Hora: Online Failure Prediction Framework for Component-based Software Systems Based on Kieker and Palladio

Teerat Pitakrat

Institute of Software Technology
University of Stuttgart
Universitätstraße 38
70659 Stuttgart, Germany
pitakrat@informatik.uni-stuttgart.de

Abstract: Predicting failures in large systems at runtime is a challenging task as the systems usually comprise a number of hardware and software components with complex structures and dependencies. The state-of-the-art techniques approach the task of failure prediction either by creating one separate prediction model for each crucial parameter, or by aggregating parameters of all components in order to build one prediction model. However, both approaches are not suitable for large-scale dynamic systems. In this paper, we propose our envisioned approach *Hora*—an online failure prediction approach for large-scale component-based systems. Our approach creates a prediction sub-model for each component and combines them using component dependencies obtained from an architectural model, e.g., PCM or SLAstic, into a model that can predict failures of internal and external services at runtime. The framework is built on the Kieker monitoring framework and other supporting tools, e.g., Weka, R, ΘPAD, which allows suitable prediction techniques to be employed for different components of the system.

1 Introduction

Preventing failures from occurring at runtime is one of the main goals when building a system. There are many techniques which improve the system's reliability during the development to ensure that they can function properly under various circumstances However, despite the effort put into the systems, the reliability still may not reach 100 percent. This implies that there are occasions when the systems in production fail to deliver expected services. The main reasons that cause the systems to fail are essentially the complex structure and inter-dependencies of the components. A failure or even a partial failure of one service can cause other services that depend on it to malfunction. This incident can create a chain of service failures that propagates until it reaches critical components and causes the system to fail.

Online failure prediction [SLM10] is an approach that aims to foresee imminent problems by analyzing the monitoring data collected from the system at runtime. The prediction is

Proc. Kieker/Palladio Days 2013, Nov. 27–29, Karlsruhe, Germany
Available online: http://ceur-ws.org/Vol-1083/

based on the knowledge of previous failure occurrences and tries to learn which patterns of events can lead to a problem. However, the techniques in this approach view the system as a black box and create the prediction models without utilizing architectural information.

In this paper, we introduce our envisioned online failure prediction approach for component-based software systems called *Hora*[1]. The primary concept of our approach is based on the traditional online failure prediction which collects system parameters at runtime and uses them to build a prediction model. However, we aim to include architectural models, e.g., PCM [BKR09] or SLAstic [vH13], which provide the system architecture and the dependencies between components into the construction of prediction models. Furthermore, to improve the data collection capability, we employ Kieker [vHWH12] as a monitoring tool to collect more detailed information from the system at runtime, and as a foundation for *Hora*'s framework architecture. This paper extends and includes materials from our previous work [PvHG13b] and focuses on the architecture of our online failure prediction approach in details.

The rest of the paper is organized as follows. Section 2 presents the related work on online failure prediction and architecture-based QoS prediction. Section 3 provides an overview of our online failure prediction approach including the component- and system-level prediction models. Section 4 describes the extraction of the architectural model from a system and how they can be transformed into a prediction model. The architecture of our prediction framework is detailed in Section 5. Section 6 draws the conclusions and outlines future work.

2 Related Work

The related work can be grouped into two main categories. The first category is based on an online failure prediction approach [SLM10] which creates prediction models from observable parameters (e.g., response time, log files) of production systems and uses the model to make predictions. The techniques used to create the prediction models can be further divided into two subcategories. The first subcategory employs time series forecasting techniques to predict future observations of crucial system parameters, e.g., response time [ACG12] or memory usage [GLVT06]. The second subcategory uses machine learning and data mining techniques to create the prediction models and predict or classify whether the system in its current state is going to encounter a problem [LZXS07, PvHG13a].

The second category is architecture-based QoS evaluation which uses design-oriented models to predict QoS parameters, such as, performance, reliability, or resource efficiency. These techniques originally aim at predicting the QoS parameters during the development phase before the system can be deployed or in offline mode where the data are collected and analyzed at a later time. The prediction results of these techniques can be obtained through analytical solution or simulations of the models. Brosch [Bro12] predicts the system reliability by annotating PCM [BKR09] with reliability parameters.

