An Architecture-aware Approach to Hierarchical Online Failure Prediction

Teerat Pitakrat, Dusăn Okanović, André van Hoorn
Institute of Software Technology
University of Stuttgart, Germany

Lars Grunske
Department of Computer Science, Software Engineering
Humboldt University Berlin, Germany

***Abstract*—Failures in software systems during operation are inevitable. They cause system downtime, which needs to be minimized to reduce or avoid unnecessary costs and customer dissatisfaction. Online failure prediction aims at identifying upcoming failures at runtime to enable proactive maintenance actions. Existing online failure prediction approaches focus on predicting failures of either individual components or the system as a whole. They do not take into account software architectural dependencies, which determine the propagation of failures. In this paper, we propose a hierarchical online failure prediction approach, HORA, which employs a combination of both failure predictors and architectural models. We evaluate our approach using a distributed RSS reader application by Netflix and investigate the prediction quality for two representative types of failures, namely memory leak and system overload. The results show that, overall, our approach improves the area under the ROC curve by 10.7% compared to a monolithic approach.**

I. INTRODUCTION

Quality of Service (QoS) problems in software systems at runtime, such as, performance degradation and service outage, can lead to frustrated customers and losses in revenue. Online failure prediction [1] aims to foresee looming QoS problems at runtime before they manifest themselves. Accurate failure predictions are a prerequisite for preemptive maintenance actions, reducing the effect of problems or even completely preventing them from occurring [2, 3, 4].

Existing online failure prediction approaches predict failures either of the whole system or of specific parts of the system. These approaches employ monolithic models which view the whole system as one entity and predict failure events based on externally observable parameters, e.g., response time [5], event logs [6, 7], or network traffic [8].

When faced with complex software systems comprised of a large number of internal and external components, these approaches may not be able to capture all the influencing parameters. One reason is the fact that failures, which are visible to the users, usually originate from complex interactions and errors inside the system. Driven by the architectural dependencies, these internal errors can propagate to other parts of the software system and cause a chain of errors up to the system boundary [9, 10, 11]. Moreover, if the system undergoes frequent updates or evolution [12], its characteristics can change, causing the prediction models to become outdated and requiring a re-learning of the models.

To overcome these challenges, we hypothesize that online failure prediction can be improved by including architectural information about software systems. There are approaches for architecture-based QoS prediction which are designed for early stages of the development process [13]. Examples of these approaches target performance [14, 15, 16] and reliability [17, 18, 19]. However, they are not designed for running systems.

In this paper, we propose a hierarchical online failure prediction approach, called HORA. The core idea is to combine architectural models with component failure prediction techniques, such as time series forecasting or machine learning. We use two types of architectural models: 1) Architectural Dependency Model (ADM) and 2) Failure Propagation Model (FPM). The ADM captures the dependencies between architectural entities. The FPM employs Bayesian network theory [20] to represent the propagation paths of failures with corresponding probabilities. At runtime, the FPM is constantly updated with individual failure probabilities, which are computed by component failure predictors based on monitoring data. The FPM is then solved to combine individual and propagated failure probabilities.

Our evaluation investigates the prediction quality of HORA with respect to the following research question: Does HORA improve the prediction quality compared to a monolithic approach? If yes, to what degree? The evaluation includes two typical failure scenarios, which are memory leak and system overload. The results of our approach are compared to those of a monolithic approach, which does not consider architectural knowledge for the prediction. The results show that our approach can predict runtime failures with a higher prediction quality, with respect to standard evaluation metrics [1].

To summarize, the paper contributes a novel online failure prediction approach—accompanied by a proof-of-concept implementation—and shows its benefits over a monolithic approach.

The remainder of the paper is structured as follows. Section II emphasizes the challenge of online failure prediction in distributed software systems. Section III details the HORA approach. The evaluation and the discussion of the results are presented in Section IV. Section V describes the related work. Finally, Section VI draws the conclusion and outlines future work. Additionally, we provide supplementary material for this paper [21].

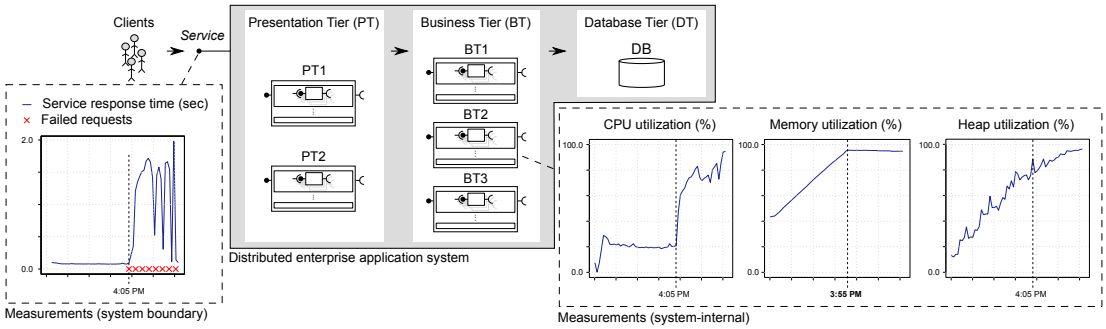


Fig. 1: Running example: high-level three-tier architecture and selected measurements

II. Motivating Example

Software service providers need to make sure that the system satisfies its QoS requirements, e.g., in terms of both response time quantiles and failure rates not exceeding a certain threshold. During operation, such QoS properties are continuously monitored at the system boundary to assess QoS compliance.

Figure 1 provides a high-level view on a typical distributed enterprise application system. The example system conforms to the common three-tier architectural style of enterprise applications. Each of the tiers comprises a number of instances, to which requests are distributed over load-balancers. Each instance comprises a complex stack of software architecture, middleware services, operating system, virtualization, and hardware components.

In this example, it can be observed that at 4:05 PM QoS problems manifest themselves at the system boundary as a prompt increase in response times and failing requests for the provided service. Online failure prediction approaches aim to predict such problems before they occur in order to allow timely actions, such as preventive maintenance, to decrease or completely prevent system downtime. However, in this case, neither of the two metrics measured at the system boundary gives an indication about the upcoming problem. Hence, traditional approaches for online failure prediction are not appropriate in this case.

