and address remaining challenges in order to guide future research.

Keywords: taxonomy; survey; failure prediction; anomaly prediction; anomaly detection; self-aware computing; self-adaptive systems; performance prediction

1. Introduction

Self-aware computing systems are defined as software systems that (1) learn models capturing knowledge about themselves and their environment (such as their structure, design, state, possible actions, and run-time behavior) on an ongoing basis, and (2) reason using the models (for example predict, analyze, consider, plan), enabling them to act based on their knowledge and reasoning in accordance with their higher-level goals [1].

Therefore, the techniques and algorithms employed in these self-aware computing systems are often required to perform failure prediction as part of the *predict* or the *analyze* phase during reasoning. Often, these techniques have to be applied on an ongoing basis or in an online fashion as the system itself is also applied in a changing environment, and the employed models change continuously.

Hence, a variety of different algorithms for the prediction of Service Level Objective (SLO) failures, i.e., the inability to comply with user-defined service quality goals, has been proposed in the literature, based on different techniques, like time-series forecasting, machine learning or queueing theory, all of which were designed with a specific use case in mind, and offering a few advantages and disadvantages over related approaches.

In this paper, we propose a taxonomy of different approaches for SLO failure prediction grouping them according to three dimensions: (1) the prediction target (e.g., anomaly detection, performance prediction, or failure prediction); (2) the time horizon (e.g., detection or prediction, online or offline application); and (3) the applied modeling type, i.e., the underlying base technique that was utilized

(e.g., time series forecasting, machine learning, or queueing theory). We furthermore conduct a systematic mapping of related works in the proposed taxonomy and in order to validate the proposed taxonomy. Finally, in Section 6, we conclude by addressing some open issues and research challenges that need to be addressed by future works, in our opinion.

The proposed taxonomy provides an overview as well as a guideline for researchers working in the field of self-aware computing or other research areas that regularly employ failure detection techniques. The systematic mapping can be used as a guideline if a technique offering certain characteristics based on our taxonomy dimensions is required. Furthermore, when developing a novel technique, our taxonomy aids at quickly identifying key characteristics and advantages of the new approach. Finally, the open challenges presented in Section 6 offer an opportunity to guide the academic research towards these remaining issues.

The remainder of this work is structured as follows. In Section 2, we delimit ourselves from other surveys approaches from related work. Then, Section 3 introduces the methodology of the systematic mapping that leads to the development of the taxonomy presented in Section 4. Section 5 presents the results of the systematic mapping in more detail. Finally, we address open research challenges in Section 6 and limitations to our study in Section 7, and concludes our work in Section 8.

2. Related Work

Multiple works targeting a survey of techniques for online failure prediction exist. A comprehensive survey about different failure prediction methods in an online context is given by Salfner et al. [2]. However, they only focus on the online prediction of failures, and also apply a different definition of failure, as we specifically focus on SLO failures in this work. Furthermore, there exists a set of surveys covering different partial aspects of our taxonomy. In addition to listing them here, we name them in their respective sections during the presentation of the survey results in Section 5. As we tried to avoid duplicating work from the respective studies, each of the studies might present a more detailed overview of the respective sub-field. However, please note that some of the surveys might include other works that do not fit in the taxonomy proposed by us. Chandola et al. [3] present a comprehensive survey focusing on anomaly detection approaches. Amiri and Mohammad-Khanli [4] give a further overview of performance prediction approaches. Their focus is on resource provisioning in the cloud. Witt et al. [5] survey Predictive Performance Modeling (PPM) approaches in the area of distributed computing and give an overview of the state-of-the-art in that field. Koziolek [6] gives an overview of performance prediction and measurement approaches with a focus on component-based software systems.

Márquez-Chamorro et al. [7] present a survey of so-called *predictive monitoring* for business processes. Their notion of predictive monitoring, in this case, includes the prediction of future events and is therefore also related to our work. However, their focus is on log-based process mining techniques for business processes. Weingärtner et al. [8] discuss different approaches for application profiling and forecasting and discuss both activities in the context of the Monitor-Analyze-Plan-Execute-Knowledge (MAPE-K) loop [9]. Although this work has a slightly different focus than our work, application profiling is also an important step in many approaches for SLO failure prediction. Their focus is on cloud resource management.

All of the enumerated surveys cover their respective sub-field in convincing depth. However, as none of these works follow the definition of *SLO failure* as we propose it, the covered respective fields are not as broad as in this work as the term *failure* can be interpreted in many different ways. Additionally, none of them related surveys focused on self-aware computing systems.

3. Methodology

This section describes the applied methodology we used for analyzing the research area. Due to the variety of different terms and terminologies applied in the literature, it is unfortunately not possible for us to conduct a systematic and exhaustive literature review based on a set of pre-defined search

term as recommended by Webster and Watson [10] as well as Kitchenham and Charters [11], since all comprehensive queries would result in too many (>5000) irrelevant search results. Therefore, the also well-known method of systematic mapping was applied [12]. We started with a set of initial literature and performed forward and backward passes on those in order to identify more works of the respective area. This procedure is known as Berry Picking with footnote chasing and backward chaining [13]. We filtered potential works by manually reading title, abstract, and keywords in order to include the context as opposed to applying an automated keyword-based mechanism. Summarizing, we included a total of 67 [14–80] scientific papers, eight theses [81–88], three tech-reports [89–91], and two US patents [92,93] for the derivation of the taxonomy presented in Section 4. After refining the taxonomy, we re-classified all works to see if each work is still in the right category and then continued to classify all remaining works by reading their approach in detail. This procedure results in an unbalanced taxonomy tree. However, this is due to the imbalance in the distribution of the underlying works.

4. Taxonomy

This section introduces our taxonomy for research in the field of failure detection. We work in a tree-like and iteratively refining structure, as shown in Figure 1. The works are split along three different dimensions: (1) the prediction target, (2) the time horizon, and (3) the applied modeling type, i.e., the underlying base technique that was utilized. We will, in the following, briefly explain the distinction criteria for each dimension.

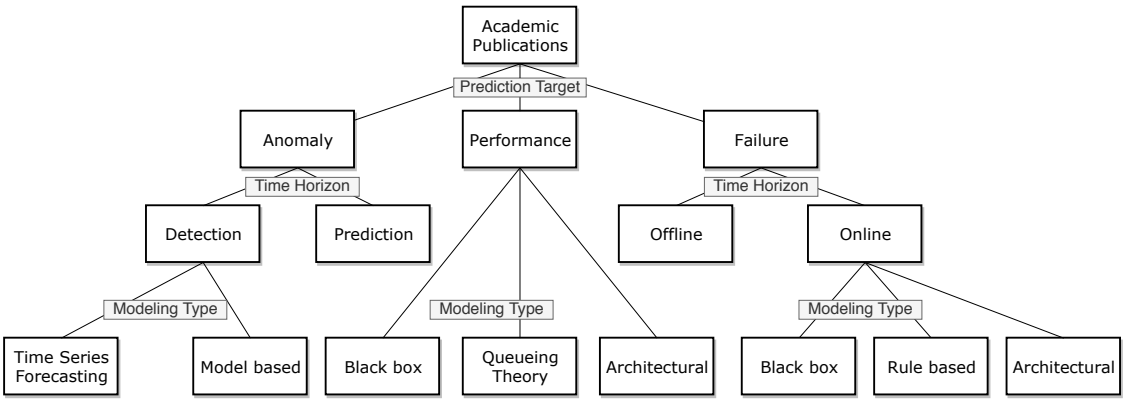

Figure 1. Graphical overview of the proposed taxonomy.

4.1. Prediction Target

Our first dimension concerns the prediction targets. As there exist a variety of different approaches for prediction of performance events, different notions for the prediction of performance properties exist. One key difference from related work is that we focus on SLO failure prediction, as opposed to general performance prediction and/or optimization, the prediction of performance anomalies (cf. Section 2) as well as the prediction failures caused by hardware faults. However, the definition of *failure* still varies between different communities and can be a software fault, a performance anomaly, a malfunction of the system, or a combination.

Hence, the first dimension aims at distinguishing the different notions of failure present in the literature. The first group is the work of anomaly prediction. Here, the focus is on detecting and/or predicting anomalous behavior of the system, typically described by a combination of time series. An anomaly can be a sudden load spike, an increase in user requests, resource consumption, or error rates.

The second group aims at predicting failures by predicting the general performance properties of a system. The rationale behind this interpretation is that by predicting the general system performance, all performance events, including erroneous or failure states, can be predicted and therefore used

for failure prediction. Therefore, these works are usually limited to predicting performance failures; however, they are also able to predict normal system states, with or without the presence of failures.