[1] Hora comes from a Sanskrit word which means horoscope

The annotated PCM is then transformed into a discrete-time Markov chain and solved using PRISM [HKNP06]. Although the approach aims at the prediction at design time, the parameters of the models can be kept synchronized with the actual system at runtime which also makes them applicable for online prediction and fault localization. Marwede *et al.* [MRvHH09] localize the faulty component of the system by analyzing call dependencies. An anomaly score of each component is calculated from the response time that the component takes to process a request. When an anomalous component is identified, that component is further analyzed by inspecting the sub-components recursively until the root cause is found. Even though this technique originally aims for on-demand analysis, it can also be applied to production systems to diagnose failures at runtime.

3 Overview of the Approach

Our online failure prediction approach is based on an assumption that a failure can propagate from one service to another if there is a dependency between them. As a consequence, a failure can cause other dependent services to fail, which creates a chain of failures that could propagate to the system boundary. If the failures can be predicted on the service level, e.g., predicting failures of other services given that one service of a component has failed, we can obtain the projection of the chain which shows the possible service failure paths that can lead to a system failure. To be able to make such predictions, the knowledge about the system architecture and component dependencies need to be integrated into the failure prediction model. Hence, in our approach we split the prediction model into sub-models where each one is responsible for predicting service failures of one component of the system. The result of the sub-models are then combined into a system-level model which analyzes and predicts whether the failures of some services will cause other services or the whole system to fail.

Figure 1 illustrates an example component-based software system which is used throughout the paper. The system is composed of three software components, C_1, C_2, and C_3, being deployed on two hardware components, H_1 and H_2. C_2 and C_3 each provides a software service which is required by other internal components (C_1 and C_2, respectively). On the other hand, C_1 provides two services S_1 and S_2 to the users at the system boundary. The failures considered in our approach concern all services of the components which include both hardware failures (e.g., CPU, hard drives, network connections) and software failures (e.g., bugs). For example, a failure of H_2 can cause C_3 to fail, which, in turn, can cause C_2 and C_1 to fail resulting in a failure of services S_1 and S_2.

In order to create a prediction model of the system, our approach first obtains the architectural model (e.g., PCM) which may be refined by dynamic analysis at runtime employing the Kieker framework. The component dependencies are extracted from the model and transformed into an intermediate representation in a form of component dependency table (Table 1). The table acts as an intermediate step for different architectural models and the prediction models. It provides a list of interface hardware and software component services and their dependencies which are used during the construction of the prediction model and the sub-models. It is also worth noting that when the system is reconfigured

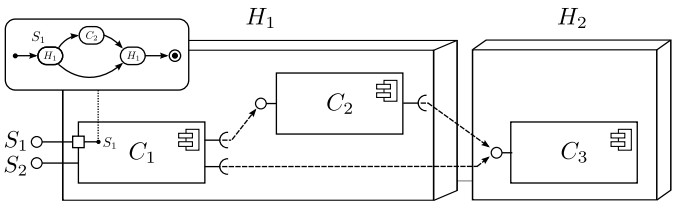

Figure 1: Architectural model of an example system

at runtime which causes changes in the architecture, the component dependency table and the prediction models are updated to keep the dependency information synchronized with the actual system. The workflow of the approach is depicted in Figure 2.

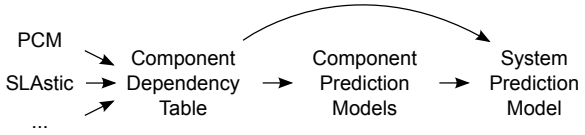

Figure 2: Workflow of the approach

3.1 Component-level Prediction Model

The goal of a component-level prediction model is to predict whether and when the services of a component of the system will encounter a problem in the near future. The services can be monitored at runtime using Kieker which collects operational data such as response times or error messages of the component. The collected data can reflect different states of a component's services and can be used to create a prediction model. The models can be constructed using techniques that are suitable to the available data. For instance, if a component provides a service of which response time is crucial, time series forecasting techniques, such as ARIMA, can be used to create the prediction model based on the historical observations. On the contrary, if the component produces log files, the messages contained in the log can be used to create classification models by machine learning techniques, such as naive Bayes classifier or decision trees.