In addition to the system architecture, Figure 1 includes three system-internal measures of BT2, namely the utilization of CPU, system memory, and heap space of the Java Virtual Machine (JVM). It can be observed that the CPU utilization increases abruptly at 4:05 PM—the same time as the increase of the service response time. The utilization of system memory increases linearly until 3:55 PM when it reaches a level close to 100% and remains stable. The JVM heap space utilization shows an increasing trend until reaching almost 100%. In this scenario, we can conclude that the increase of the response times is caused by the increase of the CPU utilization. The increase of the CPU utilization is in turn caused by garbage collection activity inside the JVM—a common problem in Java systems. In this scenario, the root cause of the failure could be a memory leak in the business-tier instance BT2, which causes a chain of errors [11] that propagates to the end users.

The online failure prediction approach, introduced in this paper, aims to predict this kind of problems by incorporating the failure probabilities of the internal components (in this case CPU, memory, and JVM) along with the failure propagation paths through the software system architecture.

III. The Hora Prediction Approach

The main idea of Hora is derived from the vision [22] that if the failure of each component in the software system can be predicted and the dependencies among the components are known, the consequence, i.e., the propagation, of the failures can also be predicted.

The Hora approach is comprised of two integral parts: 1) two types of architectural models (Section III-A), namely Architectural Dependency Model and Failure Propagation Model; 2) hierarchical online failure prediction (Section III-B), including component failure prediction and inference of failure propagation.

A. Architectural Models

1) Architectural Dependency Model: The Architectural Dependency Model (ADM) lists software and hardware components along with their weighted dependencies—as far as being relevant to the online failure prediction. This intermediate step allows us to focus on one type of model while still providing the possibility to transform other types of architectural models into ADM.

Table I shows the table representation of the ADM for the example introduced in Section II. Following a topological order, the database (DB) has no dependencies to other components; the business-tier instances BT1–3 depend on DB; the presentation-tier instances PT1–2 depend on the business-tier instances BT1–3; and the service depends on PT1–2. Additionally, Table I includes the weights associated to these dependencies—in this case, assuming that requests among the tiers are equally load-balanced to the instances of the next tier. Note that for the sake of simplicity, we consider the six nodes from the example as monolithic components. For realistic scenarios (e.g., in the evaluation in Section IV), these components can be further decomposed into software and hardware components with measures, such as service response time, method execution time, or resource utilization.

TABLE I: Table representation of the ADM for the system in Figure 1

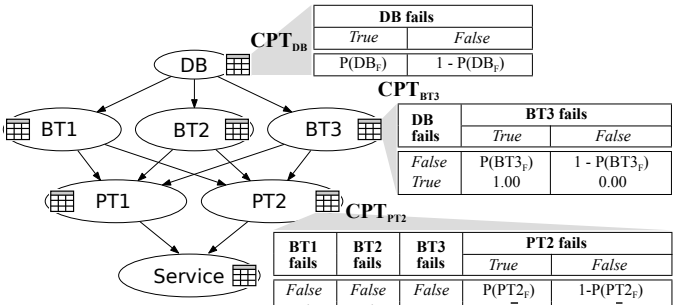
Component	Required components and weights
DB	{}
BT1	{(DB, 1.0)}
BT2	{(DB, 1.0)}
BT3	{(DB, 1.0)}
PT1	{(BT1, $0.3\overline{3}$), (BT2, $0.3\overline{3}$), (BT3, $0.3\overline{3}$)}
PT2	{(BT1, $0.3\overline{3}$), (BT2, $0.3\overline{3}$), (BT3, $0.3\overline{3}$)}
Service	{(PT1, 0.5), (PT2, 0.5)}

There are already a number of existing architectural modeling languages and model extraction mechanisms [23], e.g., PCM [14], Descartes [15], and SLAstic [24]. Working directly with them becomes a challenge due to their different specifications. Thus, we introduce ADM which aims at being an intermediate model representing only the dependencies between architectural entities. The existing architectural models can be transformed into ADM using corresponding transformations.

2) Failure Propagation Model: Although the ADM contains the component dependencies and the corresponding weights, it does not allow efficient failure prediction due to the complex relationships between components. To illustrate this, let us consider the three-tiered system in Figure 1. It can be observed from the architecture that the system allows horizontal scaling to increase capacity and reliability. If some of the nodes in each tier fail, the system is still able to continue the service. However, if the number of failed nodes increases, it will reach a point where the system cannot handle the workload with the reduced capacity. This means that the more nodes fail, the higher the probability that the whole system will fail [11].

To model these complex component dependencies, we transform the ADM into another representation, called Failure Propagation Model (FPM), which is an abstraction that represents the concept of the inference of failure propagations. The FPM employs the formalism of Bayesian networks [20] which is a probabilistic directed acyclic graph that can represent random variables and their conditional dependencies. Figure 2 depicts the Bayesian network which illustrates the relationship of node failures for the example system. Each node in the graph represents a software/hardware component and an arrow represents a causal relationship between components. The relationship implies that a failure of a parent component can cause a failure in the child component. For simplicity reasons, in this example we only consider each physical machine as a node in the graph without going into the details of each machine.

The conditional dependencies between the nodes in the graph are represented by a Conditional Probability Table (CPT). Each node in the graph has a corresponding CPT which contains conditional probabilities of possible failures occurring, given the failure probability of its parent components. For instance, the database is a node that does not depend on any other nodes. Therefore, its CPT contains only two failure probabilities that represent the probability of failure occurring, and not occurring, from inside the database itself. The table



$\mathbf{CPT_{DB}}$

DB fails	
True	*False*
$P(DB_F)$	1 - $P(DB_F)$

$\mathbf{CPT_{BT3}}$

DB fails	BT3 fails	
	True	*False*
False	$P(BT3_F)$	1 - $P(BT3_F)$
True	1.00	0.00

$\mathbf{CPT_{PT2}}$

BT1 fails	BT2 fails	BT3 fails	PT2 fails	
			True	*False*
False	*False*	*False*	$P(PT2_F)$	1-$P(PT2_F)$
False	*False*	*True*	$0.3\overline{3}$	$0.6\overline{6}$
False	*True*	*False*	$0.3\overline{3}$	$0.6\overline{6}$
False	*True*	*True*	$0.6\overline{6}$	$0.3\overline{3}$
True	*False*	*False*	$0.3\overline{3}$	$0.6\overline{6}$
True	*False*	*True*	$0.6\overline{6}$	$0.3\overline{3}$
True	*True*	*False*	$0.6\overline{6}$	$0.3\overline{3}$
True	*True*	*True*	1.00	0.00