Third, the last group focuses on the definition of failure as defined by Pitakrat et al. [87] and Avizienis et al. [94]: A *service* failure is defined as an event that occurs when the service deviates from the correct behavior. This includes an increase in response time, a service outage, or an incorrect return result. For example, a failure might follow the following definition: *A transaction type is in a failure state if and only if (i), the 95th percentile of the server-side response times of all requests in each 10-s time window exceeds 1 s or (ii), the ratio of successful requests over all requests in the same window falls below 99.99%.*

The goal of these works is to cover all non-functional properties of a software system as opposed to just performance properties, as the works in the second group. Therefore, in the following, when we refer to a *failure*, we imply the definition stated above. Additionally, our definition of failure also is compatible with definitions from other surveys [2], as we only classify behavior as a failure if the issue becomes apparent to the end-user.

4.2. Time Horizon

The second dimension of interest is the time horizon of the model learning and model predictions. For most applications, it is important that the failure predictions can be delivered in a timely fashion, desirably even ahead of time [2]. Therefore, we divide all techniques of failure prediction into techniques that can be applied in an online context, and other techniques that serve as an offline analysis tool, delivering theoretical results or hypothetical scenarios. Note that, even though the model prediction is classified as online, the actual model learning could still happen offline.

As the focus of anomaly prediction is implicitly the implementation in an online context, the notion of time horizon is interpreted differently. For anomaly prediction, we distinguish the works between works that focus on the detection of anomalies, usually within a certain time window or after a pre-defined acceptable delay, and work that are able to predict anomalies, i.e., detect anomalies before the respective anomaly manifests in the system. The obvious advantage of prediction is the increased time that one is able to analyze or react to the detected anomaly, while detection algorithms are usually more robust and stable and can implicitly deliver some preliminary analysis.

4.3. Modeling Type

Lastly, we partition the works based on the underlying modeling type, i.e., the base techniques that are applied or utilized for performing the prediction. The underlying modeling type implies several characteristics ranging from the required type and amount of information, the time-to-result, and the accuracy deliverable by the approach to the types of failures that are predictable by the approach as well as the amount of explainability (i.e., why the model results in the given prediction). Naturally, not all techniques are applicable to all prediction targets.

For anomaly detection, we distinguish between rule-based approaches and approaches based on time series forecasting. Performance prediction approaches can be divided into approaches based on machine learning, queueing theory, or architectural software modeling. Online failure prediction can be done using rule-based approaches, machine learning, or architectural approaches.

We will discuss each of these groups, as well as the representative works from these groups in the following Section 5.

5. Survey Results

This section presents an overview of the related work and groups all works into the respective groups, according to the taxonomy introduced in Section 4. First, the prediction target as discussed in Section 4.1 defines at what types of prediction the work is aiming for. Along this dimension, we split all related works into three groups. We discuss the area of event and anomaly prediction in Section 5.1, the area of general performance prediction in Section 5.2, and the area focusing on the prediction

of actual failures in Section 5.3. Finally, we summarize all results in Section 5.4 by aggregating our classification into one table.

The first group, event and anomaly prediction is outlined in Section 5.1. These works focus on the prediction and detection of special events and/or anomalies. Detected or predicted events are usually not analyzed about whether or not they have a critical or negative impact on the application performance. For example, an anomaly could be an unusual spike in the number of concurrent users of a system. However, it is unclear whether this load spike leads to a performance problem for our application or not.

Second, we describe works generally targeted at performance prediction in Section 5.2. These works focus on the performance prediction of software systems, usually targeted towards what-if analyses and/or optimization of the respective software System under study (SUS). Therefore, they are usually generally applicable, but also require a lot of information to be provided and are seldom tailored to one specific use case.

Third, Section 5.3 deals with the predictions of failures. These works focus on the prediction of failures, according to the definition we discussed in Section 4.1. Note that we restrict ourselves to software failures in this literature review as there exists a lot of work focusing on detecting hardware failures. See the works of Pitakrat et al. [95], Murray et al. [96], Eckart et al. [97], Zhu et al. [98], Ganguly et al. [99], Wang et al. [100], or Züfle et al. [101] for predicting the failure of hard disks based on the S.M.A.R.T. data set. In addition, Chalermarrewong et al. [102] derive a technique for predicting component hardware failures using Auto-Regressive Moving Average (ARMA) models and combining the individual failures using a fault tree, a technique which offers similarities to many of the approaches focusing on software.

We conclude this section with a summarizing categorization of all listed works into our taxonomy table in Section 5.4.

5.1. Event and Anomaly Prediction

This section outlines works related to event and anomaly prediction. The focus of these works is usually not on classifying the severity or the degree of any anomaly, but rather to predict or detect if a measured behavior or a measured metric entity deviates from expected or normal behavior. However, the notion of event or anomaly might also include failures, as we define it in Section 4.1. In the following, we divide the works in this area along the dimension of the corresponding time horizon, as described in Section 4.2. Section 5.1.1 focuses on works aiming at detecting anomalies, i.e., classifying events while or after they happen based on the monitoring streams. In contrast, Section 5.1.2 describes works targeting the prediction of anomalies, i.e., alerting or detecting a performance event before it actually happens on the system.

5.1.1. Anomaly Detection

In addition to the works we enumerate here, Chandola et al. [3] present a comprehensive survey of anomaly detection approaches. However, according to the derived taxonomy for our work, we divide the area into the following two subgroups. Both model the expected behavior of the system and then classify an anomaly based on the deviation of the predicted expected behavior. However, works described in the first group anticipate the expected behavior by applying time series forecasting, while the second group focuses on more explicit models.

Detection Based on Time Series Forecasting

The works of Bielefeld [81], Frotscher [82], and Oehler et al. [14] utilize time series forecasting to detect anomalies in large scale software systems. Based on historical data, a forecast for the anticipated behavior is made. Then, the observed behavior can be compared to the predicted one in order to detect anomalies. Usually, the predicted metrics are compared to the measured values, and an anomaly is

alerted based on a pre-defined threshold. Another work based on time series modeling and prediction is the patent by Iyer and Zhao [92].

As this is a promising direction for anomaly detection (as well as for other domains), forecasting is still an active area of research. Hence, there exist different methods for forecasting that are of use for the aforementioned techniques. Wold [103], Hyndman et al. [104], Goodwin et al. [105], Herbst et al. [106], De Livera et al. [107], Züfle et al. [108], Faloutsos et al. [109], and Bauer et al. [110] aim at creating or improving general forecasting techniques. Note that those works do not specifically consider anomaly detection or performance prediction but propose generally applicable techniques for forecasting.

Detection Based on Normal Behavior Modeling

Other works define normal behavior based on models and detect anomalies based on deviation from the defined behavior. Chan et al. [89] define the model with rules based on machine learning, Song et al. [15] rely on manual models. Zhang et al. [16] employ unsupervised clustering on black-box tasks to induce normal resource usage behavior patterns from historical data. Monni et al. [17,18] develop a technique for energy-based anomaly detection using Restricted Boltzmann Machines (RBMs) [18] and acknowledge how their technique can be used to detect collective anomalies and failures in software systems. Chan and Mahoney [19] automatically construct models based on the observed time series, but without explicitly applying forecasting techniques. Rathfelder et al. [83] apply workload-aware performance predictions in order to improve to improve anomaly and failure detection. This targets the problem of fixed thresholds that apply for the approaches of this and the previous sections, as observed deviations from the anticipated behavior have to be analyzed subject to a specific threshold. Similarly, Mayle et al. [93] dynamically adapt the baseline and thresholds based on the historical properties of the system.

5.1.2. Anomaly Prediction

Tan et al. [20] propose an anomaly prediction mechanism, called ALERT, designed for predictive anomaly detection in large scale systems. They achieve predictive power by applying a three-state prediction approach. They divide into normal, anomaly, and alert states, with the alert state being the state directly before an anomaly occurs and, therefore, the predictive state of interest. They furthermore improve the model accuracy by clustering the different execution contexts into different groups and deriving a mapping between execution context and predictive model.

Schörgenhummer et al. [21] propose to apply online anomaly prediction based on multiple monitoring streams. They achieve this by training machine learning models on monitoring data provided by an industrial partner. The data are then split into vectors using features consisting of 34 data metrics and 11 aggregation functions. After that, each feature vector is assigned to be either normal or faulty, based on the available log data. The obtained data are then fed into a classification algorithm in order to predict a given monitoring stream as normal or abnormal.