3.2 System-level Prediction Model

When the component-level prediction models make a prediction regarding the failure of the services, the results are fed into a system-level prediction model which analyzes the propagation sequence of service failures and predicts whether this particular sequence can lead to larger problems. The sequence of service failures could be modeled by a Markov chain where each state represents one service failure and when a failure occurs, the model

transitions into the corresponding state. However, large systems can have many levels of dependencies and a failure of one service can cause other services to fail in a particular order. The sequence of the failures therefore becomes an important piece of information which could reveal the root cause and predict which component may fail in the near future. This leads to a limitation of a traditional Markov chain as the chain is memoryless and cannot capture the sequence of length longer than two. Moreover, the chain allows the system to be only in one state at any time. This means that the model cannot represent the scenarios in which multiple services of the system are failing concurrently and independently. For example, the hardware component H_2 and software component C_1 in Figure 1 can encounter intrinsic problems which are not caused by the environment and fail at the same time. To solve the aforementioned problems, our approach employs a continuous-time transition system of order *n* where each state represents a sequence of service failures of at most length *n*. The states of the model are determined from the component dependency table extracted from the architectural models as shown in Table 1. The states that represent service failure sequences which cannot occur according to the architectural model will not be included, thus, reducing the size of the model. Furthermore, the transition system allows multiple tokens, i.e., it can represent different states of independent services at the same time.

	S_1	S_2	$C_{1.S_1}$	$C_{1.S_2}$	C_2	C_3	H_1	H_2
S_1			•					
S_2				•				
$C_{1.S_1}$					•		•	
$C_{1.S_2}$						•	•	
C_2						•	•	
C_3								•
H_1								
H_2								

Table 1: Dependencies between architectural components

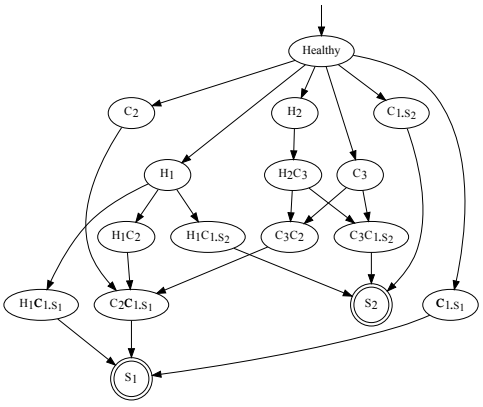

Figure 3: System-level prediction model

After the structure of the model is determined, the system and its components are monitored in order to obtain and learn the transition probabilities between the states. The

continuous-time transitions enable the model to make predictions with more precise timing, e.g., time window in which a service may fail. When failures of services can be predicted individually by component prediction models, the system model can utilize this information and predict which services are most likely to fail next and what is the probability of this failure leading to the failure of the services at the system boundary. Figure 3 illustrates a transition model of order two which considers sequences of at most two successive service failures based on the example system in Figure 1. In a normal state when all components and services are working properly, it stays in the healthy state which is the initial state of the model. If the service of component C_2 is predicted to fail, the model transitions into state C_2 and calculates the probability of the failure of C_2 causing other dependent services to fail, which, in this example, is the failure of service S_1 of component C_1 (state $C_2C_{1.S_1}$).

At runtime when the system is in production, it is possible that the system may evolve, resulting in architectural and configuration changes. To keep the prediction model synchronized, the dependencies between components and services can be re-evaluated and the system-level prediction model can be updated to represent the actual service dependencies. As the transition system is transparent, it allows new states to be added or the existing ones to be removed while affecting only a small part of the model, i.e., the adjacent states of the added or removed nodes.

4 Model Extractions and Transformations

The construction of component- and system-level prediction models described in Section 3 requires that architectural information about the system, i.e., system components and their dependencies, is available. We currently consider the use of two architectural modeling languages for component-based software systems, namely PCM [BKR09] and SLAstic [vH13]. Both languages provide similar modeling concepts split into architectural viewpoints for types, system assembly, execution environment, and deployment. While PCM is a full-blown approach for design-time performance prediction, SLAstic focuses on architecture-based online adaptation using the models at runtime. This section provides a description how architectural models can be obtained (Section 4.1) and how they can be transformed into a failure prediction model (Section 4.2).

4.1 Extraction of Architectural Models

The intuitive way of obtaining architectural models is the manual creation. However, this procedure is time-consuming and error-prone. Moreover, chances are high that the system and the model diverge due to system changes, e.g., maintenance activities or system evolution. As an alternative, approaches exist to extract architectural models using automatic techniques employing static analysis—e.g., based on source code artifacts—and dynamic analysis—e.g., based on collected system traces— or by hybrid analysis, i.e., a

combination of both.