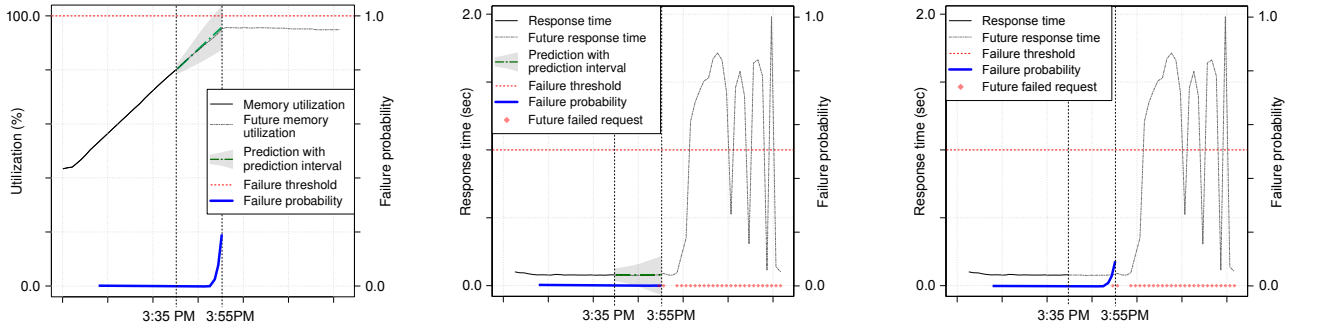
Fig. 2: Failure propagation model for the system in Figure 1 with selected CPTs

is shown in Figure 2 as CPT_{DB}. The failure probability is denoted by $P(\text{DB}_F)$ which is computed at runtime by the corresponding component failure predictor, as will be detailed in Section III-B1. On the other hand, the operation of a business tier requires a functional database with a dependency weight of 1.0, according to the ADM in Table I. This means if the database fails, the business-tier instances will also fail. The CPT of the business-tier instance BT3 is presented in Figure 2 (CPT_{BT3}). The first row indicates the failure probability of the business tier itself, if the database is operating properly. The second row indicates the probability of BT3 failing if the database fails.

A more complex relationship can be seen from the presentation tier which requires at least one business-tier instance. If one business-tier instance fails, the presentation tier can still operate by forwarding requests to the remaining business-tier instances. As listed in Table I, the dependency weight of each presentation-tier instance to each business-tier instance is $0.3\overline{3}$. This implies that, for each business tier failure, the failure probability of the presentation-tier instances will increase by $0.3\overline{3}$. Hence, if all business-tier instances fail, this failure probability will sum up to 1.0 which means that the presentation-tier instances will also fail. The CPT of PT2 is presented in Figure 2 as CPT_{PT2}.

Assuming that a component depends on n other components with $n \geq 1$, the CPT can be expressed by a multiplication of a truth table matrix $\mathbf{T}$ of size $2^n \times n$ and the weight matrix $\mathbf{W}$ of size $n \times 2$, i.e., $\text{CPT} = \mathbf{T} \cdot \mathbf{W}$, where

$$\mathbf{T} = \begin{array}{c} \begin{matrix} c_1 & \dots & c_{n-2} & c_{n-1} & c_n \end{matrix} \\ \begin{pmatrix} 0 & \dots & 0 & 0 & 0 \\ 0 & \dots & 0 & 0 & 1 \\ 0 & \dots & 0 & 1 & 0 \\ 0 & \dots & 0 & 1 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & \dots & 1 & 1 & 1 \end{pmatrix} \end{array}, \mathbf{W} = \begin{pmatrix} w_{c_1} & 1 - w_{c_1} \\ w_{c_2} & 1 - w_{c_2} \\ \vdots & \vdots \\ w_{c_n} & 1 - w_{c_n} \end{pmatrix},$$



(a) Memory failure prediction (monolithic approach)

(b) Service response time failure prediction (monolithic approach)

(c) Service response time failure prediction (HORA)

Fig. 3: Example prediction results: monolithic approach vs. HORA

with $c_i, 1 \leq i \leq n$, are required components and w_{c_i} are the corresponding weights.

The CPTs of other nodes are also created in the same manner as those for the database, business tier, and presentation tier. The complete model with all CPTs is used as a core model to infer about the failure probability of each component and failure propagation. The failure prediction and inference of the model at runtime will be discussed in detail in Section III-B.

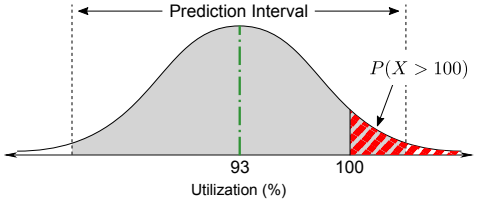


Fig. 4: Probability density function of memory utilization at 3:55 PM and memory failure probability

B. Hierarchical Online Failure Prediction

The prediction of the failures and the inference of their propagation at runtime are comprised of two main steps. The first step is the prediction of individual component failure probabilities (Section III-B1). The second step is the inference of failure propagation based on the individual component failures and the FPM (Section III-B2).

1) Component Failure Prediction: The purpose of component failure predictors is to predict failures of each individual component. Each predictor monitors the runtime measurements of one component and makes a prediction whether the current state might lead to a component failure.

At runtime when a component failure predictor produces a new prediction, the result is used to update the FPM to keep the model up-to-date. Since the prediction result of the component failure predictor indicates the probability of a failure occurring from the component itself, this probability then replaces the first row of the CPT of the corresponding node in the model. For example, if the predictor of BT3 predicts that it may fail in 20 minutes with a probability of 0.8, the probabilities in the first row of CPT_{BT3} in Figure 2, in which DB failure is `False`, will be set to 0.8 and 0.2, accordingly. This process is periodically performed for all component failure predictors.