5.2. Performance Prediction

Many approaches to online performance and resource management in dynamic environments have been developed in the literature. Approaches are typically based on control theory feedback loops, machine learning techniques, or stochastic performance models, such as Layered Queueing Networks (LQNs) or Stochastic Petri Networks (SPNs). Other approaches (listed in Section 5.2.3), develop their own modeling languages to describe the architecture of the software system, together with its performance properties. Performance models are typically used in the context of utility-based optimization techniques. They are embedded within optimization frameworks aiming at optimizing multiple criteria such as different Quality of Service (QoS) metrics [111–113]. One such use-case is, for example, the auto-scaling of containers or Virtual Machines (VMs) of applications. For more details, we refer to the study by Lorido-Botran et al. [114].

In the following, we divide the works from this area according to the underlying modeling formalism, as discussed in Section 4.3. First, black box or machine learning models included in Section 5.2.1 usually rely on statistical analysis or machine learning. Second, stochastic modeling formalisms based on queueing theory are described in Section 5.2.2, which already include specific information but not yet model the application architecture explicitly. Finally, Section 5.2.3 discusses white box or architectural modeling formalisms, where the software and its performance properties are explicitly modeled. Amiri and Mohammad-Khanli [4] give a further overview of performance prediction approaches. Their focus is on resource provisioning in the cloud.

5.2.1. Black-Box and Machine Learning Models

Black box models apply statistical approaches or machine learning techniques to historical or online measurement data in order to build a model representation of the software system. This technique has a variety of different terms in different communities and is also known in the literature as software performance curves [22,23], performance prediction functions [24], performance predictions using machine learning [25], or using statistical techniques [26]. These approaches train statistical models on measurement data usually collected during a dedicated measurement phase. The resulting models can then be used to infer the performance of a software system under different scenarios, for example, different workloads or software configurations. Other approaches based on feedback loops and control theory (e.g., [27,28]) aim at optimizing different QoS objectives while ensuring system stability. Machine learning techniques capture the system behavior based on observations at run-time without the need for an a priori analytical model of the system [29,30].

Thereska et al. [26] build a statistical performance model to predict the performance of several Microsoft applications. The authors collect data from several hundred thousand real users using instrumented applications. The respective performance predictions are then based on a similarity search after irrelevant features were filtered out using Classification and Regression Trees (CART). Mantis, developed by Kwon et al. [25], predicts the performance of Android applications using a regression model. The result is used to determine whether or not a specific task can be efficiently offloaded. The authors train the regression model on the input parameters to the application, the current hardware utilization, and on values calculated during the program execution. Westermann et al. [24] compare the algorithms Multivariate Adaptive Regression Splines (MARS), CART, Genetic Programming (GP), and Kriging in the context of constructing statistical performance models for performance prediction. In their case studies, MARS outperformed the other evaluated approaches. Furthermore, the authors compare three different algorithms for selecting measurement points used to train the regression algorithms. This is useful to reduce the number of required performance measurements. Noorshams et al. [31] evaluate the accuracy of Linear Regression (LR), MARS, CART, M5 Trees, and Cubist Forests for the performance prediction of storage systems. Furthermore, the authors propose to optimize the parameterization of the individual algorithms and propose their own algorithm, called Stepwise Sampling Search (S3) [115]. In their case study, MARS and Cubist outperform CART, M5 Trees, and LR after applying the proposed parameter optimization. Faber et al. [22] derive so-called software performance curves using GP techniques. They introduce parameter optimization approaches and a technique to prevent overfitting for GP. Additionally, they also propose a technique for parameter optimization. In their evaluation, the introduced optimized GP approach outperforms an unoptimized MARS model. Finally, Chow et al. [32], create a performance model by working with a large set of component hypotheses that get rejected if the empirical data do not support the respective hypotheses. They evaluate their model using an evaluation set of 1.3 million requests provided Facebook in order to predict and improve the end-to-end latency.

In summary, the approaches based on statistical or machine learning models can predict the impact of changes in workload intensity and workload parameterization well but are unreliable when predicting the impact of changes to the system or its deployment. Therefore, their application for

failure prediction is limited. On the other hand, the used models are relatively easy to obtain and are generically applicable.

5.2.2. Models Based on Queueing Theory

Works in this area mainly use predictive performance models to capture and predict system behavior. The platform is normally abstracted as a black-box, that is, the software architecture and configuration are not modeled explicitly (e.g., [33–38]). Such models include Queueing Networks (QNs) (e.g., [39,40]), LQNs (e.g., [41]), Queueing Petri Netss (QPNs) (e.g., [42]), stochastic process algebras [43], and statistical regression models (e.g., [44]). Models are typically solved analytically, e.g., based on mean-value analysis [34], or by simulation [38]. Both approaches have their respective benefits and downsides. Analytic solutions are usually much faster to compute; however, they might not be able to provide the same detail as a simulation or are only applicable to certain scenarios or system types. Therefore, a variety of approaches have been developed to enable dynamically switching between different methods (see, e.g., Walter et al. [116,117] for computer systems or Rygielski et al. [118,119] for networks) based on the current demand, or alternatively altering the prediction model itself [120,121]. These approaches accomplish this by working on a higher abstraction layer, e.g., the white-box descriptions of Section 5.2.3. For failure prediction, explicitly considering dynamic changes and being able to predict their effect at run-time is vital for ensuring predictable performance and ensuring to meet SLOs. Therefore, the application of the proposed approaches in the context of failure prediction is also possible; however, the required computation time of many of the simulation-based approaches make the application in an online context questionable. However, as most of the works lack a clear performance analysis for large-scale systems, it is hard to judge whether or not the application in an online context would be feasible. Conceptually, it could be possible for some approaches to apply the proposed approaches in online environments. For this reason, the performance prediction target is not divided with respect to the dimension of the time horizon.

5.2.3. Architectural White-Box Models

In addition to the techniques described in the previous sections, a number of meta-models for building architecture-level performance models, i.e., white-box models of software systems, have been proposed. Such meta-models provide modeling concepts, i.e., the ability to explicitly describe, to capture the performance-relevant behavior of a system as well as its architecture, together with some aspects of its execution environment, i.e., the hardware the respective system is deployed on [6]. The most prominent meta-models are the Unified Modeling Language (UML) Schedulability, Performance, & Time (SPT) and UML Modeling and Analysis of Real-time and Embedded systems (MARTE) profiles [90]. Further proposed meta-models include ACME [45–47], ROBOCOP [48], Software Performance Engineering Meta-Model (SPE-MM) [49], Core Scenario Model (CSM) [50], KLAPER [51], SLAstic [84], Palladio Component Model (PCM) [52], and the Descartes Modeling Language (DML) [53]. A more detailed overview and comparison of some of the given techniques can be found in the survey by Koziolek [6].

A major drawback of all white-box approaches is the missing adoption in the industry, due to their inherent complexity [122]. Additionally, all approaches focusing solely on performance prediction is that they are usually able to accurately predict performance problems (i.e., time-outs or latency-spikes), but are not able to predict content failures. Content failures include, for example, malicious or faulty user behavior that leads to erroneous responses or software failures. As most performance modeling formalisms do not include these properties into their model abstractions, they are not able to anticipate such types of failures.

5.3. Failure Prediction

This section describes all work falling in the area of failure prediction, as we defined it in Section 4.1. Since our definition is not restrictive, the area comprises a wide variety of approaches, which makes it necessary to divide the work into further subgroups.

Therefore, we split the works by analyzing the time horizon of the model learning and the model predictions, as already discussed in Section 4.2. Therefore, we will, in the following, distinguish between works intended for offline application in Section 5.3.1, and approaches explicitly developed for an online context in Section 5.3.2. Approaches designed for online applications must not only provide a reasonable time-to-result but also need to ensure a low Central Processing Unit (CPU) and memory footprint.

5.3.1. Offline Prediction

If the objective focus is not to predict software failures at run-time but rather to generally analyze the reliability or the safety of a system, one approach is using reliability models (as proposed by Goševa-Popstojanova and Trivedi [54]) or safety models (see Grunske and Han [55]) are of relevance. These works are in principle very similar to the performance models introduced in the previous Section 5.2, but instead of specifically focusing the performance of a system, these models analyze quality of a software [56–58], or the amount and severity of probable failures occurring in a software system for a specific scenario.

Notable approaches include the works of Cheung [59], Cortallessa and Grassi [60], Yilmaz and Porter [61], Brosch [85], and Uhle [86]. Cheung [59] employs a Markov model to build a reliability model of the whole software system based on the reliability of the individual components. Similarly, Cortallessa and Grassi [60] consider the error propagation probabilities in addition to the single failure probabilities of each component based on the system architecture. Yilmaz and Porter [61] classify measured executions into successful and failed executions in order to apply the resulting models to systems with an unknown failure status. They furthermore apply their technique to online environments (see Section 5.3.2.3). Brosch [85] extends the performance modeling formalism PCM introduced in Section 5.2.3 with reliability attributes in order to receive an estimation of the reliability of the system. Finally, Uhle [86] models the dependability of micro-service architecture applications using dependency graphs.

