2014 Tenth European Dependable Computing Conference

Increasing Dependability of Component-based Software Systems by Online Failure Prediction

Teerat Pitakrat, André van Hoorn, Lars Grunske
University of Stuttgart, Institute of Software Technology
Universitätsstraße 38, 70569 Stuttgart, Germany
Phone/Fax: +49 711 685 88-248/-472
{pitakrat,van.hoorn,grunske}@informatik.uni-stuttgart.de

***Abstract*—Online failure prediction for large-scale software systems is a challenging task. One reason is the complex structure of many—partially inter-dependent—hardware and software components. State-of-the-art approaches use separate prediction models for parameters of interest or a monolithic prediction model which includes different parameters of all components. However, they have problems when dealing with evolving systems. In this paper, we propose our preliminary research work on online failure prediction targeting large-scale component-based software systems. For the prediction, three complementary types of models are used: (i) an architectural model captures relevant properties of hardware and software components as well as dependencies among them; (ii) for each component, a prediction model captures the current state of a component and predicts independent component failures in the future; (iii) a system-level prediction model represents the current state of the system and—using the component-level prediction models and information on dependencies—allows to predict failures and analyze impacts of architectural system changes for proactive failure management.**

Keywords*—*dependability, online failure prediction, component-based software systems, monitoring, models at runtime

I. Introduction

High dependability—including quality-of-service (QoS) characteristics like performance, availability, and reliability—is one of the ultimate goals for software systems during operation. The architecture of the system is designed to provide high reliability with redundancy for cases that some parts of the system fail. The approach has been shown to perform well to a certain level. However, there are still situations where the designed redundancy fails and cannot prevent the whole system from failing.

One solution to increase system dependability is *online failure prediction* [1] which tries to predict possible pending failures at runtime and preventing them from occurring. The approach collects observable parameters of the system during the training phase and creates a prediction model which is used at runtime to classify the current system state and predict whether it is going to fail. The existing approaches can be divided into two groups: (i) predicting one or more QoS measures based on selected historic observations of system parameters, e.g., response time prediction employing time series forecasting [2]; (ii) predicting system events, e.g., failure prediction using machine learning techniques [3]. However, in real situations, the systems in production tend to evolve over time. The changes include replacement of hardware components, software updates, system reconfiguration, or system architecture changes. These changes pose a problem to the prediction models as the models are created with the collected data as a whole and cannot be incrementally updated to keep up with the real system.

In this paper, we propose a compositional approach to improve online failure prediction in component-based software systems. Instead of considering all parts of the system together and building one large prediction model, we create component-level prediction models which are combined to form a system-level prediction model. Our approach builds on model-based QoS prediction using architecture- and analysis-oriented models [4], [5]. The advantages of our approach are: (i) different established prediction techniques can be used for different components, (ii) runtime changes to the system can be incorporated into the prediction model, and (iii) the prediction results include not only whether the system will fail, but also when it will fail, and where the root cause might be.

The remainder of this paper is organized as follows. Section II outlines our envisioned approach and a running example used in this paper. Section III focuses on the construction of the prediction models. The different usage scenarios are detailed in Section IV, including the use and evolution of the prediction models at runtime. Related work is discussed in Section V. Section VI draws the conclusions and outlines future work.

II. Envisioned Approach

One of the challenges when predicting failures in a large system is that there are a number of components with many parameters and inter-dependencies to be considered. Including all of them when building the prediction model results in a model that is difficult to update because the concepts contained in the model often do not correspond to the entities in the software system architecture. Our proposed online failure prediction approach eases this problem by decomposing the prediction model into component models using information obtained from an architectural model of the system. The approach aims to improve system dependability by providing early warnings about pending failures with probabilities, expected time, and the probable root causes. The administrators can then choose to proactively replace the components which are predicted to be the cause of a pending system failure and preventing it from occurring. The following sections II-A and II-B give a brief description of the architectural model and an overview of our prediction approach for component-based systems.

978-1-4799-3804-9/14 $31.00 © 2014 IEEE
DOI 10.1109/EDCC.2014.28

A. Architectural Model

An architectural model of a component-based software system describes how the components are deployed and connected together to form a fully functional system. One of the advantages of the model is that it allows each component to be substituted with another component as long as the new one provides at least the same services. In our work, we plan to use the Palladio Component Model (PCM) [4].