Figure 3a illustrates the concept of a component failure predictor based on the memory consumption of business tier BT2 in Figure 1. Since the memory consumption is time series data, we employ autoregressive integrated moving average (ARIMA) [25] as a component failure prediction technique. The goal of the prediction is to predict when the memory utilization will reach the 100% threshold, assuming that the machine will have a performance degradation when the memory is depleted, which can cause a service failure. The thin solid line in the graph indicates the monitoring data of memory utilization up to 3:35 PM. The dashed line indicates the prediction of the memory utilization in the next 20 minutes with a prediction interval in light grey.

The probability of the monitoring data crossing the failure threshold α can be computed using the probability density function $f(x)$ of the predicted performance measure:

$$P(X > \alpha) = \int_{\alpha}^{\infty} f(x)dx \tag{1}$$

Figure 4 depicts the probability density function of the memory utilization at 3:55 PM. Assuming that the input data is normally distributed, the prediction error is also normally distributed [26]. Thus, the prediction interval assembles a normal distribution. The predicted value of 93% indicates the mean of the distribution. The 95% prediction interval covers the $\pm 1.96\sigma$ area of the distribution [26]. The probability of the memory utilization crossing the failure threshold at 100% can be computed using Equation 1 with $\alpha = 100$.

In this section we show how HORA employs ARIMA as a component failure predictor. HORA is designed to support any other prediction method, such as other time series forecasting [5] or machine learning techniques [27, 22], as long as they can 1) predict the failure probability and 2) provide the expected time of the failure.

2) Inference of Failure Propagation: The inference of the failure propagation is the last step of HORA, which aims to predict what can be the effects of component failures. Once the component failure probabilities are updated in the model, we

use Bayesian inference [20] to obtain the failure probabilities of all components. The inference of the components' failure probabilities takes into account not only their own failure probabilities but also those of their parents and ancestors. If a node's ancestors have high failure probabilities, its failure probability will also be high. Therefore, the inference allows us to model and predict failure propagation from inside to the outside of the system. At runtime, the inference is done at regular intervals to provide the current failure probabilities of all components.

Figure 3b shows the result of an ARIMA component failure predictor which does not take into account the failure probabilities of other components. It is obvious that this predictor cannot predict the first few occurrences of the service failure. This is because the response time starts increasing exactly when the first failure occurs. Figure 3c illustrates the prediction of HORA which takes into account the architectural dependencies of the components. HORA considers not only the failure probability of the response time but also the failure probability of memory utilization in Figure 3a and the failure propagation model in Figure 2. By comparing failure probabilities in Figure 3a and Figure 3c, it can be observed that the failure probability at 3:55 PM is scaled down from approximately 0.2 to 0.1. This effect is due to the weight of the dependency presented in Table I. As the system contains three instances of the business tier, a failure in one instance does not necessarily have to cause a service failure. Therefore, the failure probability is reduced by the inference of the model.

C. Hora Framework Implementation

According to the concept presented in Section III-A and Section III-B, we have developed a Java-based proof-of-concept implementation. The runtime measurements of the system, including application-level performance and execution traces, as well as resource utilization measurements, are collected by the Kieker monitoring framework [28]. For time series forecasting, the statistical library R [29] is used. The implementation of HORA is available as part of the supplementary material [21].

IV. EVALUATION

This section describes the evaluation of HORA and aims to answer the following research question:

RQ: Does HORA improve prediction quality compared to a monolithic approach? If yes, to what degree?

This research question refers to our main hypothesis that we can improve the quality of online failure prediction using HORA. In order to investigate the degree of improvement, we compare HORA with a monolithic approach which is a set of component failure predictors that do not use the architectural knowledge to propagate the failure probabilities. The evaluation is conducted as a lab experiment with a distributed enterprise application imposed to a synthetic workload under different failure scenarios.

A. Experimental Methodology and Settings

1) System Under Analysis: The HORA approach is evaluated using an extended version of a distributed RSS feed reader application developed by Netflix.[1] This microservice-based application [30] provides a web service where users can view, add, or delete RSS feeds.

Our setup contains two instances in the presentation tier, three instances in the business tier, and one database. The workload driver is set up on a separate node and uses Apache JMeter to generate user requests. The number of concurrent users is set to 150 throughout the experiment. On average, the workload generates approximately 90 requests per second.

The described system is deployed on Emulab [31], which is a large-scale virtualized network testbed. Each of the instances is a physical machine type pc3000 which is equipped with a 3-GHz 64-bit Xeon processor and 2 GB of physical memory, running Ubuntu 14.04.1 LTS and Java 1.7.0 update 75.

2) Experiment Configuration: This section describes the configurations of parameters and models that we use for the evaluation.

a) Architectural Models: The ADM used in the evaluation is similar to that in Table I. Other than the physical nodes in the system, we include additional software and hardware components with the following measures:

- Response times of view, add, and delete operations at the system boundary (frontend load balancer),
- Response times of methods involved in processing requests in all presentation- and business-tier instances,
- Load average, memory utilization, and heap utilization of all physical machines.

The ADM contains 59 entries with 85 dependencies. The FPM is created based on this ADM and, threfore, contains the same number of nodes and dependencies between the nodes. The complete ADM and FPM are not presented here due to the space limitation but are available as part of the supplementary material [21].

b) Failure Thresholds: In our experiment, we classify a service to be in a healthy or failure state by observing the response time and the response itself in 2-minute windows. We consider a service to fail if the following condition occurs within a certain time window: 1) the 95th percentile of the server-side response times of all of the requests in that window exceeds 1 second and/or 2) the ratio of successful requests over all requests falls below 99.99%. Additionally, the execution time threshold of all methods is set to 1 second in the same manner as the server-side response time.

The failures of other architectural entities of the system can be regarded as their intrinsic limitations. The memory utilization threshold is set to 100% according to its physical limit. The heap utilization threshold is set to 90%, since the garbage collector is triggered automatically when utilization becomes too high. The load average represents the number of tasks in the CPU queue over time and provides more

[1] https://github.com/hora-prediction/recipes-rss

information than the CPU utilization [32]. For example, a 1-minute load average of 1.0 means that there is one task in the CPU queue on average in the past minute. Thus, we set the failure threshold of the load average to 1.0.

c) Prediction Technique: At runtime, the monitoring data, containing execution traces and resource measurements, are aggregated into windows of size 2 minutes, which are then pre-processed according to the type of architectural entity measure. The 95th percentile is calculated for the response time and method execution time while the mean is calculated for the load average, memory utilization, and heap utilization.