All of the included approaches include the architecture of the application in order to assess the reliability of the approaches. However, none of them is intended for the online prediction of failures.

5.3.2. Online Prediction

This section focuses on methods for online failure prediction. A comprehensive survey about different failure prediction methods in an online context is already given by Salfner et al. [2]. In the following, we will assume all works to either specifically consider online environments or at least are easily applicable in that context.

Note that we do not explicitly consider the area of High Performance Computing (HPC) and distributed computing, mainly focusing on batch processing. Most works in the field of distributed computing aim at predicting parallel-running, long-term task durations, optimizing scheduling strategies, wait times, or overall throughput. In contrast, cloud computing usually aims at minimizing latency for user-facing web applications. Here, mostly transaction-oriented, latency-sensitive, and interactive applications are analyzed.

Witt et al. [5] survey PPM approaches in the area of distributed computing and give an overview of the state-of-the-art in that field. Note that the referenced survey focuses on general performance prediction, not solely on failure prediction in the context of batch-processing. Works focusing on distributed computing or HPC include Islam and Dakshnamoorthy [123], Zheng et al. [124], Yu et al. [125], or Liang et al. [126]. Islam and Dakshnamoorthy [123] use Long Short-Term Memory

(LSTM) networks to detect and predict task failures before they occur using traces from Google cloud. Zheng et al. [124], Yu et al. [125], and Liang et al. [126] all extract rules from log data of the Blue Gene supercomputer. These rules are then used to predict failures at run-time. Although these works focus on HPC, there is another branch of works applying the same principle on web servers (see Section 5.3.2.2).

Similarly to the previous sections, this section will further analyze all works concerned with the online prediction of failures for user-facing web applications by splitting them into different groups. For the remainder of this section, we assume that all works concentrate on user-facing web applications, as described at the beginning of this section. In the following, we analyze the type of model used to conduct the prediction of performance degradation and/or failure, as explained in Section 4.3.

We, therefore, divide this section into two sub-sections, grouping the modeling types of the area. First, we list monolithic black-box models in Section 5.3.2.1, i.e., models that treat the application as black-box and do not contain or require any architectural knowledge about the application. These models are mostly based on machine learning. Afterwards, we present works implicitly incorporating architectural or white-box information about the SUS using rules in Section 5.3.2.2, while works in Section 5.3.2.3 use explicit architectural information.

Black-Box and Machine Learning Models

Alonso et al. [62] apply and evaluate different machine learning models with each other in order to find the best suitable for their task of predicting anomalies and failures based on software aging caused by resource exhaustion. They furthermore employ Lasso regularization in order to filter features. Their evaluation is based on the TPC-W benchmark. Lou et al. [63] predict the failures of cloud services with Relevance Vector Machines (RVMs) using different flavors of Quantum-inspired Binary Gravitational Search Algorithms (QBGSAs). Li et al. [64] postulate to incorporate information from the network, the hardware, and the software in order to detect not only timing but also content failures as both contribute to failure problems. This approach, therefore, also follows the failure definition of Section 4.1. Sharma et al. [65] apply a multi-layered online learning mechanism in order to distinguish cloud-related anomalies from application faults and furthermore try to include automatic remediation. They evaluate their tool on Hadoop, Olio, and Rubis, using enterprise traces on an IaaS cloud testbed. Daraghmeh et al. [66] use local regression models and Box–Cox transformation to forecast and predict the future state of each host. This information is then utilized to optimized VM placement and consolidation within a cloud platform in order to optimize resource usage. Although the target of this work is different than our approach, the applied algorithms might still be useful for our approach. Grohmann et al. [67] use random forest machine learning techniques to build a model for resource saturation of micro-service applications running the cloud. They propose to use one holistic model for all types of applications in order to infer resource saturation of different applications, without measuring any QoS metrics at the application level. The works of Cavallo et al. [68] and Amin et al. [69–71] predict QoS violations of web servers based on time series analysis methods like Auto-Regressive Integrated Moving Average (ARIMA) or Generalized Auto-Regressive Conditional Heteroskedasticity (GARCH) models. Van Beek et al. [72] focus on predicting CPU contentions with the use of different regression models. They evaluate their approach on a workload trace from a cloud data center.

Rule-Based Models

NETradamus is a framework for forecasting failures based on event messages proposed by Clemm and Hartwig [73]. The tool mines critical event patterns from logs based on past failures in order to create rules for the detection of failures in an online context. The work of Gu et al. [74], Pitakrat et al. [75], and—as already introduced at the beginning of this section—the works of Zheng et al. [124], Yu et al. [125], and Liang et al. [126], are based on a similar idea, i.e., creating rules based on log events of past failures in order to predict new upcoming failures of the same type.

The presented works do not explicitly consider explainability. However, as rule-based approaches are generally easier to be interpreted by humans and event logs can be used for root-cause analysis, we consider these approaches to be white-box models as they implicitly deliver a certain degree of explainability.

Architectural Models

Pitakrat et al. [76,87] develop a technique for predicting failures in an online environment. Their technique is based on two components. First, failure probabilities of individual components are predicted based on defined thresholds for key metrics and ARIMA based metric forecasting. Second, these failure probabilities are inserted into a Failure Propagation Model (FPM) using a Bayesian Network (BN) in order to incorporate the probabilities of cascading failures. Mohamed [88] uses the so-called error spread signature in order to derive a Connection Dependency Graph (CDG) for propagating software function failures (excluding QoS metrics), an approach that works similarly, but does not consider probabilistic propagation. Pertet and Narasimhan [91] published a technical report, where they concluded that a lot of software failures can be traced back to fault chains, i.e., cascading failures, which supports the above modeling strategies. To that end, they also propose their own approach [77] that uses the dependency graph of the application nodes in order to deal with cascading failures. Capelastegui et al. [78] propose to combine monitoring on VM-level as well as on host-level together with different data sources like monitoring data, event logs, and failure data to derive online failure predictions. Similarly, Ozcelik and Yilmaz [79] propose to combine measurement from hardware and from inside the software to so-called *hybrid spectra*, in order to overcome the downsides of black-box failure models while keeping the monitoring overhead acceptable. This is the improved version of their offline technique already introduced in Section 5.3.1. Lastly, Mariani et al. [80] aim at predicting failures in distributed multi-tier environments by employing a two-stage approach, consisting of anomaly detection in the first stage, and a signature-based classification technique in the second stage.

5.4. Summarizing Taxonomy Table

We conclude this section by presenting a final taxonomy comprising of all categories discussed in Sections 5.1–5.3. Table 1 presents a comprehensible version of the aggregation of all works into the taxonomy presented in Section 4. In addition, we assess each group of approaches with regard to the three properties Generality (**G**), Adaptability (**A**), and Explainability (**E**) in Table 1. Categories marked with (+) apply the specific property better than categories marked with (−).

Generality estimates how generic the specific group of approaches is, and how well it can be applied to different software systems. Generally, we find that black-box approaches are usually advantageous here, as they require very little knowledge about the systems, and fewer assumptions need to be made. In contrast, architectural and queueing theoretical approaches only apply to system types that can be modeled using the particular modeling formalism, which in turn limits their generality. Similarly, the rule-based systems from the second paragraph of Section 5.3.2 mostly rely on system logs and therefore make assumptions about the structure of those.

Adaptability refers to the ability to react to changes in the system, i.e., to deliver accurate predictions, even if the original modeled system changed. This is a strength of all approaches dealing with anomaly prediction, as their goal is basically to detect such deviations from the system model. Additionally, as architectural approaches encode more information about the system, they are able to model more states of the systems. On the contrary, black-box approaches encode less information and are, therefore, seldom able to transfer the knowledge to unknown system states. The same applies to the rule-based approaches, as rules are usually hard to abstract.

Finally, we discuss the concept of explainability. Explainability, as we discuss in more detail in Section 6, refers to the ability to report on the reason for a certain SLO failure of the system, sometimes also referred to as *root-cause*. Hence, approaches designed for anomaly prediction can usually just

detect a non-normal state, without the ability to pinpoint its cause. The same applies to black-box approaches, as long as they are not explicitly trained with the corresponding root-causes. Generally, the more expressive architectural model types usually have the best chances of finding the reason for the failure as they encode most information in the system. However, we would like to note that the degree of explainability is still quite limited, in our opinion, which is why we discuss this as an open issue in Section 6.