We plan to apply and, if needed, extend existing techniques to extract SLAstic and PCM models from monitoring data even though we might combine them with static approach, such as SoMox [BHT+10] or ArchiMetrix [PvDB12]. For SLAstic models, an automatic extraction approach for Kieker-based monitoring data exists [vH13]. Due the integration in the SLAstic framework, runtime changes of the system are supported. Moreover, the SLAstic approach includes a transformation from SLAstic models to PCM models. These models are not complete in that they can be used for PCM-based performance prediction; however, they include the architectural information needed for our approach.

4.2 Model Transformations

The architectural information included in the PCM or SLAstic model can be used to generate a dependency table as shown in Table 1. Example dependencies extracted from the architectural models are dependencies among software services obtained via service implementation details (RDSEFFs in PCM) and the information about component assembly. Dependencies from software components or services to hardware components are obtained by taking resource demand and deployment information into account. The table serves as an intermediate model that provides sufficient information for building the system-level prediction model. We aim to employ the existing transformation of the PCM to Markov chain for reliability prediction [Bro12] and extend it for the continuous-time transition systems presented in Section 3.2.

5 Prediction Framework Architecture

Our online prediction framework, Hora, is currently being developed based on the Kieker architecture as illustrated in Figure 4. The monitoring data is obtained through the Kieker monitoring framework in form of monitoring records. As the prediction model is split into sub-models, the records are forwarded to the corresponding component-level prediction models. Each model makes a prediction, based on the monitoring data, whether and when the component or the service is likely to encounter a problem and passes the prediction result to the system-level prediction model. The system model collects the results from all component models and predicts whether the sequence of service failures will affect the entire system.

In this framework, the component-level prediction models are developed as filters in the pipe-and-filter architecture of Kieker which make the approach flexible if new components or services are added or the existing ones are removed. Various techniques or tools can be employed as a component-level predictor, such as, ΘPAD [Bie12], WEKA [HFH+09], or Esper [EE12]. We also aim to integrate our previous work on hard drive failure prediction using machine learning [PvHG13a] into the Hora framework. Furthermore, we are developing a failure prediction technique based on system event log using machine learn-

ing techniques (see, e.g., [GCK12, LZXS07]) which conforms to the framework as shown in Figure 4 as *event log analyzer*.

The system-level prediction model is placed at the end of the prediction process to analyze the prediction results of the component predictors and make a final prediction for the overall system. As described in Section 3.2, the system prediction model is a transition model which allows concurrent and independent service failures. When a new set of monitoring data is available, the component-level prediction models can predict failures and pass the prediction results to the system model independently from other component models. The system-level prediction model then analyzes the failure sequence of services and make a prediction whether and when other services may fail.

In addition to predicting failures of the system, Hora also allows the architecture to change at runtime. The new architecture can be obtained either by static or dynamic analysis described in Section 4 and transformed to a new component dependency table. The system-level prediction model, which includes the states and transition probabilities, is then updated according to the new component dependencies.

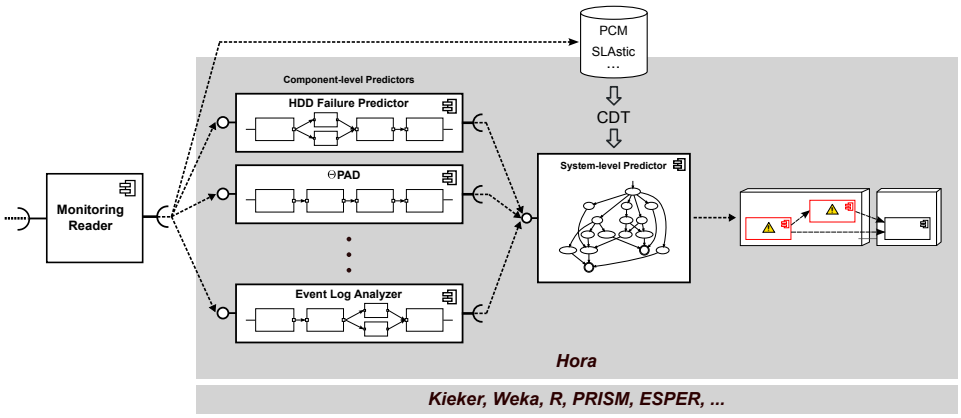

Figure 4: Hora architecture

6 Conclusion

Online failure prediction is an approach that predicts pending problems which may occur in the near future causing the system to fail. Existing techniques create prediction models either by considering only crucial parameters of the system, or by aggregating all of the available parameters. However, the architectural knowledge of the system is rarely used to create online failure prediction models. This paper presents our envisioned approach, Hora, which constructs prediction models by integrating component dependency information obtained from the architectural models of the system. The architectural models used

in this approach are PCM and SLAstic which can be created manually, or extracted from the system using static or dynamic analysis provided by Kieker. The component dependency information is used to create a continuous-time transition model which captures the sequences of service failures and predicts whether it will affect the whole system. Furthermore, the model provides not only failure warnings but also the probability, expected time, root cause, and the possible manifestation of the failure.