As stated in Section III-B, we use ARIMA as a component failure predictor. The size of the historical data for ARIMA is set to 20 minutes. The prediction lead time is 10 minutes with a 95% confidence level. The lead time of the FPM is the same as the component failure predictors, which is 10 minutes.

3) Evaluation Metrics: This section introduces the metrics used to evaluate the quality of the prediction. A complete list of metrics and detailed descriptions can be found in [1].

Table II presents the four basic evaluation metrics as a contingency table. A true positive (TP) is a correct prediction of a failure. A false positive (FP) is a prediction of a failure that never occurs. A false negative (FN) is a miss which means the failure occurs but it was not predicted. A true negative (TN) is a correct prediction that a failure does not occur. Moreover, we consider four derived metrics, which are precision, recall or true-positive rate (TPR), false-positive rate (FPR), and accuracy, as listed in Table III.

Receiver Operating Characteristic (ROC) curve [33] represents the quality of the prediction by relating TPRs to FPRs for different prediction thresholds, as shown in Figure 5. A perfect predictor has a curve that covers the entire area of the plot because the TPR is 1 while the FPR is 0. Thus, the closer the curve is to the $(0, 1)$ point, the better the prediction is.

Area Under ROC Curve (AUC) measures the area that is covered by a ROC curve and allows comparison between different ROC curves. A perfect predictor would have an AUC of 1. The AUC is recommended to be used as a single-number metric for evaluating learning algorithms [34]. Thus, in our evaluation, the AUC is used as a representative metric for the comparison of the prediction quality of HORA and the monolithic approach. Furthermore, to evaluate the significance of the improvement, we use two-sided hypothesis testing to compare the AUCs with the following null and alternative hypotheses:

$$\mathrm{H}_0 : \mathrm{AUC}_{\mathrm{HORA}} = \mathrm{AUC}_{Monolithic}$$
$$\mathrm{H}_1 : \mathrm{AUC}_{\mathrm{HORA}} \neq \mathrm{AUC}_{Monolithic}$$

The p-value is reported as the result of the test. The method used in testing is introduced by DeLong et al. [35] to compare two or more AUCs and is implemented in the *pROC* package available in R [29].

4) Failure Scenarios: The types of failures that we consider in our evaluation are two failure types from real world incidents [36], which are memory leak and system overload. We inject one type of these failures into each experiment run and apply HORA and the monolithic prediction approach to predict the failures. Each experiment lasts two hours and is repeated 10 times. The reported evaluation metrics are obtained by combining and analyzing the raw prediction results of all runs. The details of each type of failures are described as follows.

TABLE II: Contingency table

Prediction	**Actual**	
	Failure	**Non-failure**
Failure	True positive (TP)	false positive (FP)
Non-failure	False negative (FN)	True negative (TN)

TABLE III: Selected derived evaluation metrics

Metric	**Formula**
Precision	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$
Recall, True-positive rate (TPR)	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$
False-positive rate (FPR)	$\frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}$
Accuracy	$\frac{\mathrm{TP+TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$

Memory Leak— In the experiment, we introduce a memory leak in one of the business tiers. Every time a request is sent from the presentation tier to this specific instance, 1024 bytes of memory will be allocated and never released.

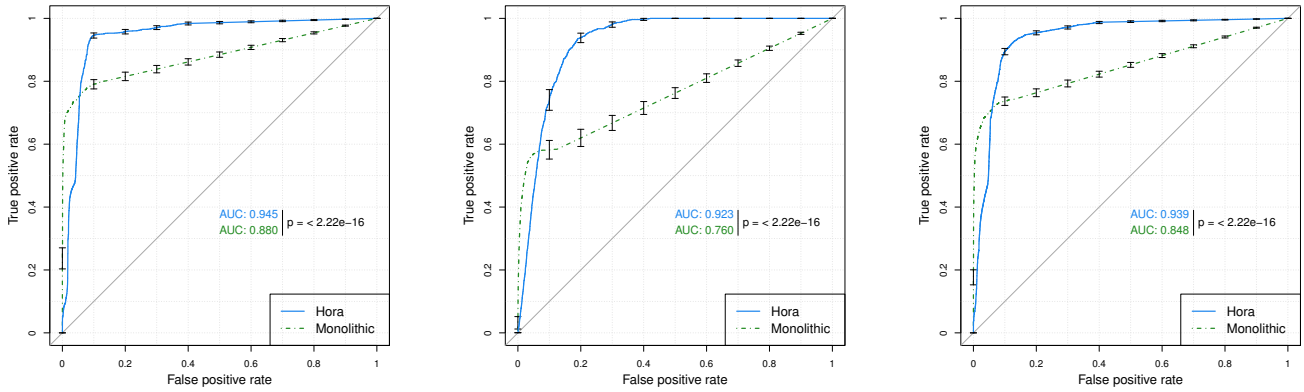
System Overload—System overloads occur when the workload increases, either gradually or abruptly, until the system cannot handle all the incoming requests. In this scenario, instead of using a constant workload, we increase the number of users until service failures occur (detailed in Section IV-A2).

B. Results

In this section, we provide the results and explanation of the failure scenarios. The ROC curves are depicted in Figure 5 and the detailed evaluation metrics of all scenarios are summarized in Table IV. The timeline plots, similar to Figure 3, of all experiment runs are available in the supplementary material [21].

Memory Leak—The memory leak in one of the business-tier instances causes the memory utilization to increase over time. At approximately 75 minutes into the experiment, the garbage collector starts continuously freeing up heap space, which causes the system to slow down. As expected, this causes a sudden increase in the service response time, which cannot be predicted by the response time predictor. On the other hand, HORA considers the memory utilization of the business tier and propagates the failure probability to the service boundary. The result shows that HORA can predict the service failure 10 minutes before it occurs with a failure probability of 0.2. The ROC curves of HORA and the monolithic approach are depicted in Figure 5a.