Table 1. Comprehensive depiction the presented classification along with our assessment of Generality (G), Adaptability (A), and Explainability (E).

Prediction Target	Time Horizon	Modeling Type	G	A	E	References
Anomaly Prediction	Detection	Time Series	+	+	-	[14,81,82,92]
		Model based	+	+	-	[15–19,83,89,93]
	Prediction		+	+	-	[20,21]
Performance Prediction		Black box	+	-	-	[22–32]
		Queueing Theory	-	+	-	[33–44]
		Architectural	-	+	+	[45–53,84,90]
Failure Prediction	Offline		-	-	+	[54–61,85,86]
	Online	Black box	+	-	-	[62–72]
		Rule based	-	-	+	[73–75]
		Architectural	-	+	+	[76–80,87,88,91]

6. Open Research Challenges

In this section, we list some remaining challenges and open research issues that have not or rarely been addressed by the respective works in the field.

6.1. Explainability

One important issue when predicting failures is the explainability of the approaches. We distinguish between two types: (1) model explainability; and (2) failure explainability. Model explainability refers to the predictions of the model itself. Here, the focus is on *why* the specific approach resulted in the respective prediction, i.e., what specific inputs led to which intermediate and final results. Model explainability is often desired or even required, when a lot of trust in the applied approach is required, i.e., in production or high-availability systems. Failure explainability targets more towards the actual software system in production. Usually, the root cause of the failure is desired in order to be able to avoid failure manifestation, quickly resolve the failure, or mitigate the noticeable effects of the failure for the end-user. The specific degree of explainability required of each failure prediction varies strongly between each use case. We argue that both types of explainability are strongly dependent on the chosen modeling types and their restrictions. Furthermore, it is crucial to address both types of explainability when designing a production-ready system. Hence, we encourage the research community to increase the effort towards these directions.

6.2. Resource Consumption

Although most approaches included in this work are intended for implementation in an online environment, many works do not explicitly consider or analyze the performance of the prediction approach itself. We argue that, in order to be applicable in an online environment of a production system, the resource consumption, i.e., the CPU and memory footprint of any approach must be analyzed in depth. However, as most of the works lack a clear performance analysis for large-scale systems, it is hard to judge whether or not the application in an online context would be feasible. A possible future work of our study could consist of a representative benchmark that evaluates the

performance of the different approaches in terms of accuracy and time-to-result, as well as resource and energy consumption.

6.3. Hybrid or Overarching Approaches

This work presents a variety of different modeling strategies and approaches targeted at achieved the same or a similar goal. As all these approaches show different advantages and disadvantages in certain scenarios, it seems natural to combine different approaches in order to create an overarching approach that is able to combine different strengths of various approaches. For example, many failure prediction methods included in Section 5.3 rely on anomaly detection. However, only seldom advanced techniques, like the ones presented in Section 5.1, are utilized. Instead, standard off-the-shelf anomaly detection is deployed. While this is a reasonable restriction in a research paper with a specific focus, it is certainly possible to improve the proposed approach by improving its anomaly detection component. Additionally, it seems useful to combine the advantages of machine learning techniques (see the first paragraph of Section 5.3.2) with methods using architectural knowledge (see the third paragrah of Section 5.3.2) for failure prediction by creating a hybrid approach. We, therefore, want to motivate the community to increase work towards compound approaches in the future.

7. Limitations and Threats to Validity

Although we conducted this survey to the best of our knowledge and belief, there are still some remaining threats to validity to be discussed.

First, the applied search method of systematic mapping might not include all relevant papers for the considered scope. As already discussed in the methodology section, a systematic literature review did not seem fruitful for our application, which is why we think the proposed methodology is the best way of collecting a reasonable amount of papers. However, we can not rule out that some relevant areas of research have been missed during the study. Nevertheless, due to the nature of the taxonomy creation, this can not challenge the validity of the proposed taxonomy. Instead, additional references and categorizations need to be added in future versions in order to complete the taxonomy view.

Second, we use the analyzed works in this study to draw conclusions about the strengths and weaknesses of the discussed approaches, as well as to identify directions for future research in the area. Due to the summarizing, abstracting, and communalizing nature of our taxonomy, we need to formulate general and generic statements that apply to all or almost all works in the specific group. Hence, although we aim to avoid such cases, there can be occasional instances where a conclusion drawn for the whole group of approaches might not be true for specific approaches of that sub-field.

Third, this study represents our critical view of the current state-of-the-art. It is therefore unavoidable that future research addresses some of the weaknesses as well as challenges and open issues we pointed out throughout this study. Hence, not all claims and statements made in this work might still apply if the research in the field progresses. However, as this lies in the nature of all sciences, this can not be seen as a threat to this research overview. Instead, this work can be used for later analysis and comparisons in what areas of research developed and improved, and which areas still need more research attention in the future.

Finally, this study does not provide a quantitative comparison in terms of prediction accuracy or required computation time. Although such a benchmark would provide a huge benefit for the research community, such a comparative evaluation is out of scope for this work. First, many of the discussed works do not disclose open-source implementations. Second, even if implementations are available, running this wide variety of approaches, each with their own interface implementations, poses a huge technical challenge. Additionally, some conceptual questions need to be answered in order to conduct a comparative study: What are realistic test scenarios for this investigation? What information can/will be provided to the individual approaches? What is the degree of detail that is expected for a specific prediction? How can the need for more information be penalized, or a more detailed prediction (e.g., smaller time window, faster prediction, mitigation or root-cause information

included) be awarded? However, answering these questions and providing quantitative comparison could be an interesting target for future work.

8. Conclusions

This work presents a taxonomy of SLO failure prediction techniques by dividing research works along three different dimensions: (1) the prediction target (e.g., anomaly detection, performance prediction, or failure prediction); (2) the time horizon (e.g., detection or prediction, online or offline application); and (3) the applied modeling type, i.e., the underlying base technique that was utilized (e.g., time series forecasting, machine learning, or queueing theory). Furthermore, we present an overview of the current state-of-the-art and categorize existing works into the different groups of the taxonomy.

Finally, we address open issues and remaining research challenges in order to guide future research. This work can be used by both researchers and practitioners to classify and compare different approaches for SLO failure prediction.

Author Contributions: Conceptualization, N.H. and S.K.; Funding acquisition, N.P.; Investigation, J.G. and Y.A.; Project administration, N.P. and S.K.; Supervision, A.C.; Writing—original draft, J.G.; Writing—review and editing, N.H., A.C., and Y.A. All authors have read and agreed to the published version of the manuscript.

Acknowledgments: The authors would like to thank the editors and the reviewers for their valuable feedback and helpful suggestions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kounev, S.; Lewis, P.; Bellman, K.; Bencomo, N.; Camara, J.; Diaconescu, A.; Esterle, L.; Geihs, K.; Giese, H.; Götz, S.; Inverardi, P.; Kephart, J.; Zisman, A. The Notion of Self-Aware Computing. In *Self-Aware Computing Systems*; Kounev, S.; Kephart, J.O.; Milenkoski, A.; Zhu, X., Eds.; Springer Verlag: Berlin Heidelberg, Germany, 2017.
2. Salfner, F.; Lenk, M.; Malek, M. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)* **2010**, *42*, 10.
3. Chandola, V.; Banerjee, A.; Kumar, V. Anomaly detection: A survey. *ACM computing surveys (CSUR)* **2009**, *41*, 15.
4. Amiri, M.; Mohammad-Khanli, L. Survey on prediction models of applications for resources provisioning in cloud. *Journal of Network and Computer Applications* **2017**, *82*, 93 – 113. doi:https://doi.org/10.1016/j.jnca.2017.01.016.
5. Witt, C.; Bux, M.; Gusew, W.; Leser, U. Predictive performance modeling for distributed batch processing using black box monitoring and machine learning. *Information Systems* **2019**, *82*, 33 – 52. doi:https://doi.org/10.1016/j.is.2019.01.006.
6. Koziolek, H. Performance evaluation of component-based software systems: A survey. *Performance Evaluation* **2010**, *67*, 634–658.
7. Márquez-Chamorro, A.E.; Resinas, M.; Ruiz-Cortés, A. Predictive Monitoring of Business Processes: A Survey. *IEEE Transactions on Services Computing* **2018**, *11*, 962–977. doi:10.1109/TSC.2017.2772256.
8. Weingärtner, R.; Bräscher, G.B.; Westphall, C.B. Cloud resource management: A survey on forecasting and profiling models. *Journal of Network and Computer Applications* **2015**, *47*, 99 – 106. doi:https://doi.org/10.1016/j.jnca.2014.09.018.
9. Kephart, J.O.; Chess, D.M. The vision of autonomic computing. *Computer* **2003**, *36*, 41–50. doi:10.1109/MC.2003.1160055.
10. Webster, J.; Watson, R.T. Analyzing the past to prepare for the future: Writing a literature review. *MIS quarterly* **2002**, pp. xiii–xxiii.
11. Kitchenham, B.; Charters, S. Guidelines for performing systematic literature reviews in software engineering. Technical report, School of Computer Science and Mathematics, Keele University, 2007.
12. Petersen, K.; Feldt, R.; Mujtaba, S.; Mattsson, M. Systematic mapping studies in software engineering. Ease, 2008, Vol. 8, pp. 68–77.