As the next steps, we aim to realize the Hora framework which includes model extraction and transformation of PCM and SLAstic to prediction models, and integrating prediction algorithms and tools into the pipe-and-filter architecture. Simulations and lab experiments will be carried out to evaluate the applicability and the prediction performance of our approach.

References

[ACG12] Ayman Amin, Alan Colman, and Lars Grunske. An Approach to Forecasting QoS Attributes of Web Services Based on ARIMA and GARCH Models. In *Proceedings of the 19th IEEE International Conference on Web Services*, pages 74–81, June 2012.

[BHT+10] Steffen Becker, Michael Hauck, Mircea Trifu, Klaus Krogmann, and Jan Kofroň. Reverse Engineering Component Models for Quality Predictions. In *Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 194–197, 2010.

[Bie12] Tillmann Carlos Bielefeld. Online performance anomaly detection for large-scale software systems. Master's thesis, March 2012. Diploma Thesis, Kiel University.

[BKR09] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio Component Model for Model-Driven Performance Prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.

[Bro12] Franz Brosch. *Integrated Software Architecture-Based Reliability Prediction for IT Systems*. PhD thesis, Institut für Programmstrukturen und Datenorganisation (IPD), Karlsruher Institut für Technologie, Karlsruhe, Germany, June 2012.

[EE12] Esper Team and EsperTech, Inc. Esper 4.9.0 Reference Documentation. http://esper.codehaus.org/esper/documentation, 2012.

[GCK12] Ana Gainaru, Franck Cappello, and William Kramer. Taming of the Shrew: Modeling the Normal and Faulty Behaviour of Large-scale HPC Systems. In *Proceedings of the 26th IEEE International Parallel Distributed Processing Symposium (IPDPS)*, pages 1168–1179, 2012.

[GLVT06] Michael Grottke, Lei Li, Kalyanaraman Vaidyanathan, and Kishor S. Trivedi. Analysis of Software Aging in a Web Server. *IEEE Transactions on Reliability*, 55(3):411–420, 2006.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[HKNP06] Andrew Hinton, Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proceeding of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS06), volume 3920 of LNCS*, pages 441–444. Springer, 2006.

[LZXS07] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra K. Sahoo. Failure Prediction in IBM BlueGene/L Event Logs. In *Proceedings of the 7th IEEE International Conference on Data Mining*, pages 583–588, 2007.

[MRvHH09] Nina Marwede, Matthias Rohr, André van Hoorn, and Wilhelm Hasselbring. Automatic Failure Diagnosis Support in Distributed Large-Scale Software Systems Based on Timing Behavior Anomaly Correlation. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering*, pages 47–58, 2009.

[PvDB12] Marie Christin Platenius, Markus von Detten, and Steffen Becker. Archimetrix: Improved Software Architecture Recovery in the Presence of Design Deficiencies. In *Proceedings of the 16th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 255–264, 2012.

[PvHG13a] Teerat Pitakrat, André van Hoorn, and Lars Grunske. A comparison of machine learning algorithms for proactive hard disk drive failure detection. In *Proceedings of the 4th International ACM Sigsoft Symposium on Architecting Critical Systems*, pages 1–10. ACM, 2013.

[PvHG13b] Teerat Pitakrat, André van Hoorn, and Lars Grunske. Increasing Dependability of Component-based Software Systems by Online Failure Prediction. 2013. (Submitted for review).

[SLM10] Felix Salfner, Maren Lenk, and Miroslaw Malek. A survey of online failure prediction methods. *ACM Computing Surveys*, 42(3):10:1–10:42, March 2010.

[vH13] André van Hoorn. *Model-Driven Online Capacity Management for Component-Based Software Systems*. Kiel, Germany, 2013. Dissertation (work in progress), Faculty of Engineering, Kiel University.

[vHWH12] André van Hoorn, Jan Waller, and Wilhelm Hasselbring. Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 247–248. ACM, April 2012.