System Overload— The failures caused by overloading the system start occurring at approximately 80 minutes into the experiment. The increasing number of concurrent users causes the load average of the business-tier instances to exceed the failure threshold. As a result, some of the requests sent from



(a) Memory Leak (b) System overload (c) Overall

Fig. 5: Comparison of ROC curves for the different failure scenarios

TABLE IV: Comparison of all evaluation metrics for the different failure scenarios

Failure type	Prediction approach	Precision	Recall, TPR	FPR	Accuracy	AUC	AUC improvement	p-value
Memory leak	HORA	0.614	0.944	0.095	0.911	0.945	7.3%	$< 2.22 \times 10^{-16}$
	Monolithic	0.645	0.772	0.07	0.908	0.88		
System overload	HORA	0.204	0.936	0.191	0.816	0.923	21.4%	$< 2.22 \times 10^{-16}$
	Monolithic	0.423	0.564	0.042	0.938	0.76		
Overall	HORA	0.437	0.921	0.123	0.881	0.939	10.7%	$< 2.22 \times 10^{-16}$
	Monolithic	0.517	0.726	0.072	0.909	0.848		

the presentation tier to the business tier are rejected. After a pre-defined number of unsuccessful retries, the presentation tier responds with a page indicating that an error has occurred.

The result shows that HORA can predict this type of service failure since it takes into account the dependency of the presentation tier on the business tier. On the other hand, the predictor that observes only the increase in the response time at the system boundary is not able to predict this type of service failure. The ROC curves of both approaches are presented in Figure 5b.

Overall—We evaluate the overall prediction quality of HORA by analyzing the combined raw prediction data of both scenarios. The results in Figure 5c and Table IV show that HORA improves the overall AUC by 10.7%, compared to the monolithic approach.

C. Discussion

Our prediction approach exploits the knowledge of the component dependencies and a set of predictors, which can predict individual component failures, to infer the failure propagation. Our results show that in the both scenarios, HORA can predict the failures with high TPR. This demonstrates that the problems that develop internally can be detected early and the failure probability can be propagated to other parts.

Although the results in Figure 5 and Table IV show that HORA achieves higher TPR and higher AUC, the number of false positives is also high. This results in a low precision and high false-positive rate. In other words, the monolithic approach performs better in the low false-positive-rate range between 0 and 0.05. On the other hand, if a higher false-positive rate is acceptable, HORA will be able to correctly predict more failures than the monolithic approach.

These prediction results are obtained by collecting the monitoring data from the system described in Section IV-A1 and executing the offline analysis on a separate machine equipped with 3.10 GHz Intel Xeon E31220 running Ubuntu 12.04.5 LTS. The analysis for a 2-hour monitoring data is completed in less than 10 minutes. This demonstrates that HORA can be deployed and make timely predictions at runtime.

D. Threats to Validity

We inject failures to introduce problems to the application in our experiments. It is possible that the failures that occur at runtime may be caused by other hidden problems rather than those that we inject. In our evaluation, the failures that occur in the memory leak scenario can be caused by a system overload if the workload is too high. As a result, an attempt to predict failures caused by a memory leak will also predict failures of the system overload problem. Therefore, we carefully choose the workload that is low enough so that it does not cause system overload while we inject other types of failures.

In order to evaluate our approach, we need datasets that include architectural dependencies, detailed runtime measurements for the architectural entities, and information regarding the types and time of the failures. Publicly available datasets exist,[2] but they are not appropriate for our evaluation because they lack the architectural information and runtime measurements. To systematically evaluate our approach, a controlled environment is needed, which includes a usage profile and the types of failures. We conduct a lab study with failure injection which presents two main threats to external validity. First, we consider only one system. Therefore, we

[2]https://www.usenix.org/cfdr

select an open-source application that is representative for the state-of-the-art enterprise systems, in terms of architectural style (microservice-based [30]) and technology (NetflixOSS ecosystem). Second, our experiment did not cover all possible types of failures. Since covering all possible failure types is practically infeasible, we select two representative failure types from real world incidents based on Pertet et al. [36].

V. Related Work

In this paper, we propose a novel approach for hierarchical online failure prediction. This work is related to QoS prediction and can be categorized based on two dimensions; 1) online vs. offline, and 2) monolithic vs. hierarchical.

Online prediction approaches aim at providing information regarding the near future state of the running system based on runtime observations [1]. In contrast, offline prediction approaches are not used to trigger runtime actions, but to focus on providing QoS measures to reason on system design and evolution decisions [37, 38].

In another dimension, monolithic prediction approaches consider the system as a black box. A prediction model can be created using different techniques, such as time series forecasting or machine learning. On the other hand, hierarchical prediction approaches consider the architecture of software systems including the components and their interdependencies. Each component has its own specification that can be combined with the others' to form a model that represents the whole system. The relevant measures of the system can then be obtained by solving the combined model.

The remainder of this section describes related work in three categories, based on the two dimensions, discussed previously. Due to the lack of relevance to our approach, we do not discuss works on monolithic offline prediction.

a) Monolithic Online Prediction: Similar to our approach, Cavallo et al. [39] and Amin et al. [5, 40] use ARIMA and GARCH models to forecast response times and time between failures of web services. In contrast, other approaches use statistical analysis with adaptive thresholds [41], complex event processing [8], rule-based classification [42], linear regression and decision trees [43], support vector machine [44, 45], nearest neighbor methods [44], Bayesian networks and submodels [46, 47, 48], and hidden semi-Markov models [7].

The HORA approach differs from these monolithic approaches at the system level in that we explicitly take architectural information into account. Although we apply similar techniques to predict individual component failures, the architectural knowledge is employed to predict how a failure can propagate and affect other components.

b) Hierarchical Offline Prediction: The approaches in this category employ architecture-based system models annotated with specific quality evaluation models or scenarios [13, 49, 50], e.g., performance [51, 16], reliability [52], and safety [53] attributes. The model can be solved by using analytical solution or simulations to obtain the relevant properties of the whole system.

Cheung [19] proposes a seminal software reliability model that takes into account reliability of individual components along with the probability of calling other components. A Markov model is employed to combine the reliability of components and represent the reliability of the whole system. Cortellessa and Grassi [10] present an approach for reliability analysis of component-based software systems. Based on the system architecture, they consider the error propagation probability between components in addition to the reliability of individual components. Becker et al. [14] introduce the Palladio Component Model (PCM) which enables performance prediction of component-based software systems. Brosch et al. [17] extend the PCM by annotating the components with corresponding reliability attributes. The model is transformed into a discrete-time Markov chain and solved to obtain the reliability of the system.