Figure 1 provides an architectural view of an example system including component assembly, deployment, and behavior. The system comprises three software components, C_1, C_2, and C_3, being deployed on two hardware components, H_1 and H_2. In addition, the component C_1 provides two services, S_1 and S_2, which can serve users at the system boundary. The model shows the dependencies between components, e.g., C_1 depends on C_2 and C_3; C_3 depends on H_2. Note that the model can also describe composite structures of hardware components, e.g., CPU, memory, or hard drives. However, they are excluded for simplicity reasons. The implementation of service S_1 by component C_1, referred to as $C_1.s_1$, uses hardware services provided by H_1 and software service provided C_2. In the same manner, service $C_1.s_2$ uses H_1 and C_3 (not depicted in the figure). We assume that the architectural model exists either by being manually created or being learned from the system observation at runtime.

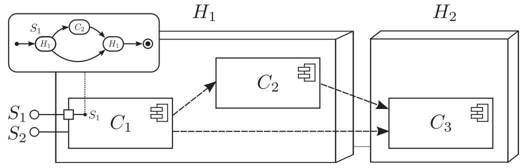


Fig. 1: Architectural component model (excerpt)

In addition to the structure and dependencies of the system, the architectural model also provides an insight regarding how a failure of one component may cause others to fail, creating a chain of failures which propagates to the system boundary. For example, a failure of C_2 can render some services of C_1 unavailable to the users, or a failure of H_2 which hosts C_3 can cause the whole system to fail. This information is useful when analyzing the root cause of the failure and the failure propagation. However, online failure prediction needs to answer further questions, such as, when a component will fail and whether it will affect the functionality of the whole system.

B. Prediction Models

The prediction models in our proposed approach are divided into two parts: component-level prediction models, which predict failures of components, and a system-level prediction model, which predicts failures of the whole system.

1) Component-level Prediction Model: A component-level prediction model aims to represent and predict the states of a component, such as the state that it is functioning perfectly, working in a degraded mode, or when it experiences a severe problem. A prediction model exists for each of the architectural components. The prediction models of different components do not necessarily have to be created using the same technique. For instance, time series forecasting can be used as a prediction model for components that provide time series data, and machine learning or data mining techniques can be employed when a set of observed data is available but their relationship to the component state is not obvious. The prediction results are returned through a unique interface providing the current state and the probability distribution for being in a future state. Each individual model can also be refined by measurement data at runtime to accurately reflect its current operational profiles.

2) System-level Prediction Model: The system-level prediction model is a combination of the states of the component models and the component dependencies extracted from the architectural model. We aim to use a transition model in which each state represents a set of component failures and the transitions represent the failure probability distribution of the next component over time. In this paper, we employ a 2-order continuous-time Markov chain (CTMC). The model has an advantage over a standard Markov chain in that the transition to the next state depends not only on the current state but also the previously visited states. This property of the model can represent the dependencies between components and can also predict whether a failure of one component will cause other components or the entire system to fail. Figure 3 depicts the model for the running example which will be detailed in Section III-B.

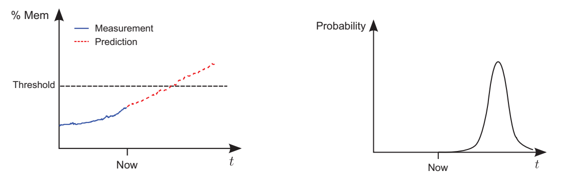
III. CONSTRUCTION OF PREDICTION MODELS

The prediction models of each architectural component in the system can be created independently and later combined into a system-level prediction model. The construction of the component and system prediction models are described in the following Sections III-A and III-B. As previously mentioned, we make no specific assumption regarding the component-level prediction models. We simplify by using time series and machine learning prediction models.

A. Component-level Prediction Model

The component model can be created by analyzing the observable parameters of the component which can be the metrics for dependability characteristics, e.g., availability or performance. For instance, if the component C_3 in Figure 1 is a database, the observable parameter can be its memory consumption. A suitable prediction model for this component is therefore time series forecasting as the memory consumption exhibits the characteristic of time series data. The time series model can provide the current state by checking whether the value is below or above a certain threshold, and predict the next state by extrapolating the time series data into the future. The model and the result are illustrated in Figure 2.

Another example is the case where C_2 provides a functional service and logs events into a log file. The log records can be analyzed to show the relationship between log sequences and the component status using machine learning algorithms, such as, support vector machine or decision tree. In our previous work [6], machine learning algorithms are used to classify hard drives as being in a healthy or failing states and predict when the transition is going to happen.