13. Booth, A. Unpacking your literature search toolbox: on search styles and tactics. *Health information and libraries journal* **2008**, *25*, 313.
14. Oehler, M.; Wert, A.; Heger, C. Online Anomaly Detection Based on Monitoring Traces. Proceedings of the 2016 Symposium on Software Performance (SSP '16), 2016.
15. Song, X.; Wu, M.; Jermaine, C.; Ranka, S.; others. Conditional anomaly detection. *IEEE Transactions on Knowledge and Data Engineering* **2007**, *19*, 631–645.
16. Zhang, X.; Meng, F.; Chen, P.; Xu, J. TaskInsight: A fine-grained performance anomaly detection and problem locating system. IEEE 9th International Conference on Cloud Computing (CLOUD 2016). IEEE, 2016, pp. 917–920.
17. Monni, C.; Pezzè, M. Energy-Based Anomaly Detection: A New Perspective for Predicting Software Failures. Proc. International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track. IEEE, 2019.
18. Monni, C.; Pezzè, M.; Prisco, G. An RBM Anomaly Detector for the Cloud. 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST), 2019, pp. 148–159. doi:10.1109/ICST.2019.00024.
19. Chan, P.K.; Mahoney, M.V. Modeling Multiple Time Series for Anomaly Detection. Proceedings of the Fifth IEEE International Conference on Data Mining; IEEE Computer Society: USA, 2005; ICDM '05, p. 90–97. doi:10.1109/ICDM.2005.101.
20. Tan, Y.; Gu, X.; Wang, H. Adaptive System Anomaly Prediction for Large-scale Hosting Infrastructures. Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing; ACM: New York, NY, USA, 2010; PODC '10, pp. 173–182. doi:10.1145/1835698.1835741.
21. Schörgenhumer, A.; Kahlhofer, M.; Grünbacher, P.; Mössenböck, H. Can We Predict Performance Events with Time Series Data from Monitoring Multiple Systems? Companion of the 2019 ACM/SPEC International Conference on Performance Engineering; Association for Computing Machinery: New York, NY, USA, 2019; ICPE '19, p. 9–12. doi:10.1145/3302541.3313101.
22. Faber, M.; Happe, J. Systematic Adoption of Genetic Programming for Deriving Software Performance Curves. Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering; ACM: New York, NY, USA, 2012; ICPE '12, pp. 33–44.
23. Westermann, D.; Momm, C. Using Software Performance Curves for Dependable and Cost-Efficient Service Hosting. Proceedings of the 2nd International Workshop on the Quality of Service-Oriented Software Systems; Association for Computing Machinery: New York, NY, USA, 2010; QUASOSS '10. doi:10.1145/1858263.1858267.
24. Westermann, D.; Happe, J.; Krebs, R.; Farahbod, R. Automated inference of goal-oriented performance prediction functions. Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. IEEE, 2012, pp. 190–199.
25. Kwon, Y.; Lee, S.; Yi, H.; Kwon, D.; Yang, S.; Chun, B.G.; Huang, L.; Maniatis, P.; Naik, M.; Paek, Y. Mantis: Automatic performance prediction for smartphone applications. Proceedings of the 2013 USENIX Annual Technical Conference; USENIX Association: Berkeley, USA, 2013; pp. 297–308.
26. Thereska, E.; Doebel, B.; Zheng, A.X.; Nobel, P. Practical Performance Models for Complex, Popular Applications. *SIGMETRICS Performance Evaluation Review* **2010**, *38*, 1–12.
27. Abdelzaher, T.F.; Shin, K.G.; Bhatti, N. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems* **2002**, *13*, 80–96.
28. Almeida, J.; Almeida, V.; Ardagna, D.; Cunha, Í.; Francalanci, C.; Trubian, M. Joint admission control and resource allocation in virtualized servers. *Journal of Parallel and Distributed Computing* **2010**, *70*, 344–362.
29. Tesauro, G.; Jong, N.K.; Das, R.; Bennani, M.N. A hybrid reinforcement learning approach to autonomic resource allocation. Autonomic Computing, 2006. ICAC'06. IEEE International Conference on. IEEE, 2006, pp. 65–73.
30. Kephart, J.O.; Chan, H.; Das, R.; Levine, D.W.; Tesauro, G.; Rawson III, F.L.; Lefurgy, C. Coordinating Multiple Autonomic Managers to Achieve Specified Power-Performance Tradeoffs. ICAC, 2007, Vol. 7, pp. 24–33.
31. Noorshams, Q.; Bruhn, D.; Kounev, S.; Reussner, R. Predictive performance modeling of virtualized storage systems using optimized statistical regression techniques. Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering; ACM: New York, USA, 2013; ICPE '13, pp. 283–294.

32. Chow, M.; Meisner, D.; Flinn, J.; Peek, D.; Wenisch, T.F. The mystery machine: End-to-end performance analysis of large-scale internet services. 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), 2014, pp. 217–231.
33. Chen, Y.; Das, A.; Qin, W.; Sivasubramaniam, A.; Wang, Q.; Gautam, N. Managing server energy and operational costs in hosting centers. ACM SIGMETRICS Performance Evaluation Review. ACM, 2005, Vol. 33, pp. 303–314.
34. Zhang, Q.; Cherkasova, L.; Smirni, E. A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications. Fourth International Conference on Autonomic Computing (ICAC'07), 2007, pp. 27–37. doi:10.1109/ICAC.2007.1.
35. Urgaonkar, B.; Pacifici, G.; Shenoy, P.; Spreitzer, M.; Tantawi, A. An analytical model for multi-tier internet services and its applications. ACM SIGMETRICS Performance Evaluation Review. ACM, 2005, Vol. 33, pp. 291–302.
36. Bennani, M.; Menasce, D. Resource allocation for autonomic data centers using analytic performance models. Second International Conference on Autonomic Computing (ICAC 2005). IEEE, 2005, pp. 229–240.
37. Abrahao, B.; Almeida, V.; Almeida, J.; Zhang, A.; Beyer, D.; Safai, F. Self-adaptive SLA-driven capacity management for internet services. 10th IEEE/IFIP Network Operations and Management Symposium. IEEE, 2006, pp. 557–568.
38. Jung, G.; Joshi, K.R.; Hiltunen, M.A.; Schlichting, R.D.; Pu, C. Generating adaptation policies for multi-tier applications in consolidated server environments. International Conference on Autonomic Computing (ICAC). IEEE, 2008, pp. 23–32.
39. Menascé, D.A.; Gomaa, H. A Method for Design and Performance Modeling of Client/Server Systems. *IEEE Transactions on Software Engineering* **2000**, *26*, 1066–1085.
40. Santhi, K.; Saravanan, R. Performance analysis of cloud computing using series of queues with Erlang service. *International Journal of Internet Technology and Secured Transactions* **2019**, *9*, 147–162.
41. Li, J.; Chinneck, J.; Woodside, M.; Litoiu, M.; Iszlai, G. Performance model driven QoS guarantees and optimization in clouds. Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing. IEEE Computer Society, 2009, pp. 15–22.
42. Kounev, S. Performance modeling and evaluation of distributed component-based systems using queueing petri nets. *IEEE Transactions on Software Engineering* **2006**, *32*, 486–502.
43. Gilmore, S.; Haenel, V.; Kloul, L.; Maidl, M. Choreographing security and performance analysis for web services. In *Formal Techniques for Computer Systems and Business Processes*; Springer, 2005; pp. 200–214.
44. Eskenazi, E.; Fioukov, A.; Hammer, D. Performance prediction for component compositions. International Symposium on Component-Based Software Engineering. Springer, 2004, pp. 280–293.
45. Garlan, D.; Monroe, R.; Wile, D. Acme: An Architecture Description Interchange Language. Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research. IBM Press, 1997, CASCON '97, p. 7.
46. Spitznagel, B.; Garlan, D. Architecture-based performance analysis. Proceedings of the 1998 Conference on Software Engineering and Knowledge Engineering. San Francisco, CA, USA, 1998, pp. 146–151.
47. Garlan, D.; Monroe, R.; Wile, D. Acme: An Architecture Description Interchange Language. CASCON First Decade High Impact Papers; IBM Corp.: USA, 2010; CASCON '10, p. 159–173. doi:10.1145/1925805.1925814.
48. Bondarev, E.; Muskens, J.; de With, P.; Chaudron, M.; Lukkien, J. Predicting real-time properties of component assemblies: A scenario-simulation approach. Proceedings of the 30th Euromicro Conference. IEEE, 2004, pp. 40–47.
49. Smith, C.U.; Lladó, C.M.; Cortellessa, V.; Marco, A.D.; Williams, L.G. From UML models to software performance results: an SPE process based on XML interchange formats. Proceedings of the 5th international workshop on Software and performance. ACM, 2005, pp. 87–98.
50. Petriu, D.B.; Woodside, M. An intermediate metamodel with scenarios and resources for generating performance models from UML designs. *Software & Systems Modeling* **2007**, *6*, 163–184.
51. Grassi, V.; Mirandola, R.; Sabetta, A. Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. *Journal of Systems and Software* **2007**, *80*, 528–558.
52. Becker, S.; Koziolek, H.; Reussner, R. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software* **2009**, *82*, 3 – 22.