The approaches in this category have shown that the capability of the prediction can be significantly improved by combining traditional approaches, which are monolithic, with the architecture-based models. Our approach applies the same concept and incorporates the architecture of the system, specifically the dependencies between components, into online failure prediction. In combination with monolithic prediction approaches, our approach is able to predict both failures of individual components and the probabilities that those failures will propagate to other parts of the system.

c) Hierarchical Online Prediction: As one recent example for performance, Brosig et al. [15] employ an architecture-based performance model to predict system performance at runtime for capacity planning and online resource provisioning. The performance characteristics are captured in an architectural performance model which is then solved by transforming it to an analytical model or by simulation, similar to Becker et al. [14]. As opposed to this, HORA focuses on predicting failure occurrences using an extensible set of tailored prediction techniques.

Chalermarrewong et al. [54] predict system unavailability in data centers using a set of component predictors and fault tree analysis. The component predictors employ autoregressive moving average (ARMA) to predict failures of hardware components. These component failures are leaf nodes in the fault tree which is evaluated to conclude whether the current set of component failures will lead to system unavailability. Even though this work does not consider software, it shares the same basic idea as HORA by having a dedicated failure predictor for each component. However, the fault tree does not incorporate the conditional probability which represents complex software architectural relationships. On the other hand, HORA employs Bayesian networks which can represent conditional dependencies and infer the probabilities of failures and their propagation.

VI. Conclusion

Failures in software systems usually develop inside the system and propagate to the boundary. Existing online failure prediction approaches do not explicitly consider the software

system architecture and failure propagation paths. In this paper, we introduce our hierarchical online failure prediction approach, HORA, which employs a combination of a failure propagation model and component failure prediction techniques. The failure propagation model uses Bayesian networks and is extracted from an architectural dependency model. The component failure predictors are updated by continuous measurements of the running system. Our evaluation shows that HORA provides a significantly higher prediction quality than the monolithic approach. In addition to the improved prediction quality, HORA allows a higher degree of modularity as different failure prediction techniques can be applied and reused among similar types of system components. This makes HORA an appropriate online failure prediction approach for software systems, particularly with short release cycles.

In our future work, we plan to investigate the effect of the granularity of the architectural dependency model and develop an incremental solution technique for increasing the efficiency of the failure propagation model. We will also incorporate structural changes and update the failure propagation model at runtime. Moreover, we plan to extend our evaluation setup to a benchmark for online failure prediction approaches.

ACKNOWLEDGMENT

This work has been partly funded by the German Federal Ministry of Education and Research (grant no. 01IS15004).

REFERENCES

[1] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Computing Surveys*, vol. 42, no. 3, pp. 10:1–10:42, 2010.

[2] Y. Brun, J. Y. Bang, G. Edwards, and N. Medvidovic, "Self-adapting reliability in distributed software systems," *IEEE Trans. on Software Engineering*, vol. 41, no. 8, pp. 764–780, 2015.

[3] R. Calinescu, L. Grunske, M. Z. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS management and optimization in service-based systems," *IEEE Trans. on Software Eng.*, vol. 37, no. 3, pp. 387–409, 2011.

[4] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot - a technique for cheap recovery," in *Proc. Symposium on Operating Systems Design & Implementation (OSDI '04)*, 2004, pp. 31–44.

[5] A. Amin, A. Colman, and L. Grunske, "An approach to forecasting QoS attributes of web services based on ARIMA and GARCH models," in *Proc. 19th Int. Conf. on Web Services (ICWS '12)*, 2012, pp. 74–81.

[6] T. Pitakrat, J. Grunert, O. Kabierschke, F. Keller, and A. van Hoorn, "A framework for system event classification and prediction by means of machine learning," in *Proc. 8th Int. Conf. on Performance Evaluation Methodologies and Tools (VALUETOOLS '14)*, 2014.

[7] F. Salfner and M. Malek, "Using hidden semi-Markov models for effective online failure prediction," in *Proc. 26th Int. Symposium on Reliable Distributed Systems (SRDS '07)*, 2007, pp. 161–174.

[8] R. Baldoni, G. Lodi, L. Montanari, G. Mariotta, and M. Rizzuto, "Online black-box failure prediction for mission critical distributed systems," in *Proc. 31st Int. Conf. on Computer Safety, Reliability and Security (SAFECOMP '12)*, 2012, pp. 185–197.

[9] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[10] V. Cortellessa and V. Grassi, "A modeling approach to analyze the impact of error propagation on reliability of component-based systems," in *Proc. 10th Int. Conf. on Comp.-Based Soft. Eng. (CBSE '07)*, 2007, pp. 140–156.

[11] M. Nygard, *Release It!: Design and Deploy Production-Ready Software*. Pragmatic Bookshelf, 2007.

[12] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. Addison-Wesley Prof., 2015.

[13] M. A. Babar and I. Gorton, "Comparison of scenario-based software architecture evaluation methods," in *Proc. 11th Asia-Pacific Software Engineering Conf. (APSEC '04)*, 2004, pp. 600–607.

[14] S. Becker, H. Koziolek, and R. Reussner, "The Palladio component model for model-driven performance prediction," *Journal of Systems and Software*, vol. 82, no. 1, pp. 3–22, 2009.

[15] F. Brosig, N. Huber, and S. Kounev, "Architecture-level software performance abstractions for online performance prediction," *Science of Computer Programming*, vol. 90, Part B, pp. 71–92, 2014.

[16] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Performance Evaluation*, vol. 67, no. 8, pp. 634–658, 2010.

[17] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner, "Architecture-based reliability prediction with the Palladio Component Model," *IEEE Trans. on Software Engineering*, vol. 38, no. 6, pp. 1319–1339, 2012.

[18] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik, "Early prediction of software component reliability," in *Proc. 30th Int. Conf. on Software Engineering (ICSE '08)*, 2008, pp. 111–120.

[19] R. C. Cheung, "A user-oriented software reliability model," *IEEE Trans. on Software Engineering*, vol. 6, no. 2, pp. 118–125, 1980.

[20] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[21] T. Pitakrat, D. Okanovi, A. van Hoorn, and L. Grunske, "An Architecture-aware Approach to Hierarchical Online Failure Prediction," Apr. 2016. [Online]. Available: http://dx.doi.org/10.5281/zenodo.46425

[22] T. Pitakrat, A. van Hoorn, and L. Grunske, "Increasing dependability of component-based software systems by online failure prediction," in *Proc. Euro. Dependable Computing Conf. (EDCC '14)*, 2014, pp. 66–69.