(a) Prediction model (b) Prediction result

Fig. 2: Component-level prediction model for C_3 and the probability of memory consumption exceeding the threshold

B. System-level Prediction Model

The system-level model is constructed by combining failure states of all component models to form a 2-order CTMC. The process can be divided into two steps; creating the states and specifying the transitions between them.

1) States: The states of the Markov chain represent the failure status of the components from the system perspective. If one component of the system fails, the system transitions to the state that this component is failing. As a consequence, this failure may cause other components to fail due to the component dependency. The chain of component failures can propagate until it causes a service failure at the system boundary which can be perceived by the users.

To model the failure dependency, we use a 2-order CTMC which takes into account the current and the previous component failures in that particular sequence. For example, if the component C_3 in Figure 1 fails, which consequently causes component C_2 to fail, the Markov chain will transition from a healthy state, in which no component fails, to the state where C_1 fails, and then to the state where C_1 and C_2 fail. The number of component failures in one state is bounded by the number of order of the chain. A higher order chain can represent longer sequences of component failures, however, the number of states also increases exponentially, i.e., *n*-permutation of the number of component failures.

In order to reduce the storage complexity of the 2-order CTMC, we incorporate the information obtained from the architectural model and create only the reachable states, that is, the failure sequences that cannot occur will not appear in the chain. Table I presents the component dependency table extracted from the architectural model explained in Section II-A. The number of dependencies between components are greatly reduced compared to when the system architecture is unknown. From the dependency table, a 2-order Markov chain can be created which is illustrated as an equivalent 1-order Markov chain in Figure 3. The entry point of the chain is the healthy state—from which it can transition to the states with one component failure, e.g., H_1 or C_3, and then to the states with two failures, e.g., H_1C_2 or C_3C_2. The maximum number of component failures in each state in this case is limited to two which is the number of the order of the chain. Note that there are two additional states in the chain, S_1 and S_2, which represent the failures of services 1 and 2 perceived by the users at the system boundary.

TABLE I: Dependencies between architectural components, e.g., C_3 depends on H_2

	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								

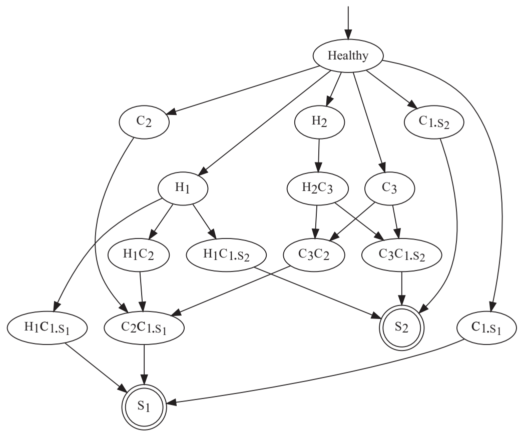


Fig. 3: System-level prediction model where each state represents a sequence of the last two component failures

2) Transitions: Each component-level prediction model provides prediction results, including time and probability of the component failure, which can be used to specify the transition between states. For example, the transition probability from state H_2 to H_2C_3 is indicated by the result of the component prediction model of C_3, given that H_2 has already failed. The probability of the failure of H_2 leading to a failure of service S_1 can then be calculated by combining the prediction results of C_3, C_2, and C_{1,S_1} into one transition probability distribution.

IV. Usage Scenarios

The prediction model in our approach allows different usage scenarios which are described in the following subsections.

A. Predicting Failures at Runtime

At runtime when the prediction model is deployed, the model begins with the system in healthy state representing no component failure. When one of the component models predicts that it is going to fail, the system model assumes that the prediction could be correct and temporarily transitions to the state of that component failure. From that state, the model computes the probability of other component failures from the component-level prediction models and searches for the paths that can lead to system failures, using a standard CTMC solver [7]. If the probability of the system transitioning into a failure state is above a pre-defined threshold, a failure warning is issued. The prediction result includes which service of the system is going to fail, what is the expected time to failure, and the possible paths in the component failure chain. This information can be passed to the system administrators or a failure management module which can prevent the problem by diagnosing the failure chain and fixing or replacing the problematic components. The transition model used in this paper can be extended to support concurrent states, for example, using Petri net concepts. Using the same mechanism we can analyze the impact of system changes—e.g., component replacement for proactive failure management—on QoS.

B. Updating and Refining Prediction Model

One of the benefits of the Markov chain in our approach is that the states can be added or removed at runtime. If a new component is added to the system, the failure states of that component can be added to the chain according to the information obtained from the architectural model. If a component is removed, the states that contain this component are removed from the chain. Furthermore, the prediction results of the component models can become unrealistic due to an incorrect estimation during the learning phase, or component reconfiguration at runtime. The models that make frequently incorrect predictions can be individually retrained based on the data observed at runtime by monitoring tools, such as Kieker [8].