53. Kounev, S.; Huber, N.; Brosig, F.; Zhu, X. A Model-Based Approach to Designing Self-Aware IT Systems and Infrastructures. *IEEE Computer* **2016**, *49*, 53–61. doi:10.1109/MC.2016.198.
54. Goševa-Popstojanova, K.; Trivedi, K.S. Architecture-based approach to reliability assessment of software systems. *Performance Evaluation* **2001**, *45*, 179–204.
55. Grunske, L.; Han, J. A comparative study into architecture-based safety evaluation methodologies using AADL's error annex and failure propagation models. 11th IEEE High Assurance Systems Engineering Symposium. IEEE, 2008, pp. 283–292.
56. Babar, M.A.; Gorton, I. Comparison of scenario-based software architecture evaluation methods. 11th Asia-Pacific Software Engineering Conference. IEEE, 2004, pp. 600–607.
57. Babar, M.A.; Zhu, L.; Jeffery, R. A framework for classifying and comparing software architecture evaluation methods. 2004 Australian Software Engineering Conference. Proceedings. IEEE, 2004, pp. 309–318.
58. Grunske, L. Early quality prediction of component-based systems–a generic framework. *Journal of Systems and Software* **2007**, *80*, 678–686.
59. Cheung, R.C. A User-Oriented Software Reliability Model. *IEEE Transactions on Software Engineering* **1980**, *SE-6*, 118–125. doi:10.1109/TSE.1980.234477.
60. Cortellessa, V.; Grassi, V. A modeling approach to analyze the impact of error propagation on reliability of component-based systems. International Symposium on Component-Based Software Engineering. Springer, 2007, pp. 140–156.
61. Yilmaz, C.; Porter, A. Combining Hardware and Software Instrumentation to Classify Program Executions. Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering; Association for Computing Machinery: New York, NY, USA, 2010; FSE '10, p. 67–76. doi:10.1145/1882291.1882304.
62. Alonso, J.; Belanche, L.; Avresky, D.R. Predicting Software Anomalies Using Machine Learning Techniques. IEEE 10th International Symposium on Network Computing and Applications, 2011, pp. 163–170. doi:10.1109/NCA.2011.29.
63. Lou, J.; Jiang, Y.; Shen, Q.; Wang, R. Failure prediction by relevance vector regression with improved quantum-inspired gravitational search. *Journal of Network and Computer Applications* **2018**, *103*, 171 – 177. doi:https://doi.org/10.1016/j.jnca.2017.11.013.
64. Li, L.; Lu, M.; Gu, T. Extracting Interaction-Related Failure Indicators for Online Detection and Prediction of Content Failures. 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2018, pp. 278–285. doi:10.1109/ISSREW.2018.00019.
65. Sharma, B.; Jayachandran, P.; Verma, A.; Das, C.R. CloudPD: Problem determination and diagnosis in shared dynamic clouds. 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2013, pp. 1–12. doi:10.1109/DSN.2013.6575298.
66. Daraghmeh, M.; Agarwal, A.; Goel, N.; Kozlowskif, J. Local Regression Based Box-Cox Transformations for Resource Management in Cloud Networks. Sixth International Conference on Software Defined Systems (SDS), 2019, pp. 229–235. doi:10.1109/SDS.2019.8768643.
67. Grohmann, J.; Nicholson, P.K.; Iglesias, J.O.; Kounev, S.; Lugones, D. Monitorless: Predicting Performance Degradation in Cloud Applications with Machine Learning. Proceedings of the 20th ACM/IFIP Middleware Conference; ACM: New York, NY, USA, 2019; Middleware '19.
68. Cavallo, B.; Di Penta, M.; Canfora, G. An Empirical Comparison of Methods to Support QoS-Aware Service Selection. Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems; Association for Computing Machinery: New York, NY, USA, 2010; PESOS '10, p. 64–70. doi:10.1145/1808885.1808899.
69. Amin, A.; Colman, A.; Grunske, L. Using automated control charts for the runtime evaluation of qos attributes. IEEE 13th International Symposium on High-Assurance Systems Engineering. IEEE, 2011, pp. 299–306.
70. Amin, A.; Colman, A.; Grunske, L. An approach to forecasting QoS attributes of web services based on ARIMA and GARCH models. IEEE 19th International Conference on Web Services. IEEE, 2012, pp. 74–81.
71. Amin, A.; Grunske, L.; Colman, A. An automated approach to forecasting QoS attributes based on linear and non-linear time series modeling. Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. ACM, 2012, pp. 130–139.

72. Van Beek, V.; Oikonomou, G.; Iosup, A. A CPU contention predictor for business-critical workloads in cloud datacenters. IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W). IEEE, 2019, pp. 56–61.
73. Clemm, A.; Hartwig, M. NETradamus: A forecasting system for system event messages. IEEE Network Operations and Management Symposium (NOMS 2010). IEEE, 2010, pp. 623–630.
74. Gu, X.; Papadimitriou, S.; Philip, S.Y.; Chang, S.P. Online failure forecast for fault-tolerant data stream processing. IEEE 24th International Conference on Data Engineering. IEEE, 2008, pp. 1388–1390.
75. Pitakrat, T.; Grunert, J.; Kabierschke, O.; Keller, F.; van Hoorn, A. A Framework for System Event Classification and Prediction by Means of Machine Learning. Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools; ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering): Brussels, BEL, 2014; VALUETOOLS '14, p. 173–180. doi:10.4108/icst.valuetools.2014.258197.
76. Pitakrat, T.; Okanović, D.; van Hoorn, A.; Grunske, L. Hora: Architecture-aware online failure prediction. *Journal of Systems and Software* **2018**, *137*, 669 – 685. doi:https://doi.org/10.1016/j.jss.2017.02.041.
77. Pertet, S.; Narasimhan, P. Handling cascading failures: the case for topology-aware fault-tolerance. Proceedings of the IEEE First Workshop on Hot Topics in System Dependability, 2005.
78. Capelastegui, P.; Navas, A.; Huertas, F.; Garcia-Carmona, R.; Dueñas, J.C. An Online Failure Prediction System for Private IaaS Platforms. Proceedings of the 2nd International Workshop on Dependability Issues in Cloud Computing; Association for Computing Machinery: New York, NY, USA, 2013; DISCCO '13. doi:10.1145/2506155.2506159.
79. Ozcelik, B.; Yilmaz, C. Seer: A Lightweight Online Failure Prediction Approach. *IEEE Transactions on Software Engineering* **2016**, *42*, 26–46. doi:10.1109/TSE.2015.2442577.
80. Mariani, L.; Pezzè, M.; Riganelli, O.; Xin, R. Predicting failures in multi-tier distributed systems. *Journal of Systems and Software* **2020**, *161*, 110464. doi:https://doi.org/10.1016/j.jss.2019.110464.
81. Bielefeld, T.C. Online performance anomaly detection for large-scale software systems. Master's thesis, Kiel University, 2012.
82. Frotscher, T. Architecture-based multivariate anomaly detection for software systems. Master's thesis, Kiel University, 2013.
83. Rathfelder, C.; Becker, S.; Krogmann, K.; Reussner, R. Workload-aware System Monitoring Using Performance Predictions Applied to a Large-scale E-Mail System. 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, 2012, pp. 31–40. doi:10.1109/WICSA-ECSA.212.11.
84. van Hoorn, A. Model-driven online capacity management for component-based software systems. PhD thesis, Department of Computer Science, Kiel University, Kiel, Germany, 2014.
85. Brosch, F. Integrated software architecture-based reliability prediction for IT systems. PhD thesis, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, 2012.
86. Uhle, J. On dependability modeling in a deployed microservice architecture. Master's thesis, Universität Potsdam, 2014.
87. Pitakrat, T. Architecture-aware online failure prediction for software systems. PhD thesis, Universität Stuttgart, 2018. doi:http://dx.doi.org/10.18419/opus-9917.
88. Mohamed, A. Software Architecture-Based Failure Prediction. PhD thesis, Queen's University, 2012.
89. Chan, P.K.; Mahoney, M.V.; Arshad, M.H. A machine learning approach to anomaly detection. Technical report, Florida Institute of Technology, 2003.
90. Object Management Group (OMG). UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), 2006.
91. Pertet, S.; Narasimhan, P. Causes of failure in web applications. Technical report, Technical Report CMU-PDL-05-109, Carnegie Mellon University, 2005.
92. Iyer, A.; Zhao, Y. Time series metric data modeling and prediction, 2016. US Patent 9,323,599.
93. Mayle, G.E.; Reves, J.P.; Clubb, J.A.; Wilson, L.F. Automated adaptive baselining and thresholding method and system, 2001. US Patent 6,182,022.
94. Avizienis, A.; Laprie, J..; Randell, B.; Landwehr, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* **2004**, *1*, 11–33. doi:10.1109/TDSC.2004.2.