[23] B. R. Schmerl, J. Aldrich, D. Garlan, R. Kazman, and H. Yan, "Discovering architectures from running systems," *IEEE Trans. on Software Engineering*, vol. 32,

no. 7, pp. 454–466, 2006.

[24] A. van Hoorn, "Model-driven online capacity management for component-based software systems," Ph.D. dissertation, 2014, faculty of Engineering, Kiel University.

[25] R. H. Shumway and D. S. Stoffer, *Time series analysis and its applications*. Springer Science, 2013.

[26] D. C. Montgomery, G. C. Runger, and N. F. Hubele, *Engineering statistics*. John Wiley & Sons, 2009.

[27] T. Pitakrat, A. van Hoorn, and L. Grunske, "A comparison of machine learning algorithms for proactive hard disk drive failure detection," in *Proc. 4th Int. Conf. on Architecting Critical Systems (ISARCS '13)*. ACM, 2013, pp. 1–10.

[28] A. van Hoorn, J. Waller, and W. Hasselbring, "Kieker: A framework for application performance monitoring and dynamic software analysis," in *Proc. Int. Conf. on Performance Eng. (ICPE '12)*, 2012, pp. 247–248.

[29] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2015. [Online]. Available: http://www.R-project.org/

[30] S. Newman, *Building Microservices*. O'Reilly Media, Inc., 2015.

[31] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Large-scale virtualization in the Emulab network testbed." in *USENIX Annual Technical Conference*, 2008, pp. 113–128.

[32] R. Walker, "Examining load average," *Linux Journal*, vol. 2006, no. 152, pp. 5–16, 2006.

[33] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006.

[34] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.

[35] E. R. DeLong, D. M. DeLong, and D. L. Clarke-Pearson, "Comparing the areas under two or more correlated receiver operating characteristic curves: A nonparametric approach," *Biometrics*, vol. 44, no. 3, pp. 837–845, 1988.

[36] S. Pertet and P. Narasimhan, "Causes of failure in web applications," *Parallel Data Laboratory*, p. 48, 2005.

[37] V. Cortellessa, A. Di Marco, and P. Inverardi, *Model-based software performance analysis*. Springer, 2011.

[38] J. D. Musa, *Software reliability engineering*. McGraw-Hill, 1998.

[39] B. Cavallo, M. D. Penta, and G. Canfora, "An empirical comparison of methods to support QoS-aware service selection," in *Proc. 2nd Int. Workshop on Principles of Engineering Service-Oriented Systems (PESOS '10)*. ACM, 2010, pp. 64–70.

[40] A. Amin, L. Grunske, and A. Colman, "An automated Approach to Forecasting QoS Attributes based on linear and non-linear Time Series Modeling," in *Proc. 27th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE '12)*, 2012, pp. 130–139.

[41] A. Bovenzi, F. Brancati, S. Russo, and A. Bondavalli, "A statistical anomaly-based algorithm for on-line fault detection in complex software critical systems," in *Proc. 30th Int. Conf. on Computer Safety, Reliability, and Security (SAFECOMP '11)*, 2011, pp. 128–142.

[42] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam, "Critical event prediction for proactive management in large-scale computer clusters," in *Proc. 9th Int. Conf. on Knowledge Discovery and Data Mining (KDD '03)*, 2003, pp. 426–435.

[43] J. Alonso, J. Torres, and R. Gavalda, "Predicting web server crashes: A case study in comparing prediction algorithms," in *Proc. 5th Int. Conf. on Autonomic and Autonomous Systems (ICAS '09)*, 2009, pp. 264–269.

[44] Y. Liang, Y. Zhang, H. Xiong, and R. K. Sahoo, "Failure prediction in IBM BlueGene/L Event Logs," in *Proc. Int. Conf. on Data Mining (ICDM '07)*, 2007, pp. 583–588.

[45] I. Fronza, A. Sillitti, G. Succi, M. Terho, and J. Vlasenko, "Failure prediction based on log files using random indexing and support vector machines," *Journal of Systems and Software*, vol. 86, no. 1, pp. 2–11, 2013.

[46] Q. Guan, Z. Zhang, and S. Fu, "Ensemble of Bayesian predictors and decision trees for proactive failure management in cloud computing systems," *Journal of Communications*, vol. 7, no. 1, pp. 52–61, 2012.

[47] Y. Watanabe, H. Otsuka, M. Sonoda, S. Kikuchi, and Y. Matsumoto, "Online failure prediction in cloud datacenters by real-time message pattern learning," in *Proc. 4th Int. Conf. on Cloud Computing Technology and Science (CloudCom '12)*, 2012, pp. 504–511.

[48] S. Fu and C.-Z. Xu, "Exploring event correlation for failure prediction in coalitions of clusters," in *Proc. 2007 Conf. on Supercomputing (SC '07)*, 2007, pp. 41:1–41:12.

[49] M. A. Babar, L. Zhu, and D. R. Jeffery, "A framework for classifying and comparing software architecture evaluation methods," in *Australian Soft. Eng. Conf. (ASWEC 2004)*. IEEE Computer Society, 2004, pp. 309–319.

[50] L. Grunske, "Early quality prediction of component-based systems - A generic framework," *Journal of Systems and Software*, vol. 80, no. 5, pp. 678–686, 2007.

[51] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Trans. on Software Engineering*, vol. 30, no. 5, pp. 295–310, 2004.

[52] K. Goseva-Popstojanova and K. S. Trivedi, "Architecture-based approach to reliability assessment of software systems," *Perf. Eval.*, vol. 45, no. 2-3, pp. 179–204, 2001.

[53] L. Grunske and J. Han, "A comparative study into architecture-based safety evaluation methodologies using AADL's error annex and failure propagation models," in *Proc. 11th IEEE High Assurance Systems Engineering Symposium (HASE) '08*, 2008, pp. 283–292.

[54] T. Chalermarrewong, T. Achalakul, and S. See, "Failure prediction of data centers using time series and fault tree analysis," in *Proc. 18th Int. Conf. on Parallel and Distributed Systems (ICPADS '12)*, 2012, pp. 794–799.