V. Related Work

Related work comes from the areas of online failure prediction [1] and architecture-based QoS evaluation [5]. Our proposed approach combines both areas and provides more information not only when the failure will occur but also the whereabouts.

The techniques of online failure prediction can be grouped into two categories. The first category aims at selecting crucial parameters of the system and building a prediction model for each of them. An example of this method is predicting when the response time of a service will exceed an acceptable threshold using time series forecasting techniques. The techniques in this group particularly model the parameter of interest by considering the factors which have direct influences on it (see, e.g., [2], [9]). On the other hand, the techniques in the second category aggregate various available parameters and system events whether or not they have direct effect on the system dependability. Machine learning or data mining techniques are then applied to the collected information during the training phase to discover the relationship between system events and QoS problems. The discovered relationship is regarded as a prediction model and is employed to classify the system at runtime as healthy or failing (see, e.g., [3], [6]). These well-established techniques can be utilized in our approach serving as component-level prediction models.

Architecture-based QoS evaluation approaches use design-oriented system models which are completed by QoS-relevant information [5]. Analytical or simulative model solvers are applied to obtain QoS measures of interest. They focus on different QoS characteristics, e.g., performance [4], reliability [10], resource efficiency [11]. Originally, these approaches aimed at design-time predictions. However, during the past years approaches emerged which use model-based QoS evaluation at runtime. Following the aforementioned general approach of architecture-based QoS models, our focus is on incorporating analysis models for online failure prediction.

VI. Conclusion

Online failure prediction is one of the solutions to improve system dependability by predicting possible QoS problems and issuing a warning before they occur. However, creating a prediction model is a challenging task especially when the system is large and contains a number of components with inter-dependencies. We proposed an approach to construct the prediction model by breaking it down into component models. Each model is a technique-independent predictor responsible for predicting failures of one component. The system-level prediction model is built by combining the states of component models into a transition model using knowledge obtained from the architectural model. The prediction result of our proposed model provides information of the failure including the time, the probability, and the components that might be the root cause of the failure. Our approach also allows the structure of the system-level prediction model to be updated at runtime if components are added or removed.

In our future work, we aim to extend our online prediction approach to a reusable framework. A preliminary description of the framework, built on Kieker [8] and Palladio [4], can already be found in our previous work [12]. We will investigate the selection of a suitable model for the system-level prediction model and the automation process of the model construction—both component- and system-level prediction models. We plan to evaluate our approach by a combination of real-world data, simulation, and lab experiments.

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] A. Amin, A. Colman, and L. Grunske, “An approach to forecasting QoS attributes of web services based on ARIMA and GARCH models,” in *Proc. ICWS ’12*. IEEE, 2012, pp. 74–81.

[3] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. K. Sahoo, “Bluegene/l failure analysis and prediction models,” in *Proc. DSN ’06*, 2006, pp. 425–434.

[4] 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.

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

[6] T. Pitakrat, A. van Hoorn, and L. Grunske, “A comparison of machine learning algorithms for proactive hard disk drive failure detection,” in *Proc. ISARCS ’13*. ACM, 2013, pp. 1–10.

[7] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, “Prism: A tool for automatic verification of probabilistic systems,” in *Proc. TACAS ’06*. Springer, 2006, pp. 441–444.

[8] A. van Hoorn, J. Waller, and W. Hasselbring, “Kieker: A framework for application performance monitoring and dynamic software analysis,” in *Proc. ICPE ’12*. ACM, 2012, pp. 247–248.

[9] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, “Analysis of software aging in a web server,” *IEEE Transactions on Reliability*, vol. 55, no. 3, pp. 411–420, 2006.

[10] F. Brosch, “Integrated software architecture-based reliability prediction for IT systems,” Ph.D. dissertation, Karlsruher Institute of Technology, Karlsruhe, Germany, 2012.

[11] S. Kounev, F. Brosig, N. Huber, and R. Reussner, “Towards self-aware performance and resource management in modern service-oriented systems,” in *Proc. SCC ’10*. IEEE, 2010, pp. 621–624.

[12] T. Pitakrat, “Hora: Online failure prediction framework for component-based software systems based on Kieker and Palladio,” in *Proc. KP-Days ’13*. CEUR-WS.org, 2013, pp. 39–48.