95. Pitakrat, T.; Van Hoorn, A.; Grunske, L. A comparison of machine learning algorithms for proactive hard disk drive failure detection. Proceedings of the 4th international ACM Sigsoft symposium on Architecting critical systems. ACM, 2013, pp. 1–10.
96. Murray, J.F.; Hughes, G.F.; Kreutz-Delgado, K. Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning Research* **2005**, *6*, 783–816.
97. Eckart, B.; Chen, X.; He, X.; Scott, S.L. Failure prediction models for proactive fault tolerance within storage systems. 2008 IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems. IEEE, 2008, pp. 1–8.
98. Zhu, B.; Wang, G.; Liu, X.; Hu, D.; Lin, S.; Ma, J. Proactive drive failure prediction for large scale storage systems. 2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST). IEEE, 2013, pp. 1–5.
99. Ganguly, S.; Consul, A.; Khan, A.; Bussone, B.; Richards, J.; Miguel, A. A practical approach to hard disk failure prediction in cloud platforms: Big data model for failure management in datacenters. 2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService). IEEE, 2016, pp. 105–116.
100. Wang, Y.; Ma, E.W.; Chow, T.W.; Tsui, K.L. A two-step parametric method for failure prediction in hard disk drives. *IEEE Transactions on Industrial Informatics* **2013**, *10*, 419–430.
101. Züfle, M.; Krupitzer, C.; Erhard, F.; Grohmann, J.; Kounev, S. To Fail Or Not To Fail: Predicting Hard Disk Drive Failure Time Windows. Proceedings of 20th International GI/ITG Conference on Measurement, Modelling and Evaluation of Computing Systems (MMB 2020). Springer, 2020.
102. Chalermarrewong, T.; Achalakul, T.; See, S.C.W. Failure prediction of data centers using time series and fault tree analysis. 2012 IEEE 18th International Conference on Parallel and Distributed Systems. IEEE, 2012, pp. 794–799.
103. Wold, H. A study in the analysis of stationary time series. PhD thesis, Almqvist & Wiksell, 1938.
104. Hyndman, R.J.; Koehler, A.B.; Snyder, R.D.; Grose, S. A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of Forecasting* **2002**, *18*, 439–454.
105. Goodwin, P. The holt-winters approach to exponential smoothing: 50 years old and going strong. *Foresight* **2010**, *19*, 30–33.
106. Herbst, N.R.; Huber, N.; Kounev, S.; Amrehn, E. Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning. *Concurrency and Computation - Practice and Experience, John Wiley and Sons, Ltd.* **2014**, *26*, 2053–2078. doi:10.1002/cpe.3224.
107. De Livera, A.M.; Hyndman, R.J.; Snyder, R.D. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American Statistical Association* **2011**, *106*, 1513–1527.
108. Züfle, M.; Bauer, A.; Herbst, N.; Curtef, V.; Kounev, S. Telescope: A Hybrid Forecast Method for Univariate Time Series. Proceedings of the International work-conference on Time Series (ITISE 2017), 2017.
109. Faloutsos, C.; Flunkert, V.; Gasthaus, J.; Januschowski, T.; Wang, Y. Forecasting Big Time Series: Theory and Practice. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019., 2019.
110. Bauer, A.; Züfle, M.; Grohmann, J.; Schmitt, N.; Herbst, N.; Kounev, S. An Automated Forecasting Framework based on Method Recommendation for Seasonal Time Series. Proceedings of the 11th ACM/SPEC International Conference on Performance Engineering (ICPE 2020); ACM: New York, NY, USA, 2020. doi:10.1145/3358960.3379123.
111. Verma, A.; Ahuja, P.; Neogi, A. pMapper: power and migration cost aware application placement in virtualized systems. Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware. Springer-Verlag New York, Inc., 2008, pp. 243–264.
112. Jung, G.; Hiltunen, M.A.; Joshi, K.R.; Schlichting, R.D.; Pu, C. Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. IEEE 30th International Conference on Distributed Computing Systems (ICDCS 2010). IEEE, 2010, pp. 62–73.
113. Mi, H.; Wang, H.; Yin, G.; Zhou, Y.; Shi, D.; Yuan, L. Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers. IEEE International Conference on Services Computing (SCC 2010). IEEE, 2010, pp. 514–521.

114. Lorido-Botran, T.; Miguel-Alonso, J.; Lozano, J.A. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. *Journal of Grid Computing* **2014**, *12*, 559–592. doi:10.1007/s10723-014-9314-7.
115. Noorshams, Q. Modeling and Prediction of I/O Performance in Virtualized Environments. PhD thesis, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, 2015.
116. Walter, J.; van Hoorn, A.; Kounev, S. Automated and Adaptable Decision Support for Software Performance Engineering. Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools, 2017.
117. Walter, J. Automation in Software Performance Engineering Based on a Declarative Specification of Concern. PhD thesis, University of Würzburg, Germany, 2018.
118. Rygielski, P.; Kounev, S.; Tran-Gia, P. Flexible Performance Prediction of Data Center Networks using Automatically Generated Simulation Models. Proceedings of the Eighth International Conference on Simulation Tools and Techniques (SIMUTools 2015), 2015, pp. 119–128. doi:10.4108/eai.24-8-2015.2260961.
119. Rygielski, P. Flexible Modeling of Data Center Networks for Capacity Management. PhD thesis, University of Würzburg, Germany, 2017.
120. Grohmann, J.; Eismann, S.; Kounev, S. The Vision of Self-Aware Performance Models. 2018 IEEE International Conference on Software Architecture Companion (ICSA-C), 2018, pp. 60–63. doi:10.1109/ICSA-C.2018.00024.
121. Eismann, S.; Grohmann, J.; Walter, J.; von Kistowski, J.; Kounev, S. Integrating Statistical Response Time Models in Architectural Performance Models. Proceedings of the 2019 IEEE International Conference on Software Architecture (ICSA), 2019, pp. 71–80. doi:10.1109/ICSA.2019.00016.
122. Bezemer, C.; Eismann, S.; Ferme, V.; Grohmann, J.; Heinrich, R.; Jamshidi, P.; Shang, W.; van Hoorn, A.; Villavicencio, M.; Walter, J.; Willnecker, F. How is Performance Addressed in DevOps? Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, 2019, pp. 45–50. doi:10.1145/3297663.3309672.
123. Islam, T.; Manivannan, D. Predicting application failure in cloud: A machine learning approach. 2017 IEEE International Conference on Cognitive Computing (ICCC). IEEE, 2017, pp. 24–31.
124. Zheng, Z.; Lan, Z.; Gupta, R.; Coghlan, S.; Beckman, P. A practical failure prediction with location and lead time for blue gene/p. 2010 International Conference on Dependable Systems and Networks Workshops (DSN-W). IEEE, 2010, pp. 15–22.
125. Yu, L.; Zheng, Z.; Lan, Z.; Coghlan, S. Practical online failure prediction for blue gene/p: Period-based vs event-driven. 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W). IEEE, 2011, pp. 259–264.
126. Liang, Y.; Zhang, Y.; Xiong, H.; Sahoo, R. Failure prediction in ibm bluegene/l event logs. Seventh IEEE International Conference on Data Mining (ICDM 2007). IEEE, 2007, pp. 583–588.